MACHINE LEARNING METHODS FOR 3D OBJECT CLASSIFICATION AND SEGMENTATION

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

TRUC DUC LE

Dr. Ye Duan, Thesis Supervisor

JULY 2018

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

MACHINE LEARNING METHODS FOR
3D OBJECT CLASSIFICATION AND SEGMENTATION

presented by Truc Duc Le,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Ye Duan

Dr. Kannappan Palaniappan

Dr. Yi Shang

Dr. Jeffrey Uhlmann

Dr. Zhihai He

ACKNOWLEDGMENTS

I am most grateful to my supervisor, Prof. Ye Duan, for his guidance during the time I have been completing my doctoral degree. Prof. Ye Duan is a great supervisor who constantly gives useful advices and ideas. He has spent a substantial amount of time on my research progress. He has also inspired and encouraged me endless times during my study. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, the completion of this thesis would not have been possible. I am very grateful for the humane environment the Computer Graphics and Image Understand lab has provided during the years of my study in Mizzou.

I would like to thank all committee members including Prof. Kannappan Palaniappan, Prof. Yi Shang, Prof. Jeffrey Uhlmann and Prof. Zhihai He for being helpful reviewers of my research and for their valuable time on fruitful discussion during my study.

I would like to thank all of my labmates and my friends (Brittany Morago, Giang Bui, Qing Lei, Huanhuan Xia, Xu Wang, Yuyan Li, Fan Gao) who have proposed and discussed ideas on my work and their time for providing feedbacks after listening to my presentations.

Lastly, I would like to thank my family for their greatest-ever suuport, encouragement as I completed my Ph.D..

TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . . . **ii**

LIST OF TABLES . . . . . **vi**

LIST OF FIGURES . . . . . **viii**

ABSTRACT . . . . . **xv**

CHAPTER . . . . .

1 Introduction **1**

1.1 Contributions 2

1.2 Organization 4

2 A Primitive-based 3D Segmentation Algorithm for Mechanical CAD Models 6

2.1 Introduction 7

2.2 Other Related Works 12

2.3 Our Approach 16

2.3.1 Patch-based Over-segmentation 16

2.3.2 Major Direction Estimation 17

2.3.3 Plane Detection 20

2.3.4 Generation of Over-complete Set of Circular Primitives 20

2.3.5 Set Cover Optimization 35

2.3.6 Boundary Refinement 36

2.4 Experimental Results 36

2.4.1 Limitation and Future Work 47

2.5 Conclusion 48

3 A Multi-view Recurrent Neural Network for 3D Mesh Segmentation 49

3.1 Introduction . . . 49
3.2 Related Work . . . 53
3.3 Background on Recurrent Neural Network . . . 56
3.4 Multi-view Recurrent Neural Network (MV-RNN) . . . 57
3.4.1 Input . . . 57
3.4.2 CNN Module . . . 58
3.4.3 LSTM Module . . . 59
3.4.4 Training . . . 62
3.4.5 Back Projection to 3D and Post-processing . . . 63
3.5 Evaluation . . . 68
3.5.1 Limitation . . . 71
3.6 Conclusion . . . 71
4 REDN: A Recursive Encoder-Decoder Network with Skip-Connections for Edge Detection . . . 74
4.1 Introduction . . . 75
4.2 Related Work . . . 77
4.3 Recursive Encoder–Decoder Network with Skip-Connections . . . 80
4.3.1 Training Formulation . . . 81
4.3.2 Testing Formulation . . . 82
4.3.3 Network Architecture . . . 83
4.3.4 Loss function . . . 84
4.3.5 Implementation . . . 85
4.4 Evaluation . . . 87
4.4.1 Datasets . . . 87
4.4.2 Visual Comparison . . . 92

4.4.3 Quantitative Comparison . . . 93
4.4.4 Cross-dataset Evaluation . . . 93
4.5 Conclusion . . . 94
5 PointGrid: A Deep Network for 3D Shape Understanding . . . 95
5.1 Introduction . . . 95
5.2 Related Work . . . 98
5.3 PointGrid . . . 101
5.3.1 Input Layer . . . 102
5.3.2 Classification Network . . . 103
5.3.3 Segmentation Network . . . 103
5.3.4 Implementation details . . . 104
5.4 Experiments . . . 106
5.4.1 Shape Classification . . . 106
5.4.2 Object-part Segmentation . . . 112
5.5 Conclusion . . . 114
6 Summary . . . 115
BIBLIOGRAPHY . . . 116
VITA . . . 141

LIST OF TABLES

Table Page

2.1 Comparison of three algorithms on our benchmark. 32
2.2 Comparison of primitive quality over processed models 40
2.3 Timings statistics (in seconds) of each step in our method on processed models 41
2.4 Numerical comparison of primitive quality for Fig. 2.16 42

3.1 The Rand Index scores of segmentation for each category with different methods. Smaller is better. 64
3.2 Average cut discrepancy, hamming distance, consistency error scores of segmentation for each category with different methods. Smaller is better. . . 65

4.1 Datasets and Parameters 87
4.2 BSDS500 [1] test evaluation 89
4.3 NYUD-v2 [2] test evaluation 90
4.4 Pascal Context [3] test evaluation 90
4.5 Cross-dataset performance of REDN 90

5.1 Object classification results on ModelNet40 [4]. 105
5.2 Object classification results on ShapeNet-55 [5]. 105
5.3 Accuracy of PointGrid's alternative structures on ModelNet40 [4]. 105
5.4 Object-part segmentation results on ShapeNet-part [6]. 107

5.5 Average testing time on ModelNet40 [4]. 113

LIST OF FIGURES

Figure Page

2.1 Failure case of RANSAC-based approach [7] and GlobFit [8]. **First column**: Input model (point cloud); **Second column**: Segmentation from RANSAC-based approach is locally optimal (black lines indicate the axes of the primitives); **Third column**: GlobFit's optimization can align the parallel primitives but increase the fitting error and fail to recover the correct segmentation; **Last column**: Our segmentation. 8

2.2 Flowchart of our segmentation approach. 12

2.3 Illustration of our approach on the *mechanical-part* model. **A**: Input point cloud; **B**: Patch-based over-segmentation; **C**: Major orientation estimation; **D**: Planes are detected from over-segmented patches from **B** and the major orientations from **C**; **E**: Cross-sectional slicing of the model along each orientation and circle detection on each slice; **F**: *top*: Group adjacent circles into connected components and for each component, fit line to the circle centers to estimate the circular primitive's axis location (and possibly refine the axis orientation); *bottom*: circular primitives are generated from the profile curve analysis (best viewed with electronic zoom-in or see Fig. 2.13 for more details); **G**: Set cover optimization; **H**: Final segmentation result after boundary refinement (*left*: segmentation colored by primitives; *right*: segmentation colored by primitive types). 13

2.4 Crest lines prevent the patches from spanning multiple primitives. **Left**: patches (without crest lines); **Middle**: detected crest lines shown in black [9]; **Right**: patches from *constrained* region growing. 17

2.5 Partition of a sphere into 1,600 regions by sub-dividing the spherical coordinates (**left**) and the equal-area partitioning [10] (**right**). 17

2.6 Major direction estimation. **A**: Input model and the estimated major directions; **B**: Projection of point cloud's normals (black dots) onto the Gaussian sphere; **C**: the poles (in pink, dodger blue and dark green) of the normals' distribution, **D**: mapping of randomly chosen triples (a triple of three black dots in B corresponds to a gray dot); **E**: poles (in pink, dodger blue, dark green, brown and purple) in the mapped space; **F**: corresponding rings (in pink, dodger blue, dark green, brown and purple) of the normals' distribution. The final model's major directions are shown in **A**. 18

2.7 The estimated major orientations for all tested models. Note that the colored axes represent orientations only, not location. See Fig. 2.1 (last column) for our detected axes. 20

2.8 Illustration of our circle detection (*best view in electronic version*). *From left to right, top to bottom*: input image; edge map; line extraction; circles' centers generated from pairs of lines; circles' centers candidates; final detected circles. 25

2.9 Comparison of the circle detection methods on synthesized images (images on rows 1, 2, 3, and 5 are copied from [11]). *First column*: input image; *Second column*: edge map; *Third column*: our approach; *Fourth column*: the Circular Hough Transform (CHT); *Last column*: the Randomized Circle Detection (RCD). 28

2.10 Comparison of the circle detection methods on natural images (images on rows 4, 5 and 6 are copied from [12]). *First column*: input image; *Second column*: edge map; *Third column*: our approach; *Fourth column*: the Circular Hough Transform (CHT); *Last column*: the Randomized Circle Detection (RCD). 29

2.11 The effect of two parameters T_r and T_c. *First row*: input image (from CDBD); *Second row*: edge map; *Third row*: our circle detection with T_r and T_c are set to default values (i.e. $T_r = 0.6$ and $T_c = 180^\circ$); *Last row*: our circle detection with $T_r = 0.4$ and $T_c = 60^\circ$. 31

2.12 The effect of the edge detection on our circle detection. *Top left*: input image (from CDBD); *Second column*: Canny's edge detector [13] on top and Sobel's edge detector [14, 15] at bottom; *Third column*: our circle detection running on two corresponding edge maps. 31

2.13 Two examples of profile curve analysis. **A**: Input models and detected orientations; **B**: Model's slicing along the orientation; **C**: Group of adjacent detected circles; **D**: Profile curve from **C**; **E**: Segmentation of profile curve into line segments and circular arc; **F**: Detected primitives in 3D. 33

2.14 Projection of the point cloud onto the detected primitives obtained from our segmentation algorithm. 37

2.15 Comparison primitive type recognition between our segmentation and RANSAC-based segmentation [7]. **Odd columns**: RANSAC's result; **Even columns**: Our result. 39

2.16 Comparison of the robustness between our segmentation and RANSAC-based segmentation [7] on the *joint* model. Noise (on both points' positions and normals) increases from left to right, starting with a clean model. 43

2.17 Illustration of segmenting profile curve involving surface of revolution. **A**: A *bishop* model with detected axis; **B**: Cross-sectional slicing; **C**: Profile curve from detected circles; **D**: Color map of the (approximated) profile curve's normal variation where critical points are marked bigger; **E**: Segmentation of profile curve; **F**: Profile curve fitted by line segments, circular arcs and cubic splines; **G**: Segmentation result; **H**: Segmentation colored by primitive type. 44

2.18 Our segmentation algorithm for some models with surface of revolution. **Top**: Random colors for different parts. **Bottom**: Segmentation colored by primitive type (surface of revolution is purple, others are colored similar to Fig. 2.3H). 45

2.19 Effect of various values of the slice's thickness τ. **Leftmost**: too fine slicing makes the projection image too sparse for circle detection; **Rightmost**: too coarse slicing makes the profile curve too sparse for circular primitive detection. 45

3.1 Given a 3D model, we try to detect boundary between segments by using multi-view approach. We apply non-maximum suppression [16] to the MV-CNN results shown on the second row for visualization. The main drawback of MV-CNN is its inconsistency across multiple views (e.g. the elbow and arm regions). On the other hand, our MV-RNN could correlate multiple views and generate more coherent results. 52

3.2 Overview of our MV-RNN approach. Given an input 3D mesh model, we render it with a sequence of ordered viewpoints. Each of view is passed through an identical (shared weights) CNN to obtain a boundary probability map, which is correlated by a two-layer LSTM followed by a fully connected layer. The consistent edge images from multiple views are unprojected back to 3D followed by a region growing and CRF for boundary smoothing. 53
3.3 Multiple views ordered in a helix-like sequence. 59
3.4 LSTM learning process (only four views are shown due to space limit). **First row**: Input shaded images to a CNN. **Second row**: Outputs from CNN. **Third to Tenth rows**: Edges returned from LSTM during training. **Last row**: Ground truth edges. 60
3.5 Representative segmentation results produced by our MV-RNN on PSB dataset. 61
3.6 Performance plots of different segmentation algorithms with respect to four evaluation metrics. Lower value is better. 61
3.7 Comparison of segmentation algorithms. 66
3.8 More comparisons of segmentation algorithms. 67
3.9 The Rand Index with respect to the number of views. We choose $K = 60$ as a reasonable trade-off between accuracy and time/memory usage. 71
3.10 Limitation of our approach. The area under the torso is occluded and hence the left and right thighs are not separated although our MV-RNN can detect 2D edges correctly in all views. 72

4.1 The architecture of our Recursive Encoder-Decoder Network with Skip Connections (REDN). Encoder-decoder network is at the heart of our design which consists of DenseNet blocks. There are four skip-connections that connects one layer of the encoder to a corresponding layer of the decoder. The feedback connection enables a deeper network with no extra parameters. 80

4.2 Original images (top row) are augmented with Gaussian noise to force the network to extract stronger edges. 82

4.3 Ground truth edge image generation. From an input image (first row) with ground truth semantic segmentation (second row), we identify all boundary pixels (third row) and then apply image thinning (e.g. `MATLAB`'s `bwmorph`) to obtain the ground truth edge image (last row). 86

4.4 Side-by-side comparison of edge detection algorithms. All edge images are originally returned by the algorithms before non-maximum suppression. . . 88

4.5 The recursive network ($L = 2$) improves the results of encoder-decoder network with skip-connections ($L = 0$) by cleaning noisy edges and enhance stronger ones. 91

5.1 A 2D illustration of the proposed PointGrid, which is a hybrid 3D shape representation between discrete points (b) and volumetric grid (c) and (d). Points within each grid cell will be quantitized (e), so that both occupied (yellow) and empty (blue) cells have exactly K points (f). 96

5.2 The architecture of PointGrid. Starting from the grid obtained from our sampling strategy, both classification and segmentation networks share the feature extraction (encoder) part. Segmentation network uses skip connections to preserve information of different hierarchical levels. All convolutions, deconvolutions and fully connected layers include batch normalization and ReLU (except object category and object-part segmentation layers). The notion 32@16x16 means there are 32 convolutional filters and the spatial dimension is 16×16. The network is visualized in 2D. 101

5.3 Some wrongly classified models. Predicted and actual labels are italicized and highlighted, respectively. 108

5.4 Visualization of object saliency. Magnitude of the gradient of the probability w.r.t. input grid is populated to points. Red indicates highly salient regions. 109

5.5 Comparison between PointGrid and PointNet on object-part segmentation. This result is based on grid size $16 \times 16 \times 16$ and $K = 4$. 110

ABSTRACT

Object understanding is a fundamental problem in computer vision and it has been extensively researched in recent years thanks to the availability of powerful GPUs and labelled data, especially in the context of images. However, 3D object understanding is still not on par with its 2D domain and deep learning for 3D has not been fully explored yet. In this dissertation, I work on two approaches, both of which advances the state-of-the-art results in 3D classification and segmentation.

The first approach, called MVRNN, is based multi-view paradigm. In contrast to MVCNN which does not generate consistent result across different views, by treating the multi-view images as a temporal sequence, our MVRNN correlates the features and generates coherent segmentation across different views. MVRNN demonstrated state-of-the-art performance on the Princeton Segmentation Benchmark dataset.

The second approach, called PointGrid, is a hybrid method which combines points and regular grid structure. 3D points can retain fine details but irregular, which is challenge for deep learning methods. Volumetric grid is simple and has regular structure, but does not scale well with data resolution. Our PointGrid, which is simple, allows the fine details to be consumed by normal convolutions under a coarser resolution grid. PointGrid achieved state-of-the-art performance on ModelNet40 and ShapeNet datasets in 3D classification and object part segmentation.

Chapter 1

Introduction

3D shape understanding is a fundamental problem in both computer graphics and computer vision, especially 3D classification and segmentation. A variety of applications in robotics, autonomous driving car, etc. are directly beneficial from 3D classification and segmentation. There are two types of 3D data, engineering computer aided design (CAD) objects and free-form objects. The former usually consists of predefined set of geoemtric primitives such as planes, spheres, cylinders, cones, tori while the later does not limit to any geometric shape but usually has a semantic meaning (e.g. head, torso, hand, leg of a human model). Researchers have explored different methods to segment the 3D models. In the early stage, hand-crafted features such as curvature, planarity, shape diameter function are used, but the success is only limited to a specific set of objects and cannot be generalized to different kinds of models. Later, thanks to better GPUs and the availability of labeled 3D data, data driven approaches demonstrate their power over traditional methods, especially with the recent advances of deep learning.

In this disseration, we process two types of data separately. For the CAD data, we propose a novel hypothesis generation and verification approach where the former generates an over-complete set of geometric primitives by utilizing dimension reduction to enhance detection robustness and the later selects the final segmenation via set cover optimization.

For the free-form objects, we propose two different solutions, MVRNN and PointGrid. MVRNN is a multi-view based approach where it renders the object into multiple views, segments it in each view in 2D and projects the results back to 3D. The key success of MVRNN is the usage of recurrent neural network to correlate the segmentation accress various views, which makes the segmentation consistent. PointGrid is designed to consume directly point clouds by combining points and grid, a hybird approach which facilitates construction of 3D convolution neural network to achieve high segmentation accuracy.

1.1 Contributions

This dissertation presents the following contributions to the area of computer vision.

1. We present a novel segmentation algorithm for mechanical CAD models (represented by either mesh or point cloud) constructed from planes, cylinders, cones, spheres, tori and easily extendable to surfaces of revolution. Our proposed approach differs from existing techniques in the following aspects. First, by assuming that common mechanical models only have a limited number of dominant orientations that their primitives are either parallel or orthogonal to, we narrow down the search space for detecting the primitives to the automatically estimated major orientations of the input model. Second, we employ a dimension reduction method which transforms the problem of detecting 3D primitives into the classical 2D problems such as circle and line detection in images. Third, we generate an over-complete set of primitives and formulate the segmentation as a set cover optimization problem. We demonstrate our method's robustness to noise and show that it compares favorably with state-of-the-art solutions such as the RANSAC-based [7] and GlobFit [8] approaches on many synthetic and real scanned examples.

2. We propose a new method to detect circles from images. Our approach consists of two steps, circle candidate generation and verification. In particular, we start by line

segment detection from the edge image, then use pairs of detected line segments to generate an initial set of circle candidates. Non-maximum suppression is applied to this initial set via mean-shift based clustering while the cluster centers are considered as the final set of circle candidates. These circle candidates are then individually verified by both the ratio between number of edgels to their radii and their *completeness*. We also create CDBD, the first benchmark dataset for circle detection which consists of more than a thousand images with ground truth circles labeled by human.

3. We propose a multi-view recurrent neural netowrk (MV-RNN) approach for 3D mesh segmentation. Our architecture combines the convolutional neural networks (CNN) and a two-layer long short term memory (LSTM) to yield coherent segmentation of 3D shapes. The imaged-based CNN are useful for effectively generating the edge probability feature map while the LSTM correlates these edge maps across different views and output a well-defined per-view edge image. Evaluations on the Princeton Segmentation Benchmark dataset show that our framework significantly outperforms other state-of-the-art methods.

4. We introduce REDN: A Recursive Encoder-Decoder Network with Skip-Connections for edge detection in natural images. The proposed network is a novel integration of a Recursive Neural Network with an Encoder-Decoder architecture. The recursive network enables us to increase the network depth without increasing the number of parameters. Adding skip-connections between encoder and decoder helps the gradients reach all the layers of a network more easily and allows information related to finer details in the early stage of the encoder to be fully utilized in the decoder.

5. We propose the PointGrid, a 3D convolutional network for processing point clouds that incorporates a constant number of points within each grid cell thus allowing the network to learn higher order local approximation functions that could better represent the local geometry shape details. With experiments on popular shape recognition

benchmarks, PointGrid demonstrates state-of-the-art performance over existing deep learning methods on both classification and segmentation.

1.2 Organization

This dissertation is organized as following in the next chapters.

1. **A Primitive-based 3D Segmentation Algorithm for Mechanical CAD Models**: The goal of this work is to derive a robust algorithm for segmenting man-made engineering models into geometric primitives such as planes, cylinders, cones, spheres and tori. There are two publications related to this work.

    - Truc Le, Ye Duan, "Circle Detection on Images by Line Segment and Circle Completeness", IEEE Conference on Image Processing (ICIP), 2016, pp. 3648–3652.
    - Truc Le, Ye Duan, "A primitive-based 3D segmentation algorithm for mechanical CAD models", Computer Aided Geometric Design (CAGD), 2017, pp. 231–246.

2. **A Multi-view Recurrent Neural Network for 3D Mesh Segmentation**: This work advances the 3D segmentation by following the multi-view segmentation approach and by treating multi-view images as a temporal sequence. There is a journal publication for this work.

    - Truc Le, Giang Bui, Ye Duan, "A Multi-view Recurrent Neural Network for 3D Mesh Segmentation", Computers & Graphics, 2017, pp. 103–112.

3. **REDN: A Recursive Encoder-Decoder Network with Skip-Connections for Edge Detection**: This work advances the edge detection by combining a recursive connection with an encoder-decoder network.

4. **PointGrid: A Deep Network for 3D Shape Understanding**: This work advances the 3D classification and segmentation of point cloud by a hybrid model which combines the advantages of regular structure (grid) with fine detail (points). There is a conference publication for this work.

    - Truc Le, Ye Duan, "PointGrid: A Deep Network for 3D Shape Understanding", to appear as spotlight at IEEE Conference on Computer Vision and Pattern Recognition, 2018.

Chapter 2

A Primitive-based 3D Segmentation Algorithm for Mechanical CAD Models

This work presents a novel segmentation algorithm for mechanical CAD models (represented by either mesh or point cloud) constructed from planes, cylinders, cones, spheres, tori and easily extendable to surfaces of revolution. Our proposed approach differs from existing techniques in the following aspects. First, by assuming that common mechanical models only have a limited number of dominant orientations that their primitives are either parallel or orthogonal to, we narrow down the search space for detecting the primitives to the automatically estimated major orientations of the input model. Second, we employ a dimension reduction method which transforms the problem of detecting 3D primitives into the classical 2D problems such as circle and line detection in images. Third, we generate an over-complete set of primitives and formulate the segmentation as a set cover optimization problem. We demonstrate our method's robustness to noise and show that it compares favorably with state-of-the-art solutions such as the RANSAC-based [7] and GlobFit [8] approaches on many synthetic and real scanned examples.

2.1 Introduction

Computer-aided design (CAD) models are geometric models which are a conceptually higher level and more accurate representation of an object than mesh or point cloud. In normal cases, engineers and designers create CAD models for printing, machining or other manufacturing operations in many areas including automotive, shipbuilding, aerospace industries, industrial and architectural design, prosthetics, and many more. However, during the process, the original CAD models may be lost so people would like to reconstruct them from meshes or point clouds being scanned from the real objects. The reconstruction process refers as reverse engineering and it has ample applications [17, 18, 19]. The reverse engineering saves engineers and designers time in a way that they do not need to create a whole CAD model from scratch but instead get it reconstructed from a real model followed by modification.

Segmentation of a mesh or point cloud is at the heart of reverse engineering and it is also a fundamental problem of computer graphics which has been extensively studied over the past several years. In general, the segmentation problem is ill-posed and no objective measurement does exist for universally assessing segmentation quality. Judging the quality of a segmentation is application dependent [20, 21] and it is very difficult to propose an approach capable of segmenting all various kinds of models.

Primitive type recognition and primitive fitting are key issues for mechanical CAD model segmentation [22]. Schnabel et al. [7] developed an efficient RANSAC-based [23] framework for recognizing planes, spheres, cylinders, cones and tori. This approach is quite fast and robust to outliers. However since the RANSAC-based approach only looks for local cues it may result in over or under-segmentation or even wrongly identified primitive types because its estimation of the primitive parameters is sensitive to noise of both the sampled points' positions and normals [24] (see examples in Figs. 2.15 and 2.16 for its failure cases). This issue also applies to other methods that use low level information such as Gaussian or mean curvature [25, 26].

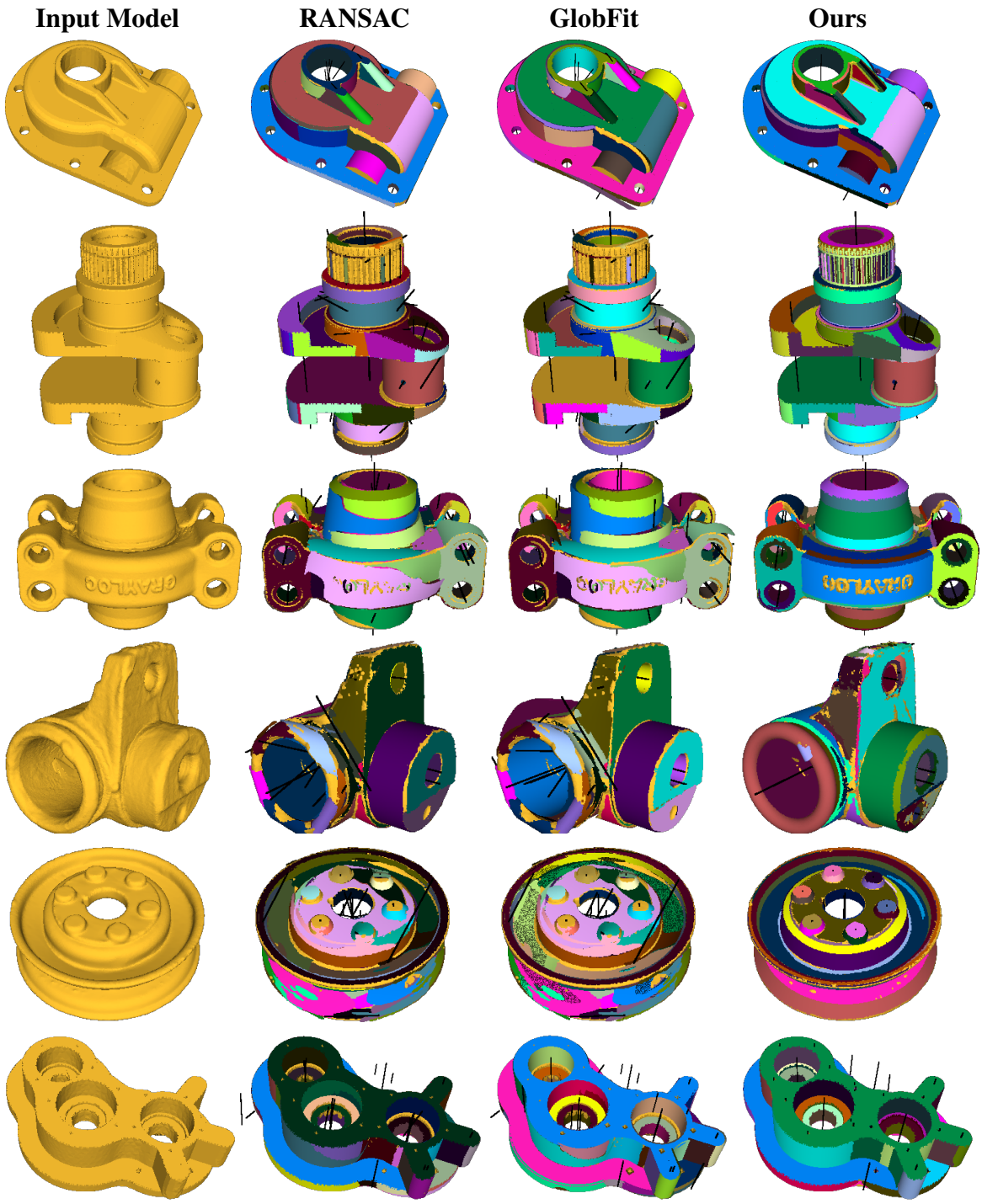


Figure 2.1: Failure case of RANSAC-based approach [7] and GlobFit [8]. **First column**: Input model (point cloud); **Second column**: Segmentation from RANSAC-based approach is locally optimal (black lines indicate the axes of the primitives); **Third column**: GlobFit's optimization can align the parallel primitives but increase the fitting error and fail to recover the correct segmentation; **Last column**: Our segmentation.

Steiner et al. [27] proposed the variational shape approximation method that employs linear approximation, or planes, to segment the model. Later, Wu et al. [28] extended this algorithm to cylinders, spheres and rolling ball surfaces. This variant also tends to over-segment the regions between patches and is sensitive to noise as it usually creates many new proxies for the noisy regions. Yan et al. [29, 30] used iterative quadratic surface fitting for segmenting the meshes of CAD models as well as free-form geometry, and the experimental results seem to be very promising. This method uses quadratic surfaces to approximate all kinds of primitives (plane, sphere, cylinder, cone, etc). The results obtained from this approach can be adjusted easily as the user can insert and merge quadratic surfaces interactively. However for CAD objects, quadratic surfaces are sometimes too flexible because they may over-fit the underlying surface and create an undesired segmentation. In addition, the segmentation quality is heavily dependent on both the number and the positions of the initial seeds.

Recently, researchers have been trying to unearth global information from the 3D models. Li et al. [8] designed the GlobFit method which improves RANSAC's output by exploring the global relationship among primitives such as parallelism, orthogonality, co-axis, equality, etc. The authors pose the discovery of a global relationship as a constrained optimization problem where the objective function is the least-squares fitting term and the constraints are the relations among primitives. This method, as its name suggests, focuses on fitting rather than segmentation. It is able to refine primitives' parameters such as location and orientation to globally align them. However since the global relationship is extracted during a post-processing stage, it is highly dependent on and limited by the initial primitives output by RANSAC. If the initial primitive types provided by RANSAC are wrong, it is very difficult if not impossible for the algorithm to correct them (see Fig. 2.1). Monszpart et al. [31] proposed the RAPter approach to extract a regular arrangement of planes from point cloud (of man-made scene). Their main technical contribution is a formulation that balances the dominant scene orientations and the less-dominant orientations

as the internal integrity. Demir et al. [32] segments and detects similarities within an existing 3D architectural model by casting the segmentation problem as a weighted minimum set cover over an input triangle soup, and maximizes the repetition of similar segments to find a best set of unique component types and instances. Golovinskiy et al. [33] attempted to apply machine learning concepts to recognize semantic objects in 3D point clouds of urban environments. However, it is difficult to propose an appropriate shape feature for all kinds of objects, especially for the geometric primitives in the context of CAD models. Moreover, the requirement for moderate to large training data in 3D cannot always be met.

In this work, we propose a novel segmentation framework for mechanical CAD models that overcomes some of the limitations of the existing work. Our method can deal with planes, cylinders, cones, spheres, tori and can be easily extended to surfaces of revolution. Based on the observation that common mechanical models only have a limited number of dominant axes around which the primitives are constructed (i.e. the primitives are either parallel or orthogonal to one of these axes), we estimate the dominant directions of the model by detecting *poles* and *rings* formed by the point cloud's normals. Note that the idea of using the normals to estimate the major directions has been studied before [34, 35], but people have only explored incorporating major orientations for planes in building scans [34] or cylinders in pipelines [35]. We estimate them for all planes, cylinders, cones, tori and use them to generate an over-complete set of various segmentations of the CAD models. The estimated major directions significantly reduce the search space and degrees of freedom for the subsequent primitive detection and enhance the robustness as well as the accuracy of the segmentation. We then convert the 3D primitive detection into a sequence of 2D circle and line detection by slicing the point cloud along each main direction, finding circles in each slice, constructing and segmenting the profile curve. The final segmentation is formulated as a set cover optimization where "items" are over-segmented planar patches and "subsets" are the over-complete set of detected primitives obtained from the profile curve analysis.

The proposed dimension reduction approach makes our method more robust to noise. For example, suppose we need to find a cylinder among 100 3D-points with 20% outliers. To estimate a cylinder with 5 degrees of freedom, we need at least 5 points, and hence, the solution space is $\binom{100}{5} = 75{,}287{,}520$. Because there are only 80 points actually belong to the cylinder, the number of correct tuples is $\binom{80}{5} = 24{,}040{,}016$. It means that no matter what detection algorithm we use, it has to distinguish 24,040,016 tuples from the total of 75,287,520 tuples, which leads to the successful detection probability $\frac{24{,}040{,}016}{75{,}287{,}520} \approx 31.93\%$. Now consider an equivalent 2D circle detection with 100 2D-points with the same 20% ouliers. At least 3 points are required to calculate a 2D circle. Following similar computation, we obtain the successful detection rate $\frac{82{,}160}{161{,}700} \approx 50.81\%$, which is significantly higher than that of the cylinder. Moreover, slicing the model, detecting circles in each slice and analyzing the profile curve altogether make our approach robust. In fact, a stack of many co-centered circles (of the same radius) provides more confidence of the presence of a cylinder in 3D than a fragile direct detection of a cylinder in 3D. Our main contributions include:

- A global model major orientation estimation of the mechanical model which is done early in the pipeline instead of in a post-processing step.
- A dimension reduction approach that transforms the 3D primitive detection problem into the classical 2D problems such as circle and line detection in image.
- A formulation of the segmentation as a set cover problem.

Our proposed algorithm is more robust than the RANSAC-based approach [7] due to the fact that we globally identify the model's major orientations, incorporate a dimension reduction approach, produce an over-complete set of geometric primitives and find the optimal segmentation as a set cover optimization. Comparing with the GlobFit [8], the segmentation result of our algorithm is not dependent on the quality of the initial segmentation result and is much faster. The remainder of this chapter is organized as follows. Section 2.2

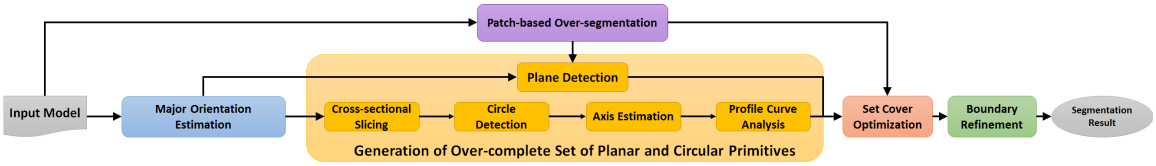


Figure 2.2: Flowchart of our segmentation approach.

briefly reviews other related works on 3D segmentation. Section 2.3 describes our novel segmentation method in detail. Experimental results are shown in Section 2.4 along with a side-by-side comparison with some of the state-of-the-art methods including RANSAC [7] and GlobFit [8]. Section 2.5 concludes our work. In this work, we use random colors to differentiate parts of a segmentation result. To visualize the primitive types, we use red, green, blue, cyan, olive and purple for planes, cylinders, cones, spheres, tori and surfaces of revolution, respectively. The unlabeled regions are in golden color.

Algorithm 1 Our algorithm for segmenting a point cloud P

1: Over-segmentation the model into patches (Section 2.3.1).
2: Estimate the model's major orientations (Section 2.3.2).
3: Detect planes orthogonal to each of the major orientations (Section 2.3.3).
4: Generate an over-complete set of circular primitives (Section 2.3.4).
   - Slice the model along each of the major directions, generate slicing image and detect 2D circles.
   - Group adjacent circles with common centers into connected components. Estimate the axis location and orientation of each connected component by fitting line to the circle centers in the connected component.
   - Construct profile curve for each component and segment the profile curve into lines and circles.
5: Set cover optimization (Section 2.3.5).
6: (Optional) Refine segmentation boundary (Section 2.3.6).

2.2 Other Related Works

The CAD models are often represented with either point clouds or meshes. The class of segmentation algorithms applied for point clouds and that applied for meshes actually over-

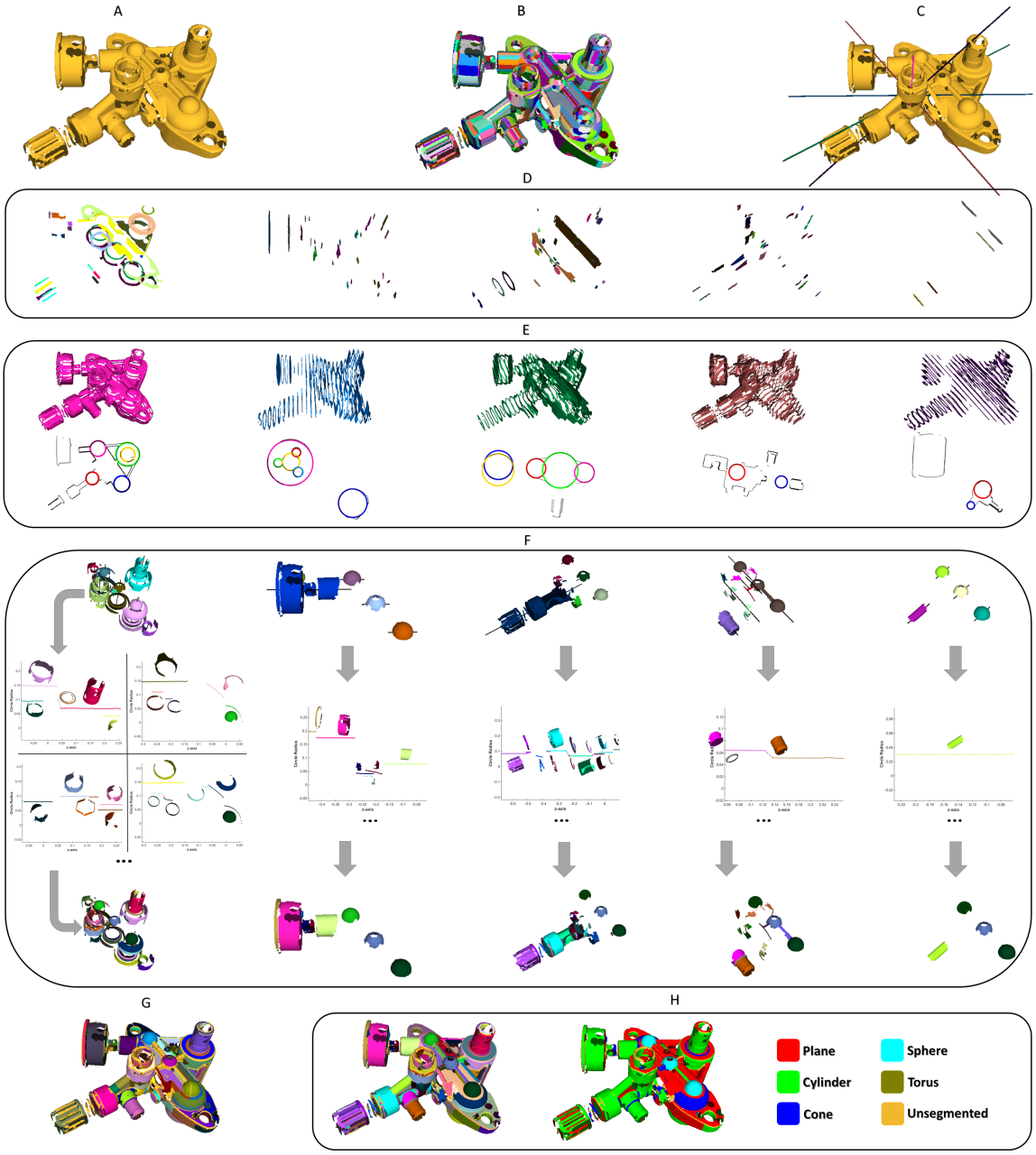


Figure 2.3: Illustration of our approach on the *mechanical-part* model. **A**: Input point cloud; **B**: Patch-based over-segmentation; **C**: Major orientation estimation; **D**: Planes are detected from over-segmented patches from **B** and the major orientations from **C**; **E**: Cross-sectional slicing of the model along each orientation and circle detection on each slice; **F**: *top*: Group adjacent circles into connected components and for each component, fit line to the circle centers to estimate the circular primitive's axis location (and possibly refine the axis orientation); *bottom*: circular primitives are generated from the profile curve analysis (best viewed with electronic zoom-in or see Fig. 2.13 for more details); **G**: Set cover optimization; **H**: Final segmentation result after boundary refinement (*left*: segmentation colored by primitives; *right*: segmentation colored by primitive types).

lap. The point cloud-based approaches usually can handle mesh as well by sampling points from a mesh, but generally not the other way around. Shamir et al. [20], Agathos et al. [36] and Theologou et al. [37] give a detailed survey on mesh segmentation techniques. Many segmentation approaches are usually formulated as an energy minimization problem and various error measures have been proposed in the literature to define the energy. These measurements quantify the properties such as planarity of various forms, higher degree geometric proxies (cylinders, cones, spheres, etc.), dihedral angles between triangles [38], curvatures (Gaussian curvature or mean curvature) [39], geodesic distances on a mesh, slippage, symmetry, convexity, medial axis, shape diameter [40] and motion characteristics [20]. According to [20], a majority of mesh segmentation algorithms can be divided into five classes: region growing, hierarchical clustering, iterative clustering, spectral analysis and implicit methods, some of which can also be applied for point cloud. These algorithms make use of the error metrics described above to group "similar" triangles into segments.

Region growing, the simplest of all possible segmentation methods, is locally greedy and very fast [41, 42]. Region growing includes single-source (where the growing starts from a single seed and stops before growing from another seed) and multiple-source region growing. Variants of the latter include the watershed method [43] and the distortion-minimizing flooding algorithm [27] which cleverly controls the growing based on a priority queue. Researchers apply hierarchical clustering in both a bottom-up and top-down fashion. In the bottom-up hierarchical clustering [44, 45], the algorithm starts with every triangle as a separate segment. At each iteration, the best fitting geometric primitive (plane, cylinder, sphere, cone, torus) approximating the triangles among every pair of adjacent segments determines the merging process, until hitting a required number of segments. In the top-down approach, the partition is achieved by finding the best boundary between parts at each step [46]. Iterative clustering methods such as the K-means method and mean shift [47] usually cannot be applied directly for recognition of mechanical parts. Spectral

graph theory-based segmentation algorithms [48] demonstrate some success in image segmentation but cannot be directly applied to CAD segmentation because the construction and analysis of the affinity matrix on millions of triangles is infeasible. The spectral approach also does not exhibit primitive information, which is important for mechanical parts. It has recently been proven that noisy mesh normals significantly affect shape recognition results [49]. Yi et al. [49] developed an iterative slippage analysis which is less affected by normals' noise. This method works well for small to medium-sized models but does not scale well for large-sized models. Implicit methods extract the contours and hence implicitly define the segmentation [50, 51]. The random walk method [52] usually cannot be directly applied to segment CAD models, because it depends on the initial seeds and, more importantly, the probability computation is based on local cues such as Gaussian and mean curvatures which do not explicitly enforce the model's primitive structure. The heat walk algorithm [53] was designed for segmenting free-form objects and hence it yields bad segmentation results (over-segmentation, undesired boundary creation, non-primitive conforming segments) on CAD models.

The most relevant works to ours are the pipe-run extraction and reconstruction [35], the generalized cylinder decomposition [54] and the polycube map construction by [55]. Qiu et al. [35] captures the dominant cylindric shapes and reveal similarities between cylinders. They also use the circle detection to guide the primitive fitting, but their method is only applied for cylinders with T-joints, boundary joints or curved joints. Zhou et al. [54] locally fits generalized cylinders (GCs) by computing the cylindrical term (a linear combination between the straightness of the local skeleton and the profile variation), then merges local GCs into non-local GCs and finalizes the decomposition by solving the Exact Cover Problem. GCs, however, are too abstract to reveal the accurate structure of the object, which is essential in the CAD context. He et al. [55] slices the model by horizontal planes which serves as a divide and conquer approach for polycube construction. However, these two later works are not designed for decomposing CAD models into geometric primitives. To

the best of our knowledge, we are the first one using the model's major directions to decompose the model and doing the dimension reduction to detect various types of geometric primitives.

2.3 Our Approach

Fig. 2.2 and Algorithm 1 summarize the main pipeline of our segmentation framework with illustration on a model named *mechanical part* in Fig. 2.3. It consists of the following main stages: patch-based over-segmentation, model's major direction estimation, generation of an over-complete set of planar and circular primitives, set cover optimization and boundary refinement. In general, our approach first over-segments the model by planar patches. Then using the detected major orientations, we produce an over-complete set of (planar and circular) primitives and optimize them as a set cover problem. In our framework, two predefined thresholds, the distance (ε) and normal (α) enforce the tolerance between a primitive and the sampled point. We set ε to 0.5 percent of the length of the diagonal of the point cloud's bounding box and α to 20° for all of our tested models. In the subsequent sections, we will elaborate on each step.

2.3.1 Patch-based Over-segmentation

This step takes as input the point cloud and produces a set of planar patches, which is essential for the optimization (in Section 2.3.5) as well as plane detection. A plane only has three degrees of freedom and is easier to be robustly detected than cylinder, cone, sphere and torus. We deploy the variational shape approximation [27] to linearly approximate the input model. A simple region growing is used to bootstrap the algorithm. We randomly pick a point s as a seed point and merge the neighboring points whose normals do not deviate more than α from the normal of s. The process is repeated until no more patches

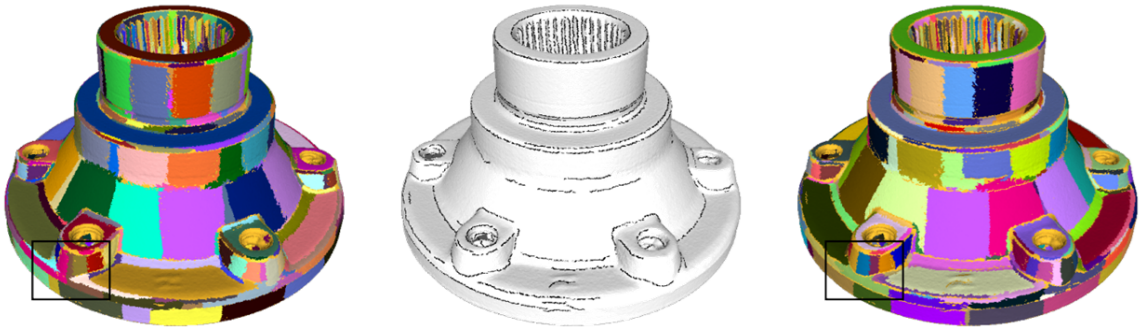


Figure 2.4: Crest lines prevent the patches from spanning multiple primitives. **Left**: patches (without crest lines); **Middle**: detected crest lines shown in black [9]; **Right**: patches from *constrained* region growing.

Figure 2.5: Partition of a sphere into 1,600 regions by sub-dividing the spherical coordinates (**left**) and the equal-area partitioning [10] (**right**).

can be generated. Fig. 2.3B shows an example of patch generation from the *mechanical-part* model.

Nevertheless, it is possible that a patch crosses the boundary between two different primitives (Fig. 2.4 left). To solve this problem, we compute the crest lines [9] (a subset of curvature extrema) of the input model and use them as blocking markers to prevent the patches from growing across the primitives. As we can see in Fig. 2.4, the crest lines (middle) detected in the *carter* model improves the over-segmentation result (right).

2.3.2 Major Direction Estimation

A common property of a mechanical model is that it has a limited number of (usually from 1 to 3) main orientations around which the geometric primitives (planes, cylinders, cones, tori) reside. If these major orientations are known, it is possible to narrow down the search space for detecting the primitives because their degrees of freedom are reduced by two. In

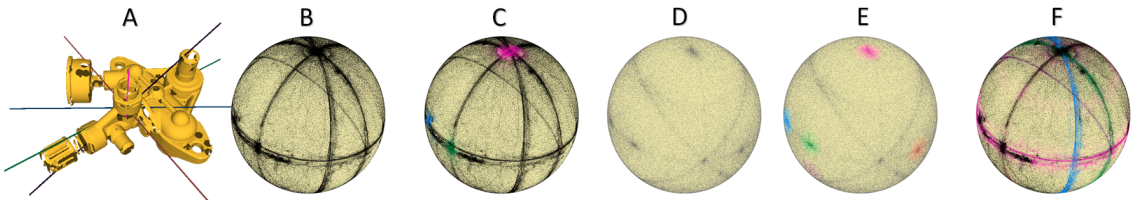


Figure 2.6: Major direction estimation. **A**: Input model and the estimated major directions; **B**: Projection of point cloud's normals (black dots) onto the Gaussian sphere; **C**: the poles (in pink, dodger blue and dark green) of the normals' distribution, **D**: mapping of randomly chosen triples (a triple of three black dots in B corresponds to a gray dot); **E**: poles (in pink, dodger blue, dark green, brown and purple) in the mapped space; **F**: corresponding rings (in pink, dodger blue, dark green, brown and purple) of the normals' distribution. The final model's major directions are shown in **A**.

this work, we are dealing with planes, cylinders, cones, spheres and tori (please note that spheres do not have any natural axis and we will talk about it later in Section 2.3.4). We refer the primitive's axis to the plane's normal and the cylinder, cone or torus's axis and it can be found from the distribution of the point cloud's normals. Ideally, the distribution of the plane's normals on a spherical Gaussian surface is a single point and that of a cylinder or a cone is a circle. In other words, if we can locate such special points and circles from the normals' distribution mapped on the Gaussian sphere, the model's major orientations can be derived in a straightforward manner.

However, the real scanned models inevitably have noise. As a result, the distribution of the plane's normals becomes a high density circular region, or *pole*, and the circle obtained from the cylinder or cone's normals becomes a *ring* (Fig. 2.6B), which makes the model's major orientations harder to be revealed. To tackle the problem, we find the poles and rings separately. The poles are easier to be detected as the mean shift clustering [56] can be applied, but we have made two modifications. First, to initialize the starting position, we use the histogram method. The Gaussian sphere is divided into 1,600 regions, each of which represents a bin to accumulate the normals. There are many ways to achieve such division. The most naive method is using the spherical coordinates and sub-dividing the azimuth angle and the elevation (e.g. 40×40). Unfortunately, this leads to non-equal zones and distortions near the pole (see Fig. 2.5 left). As a result, some bins are bigger and

have higher probability of receiving more votes while the bins near the poles are too fine to expose any local maxima. To overcome such problem, we use the equal-zone sphere partition [10] which ensures that each bin has the same surface area, hence have equal probability of receiving votes (Fig. 2.5 right). Second, we check the stability of the final modes. Upon convergence, each mode is randomly perturbed (it is shifted by some random noise) a few times (e.g. 20) and the mean shift is run again at the new starting location. If the mode converges to the same position, it is called stable and is accepted as a pole.. Fig. 2.6C shows the poles detected from the normals' distribution from Fig. 2.6B.

The rings, on the other hand, are more challenging because a small deviation of the ring plane results in a relatively big change in its orientation. Yet we only need to know the rings' orientations, not to exactly locate every ring. Consequently, we choose three normals (black points in Fig. 2.6B) at random and compute the orientation of the plane passing through them. The process is repeated over a relatively large number of triples (e.g. a million triples). That is to say we have transformed the problem of finding the circular rings' orientations in the original Gaussian sphere into that of finding the poles in the new Gaussian sphere (see Fig. 2.6D). Hence the same procedure of recognizing poles in the preceding paragraph is deployed in the converted space (Fig. 2.6E and 2.6F). Moreover, another layer of verification is added. We project the point cloud onto a plane perpendicular to each orientation candidate and assert if there exists a circle in the projection image.

The list of poles and rings' orientations are unified together to eventually become the model's major directions (Fig. 2.3C). These orientations serve as a decomposition of the input model which reduces the search space for the planar and circular primitive detection described in the subsequent sections.

It is worth mentioning that we use conservative thresholds for the mean-shift clustering during the pole detection (both in the original and mapped space) to make sure that we do not miss any orientation. Our algorithm, however, can still work smoothly given redundant orientations as it would only take more time for detecting the circular primitives for each

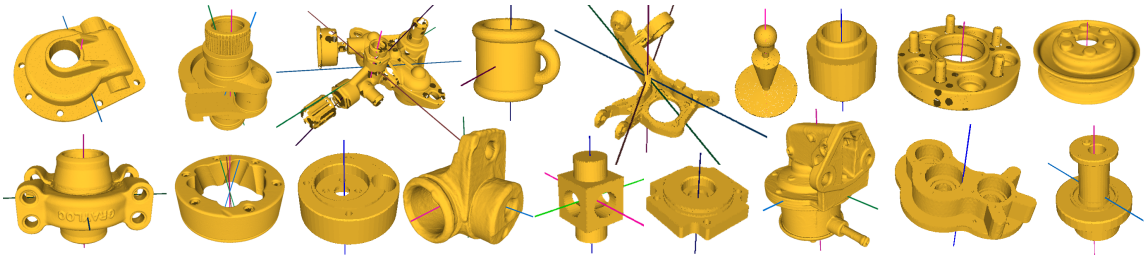


Figure 2.7: The estimated major orientations for all tested models. Note that the colored axes represent orientations only, not location. See Fig. 2.1 (last column) for our detected axes.

orientation. The major orientation detection for all tested models are shown in Fig. 2.7.

2.3.3 Plane Detection

The planar patches obtained from the over-segmentation (Section 2.3.1) are planes of themselves. However, to increase the robustness of the plane detection, we further merge it under the constraint of the model's major orientation. More specifically, for each model's major orientation, we group the patches orthogonal to it. Then the z-value of each patch (with respect to the orientation and the origin) is calculated, which is clustered to get the planes orthogonal to each of the model's orientations. Fig. 2.3D shows examples of plane detection for five different orientations of the *mechanical part* model.

2.3.4 Generation of Over-complete Set of Circular Primitives

This step is to propose various hypothetical segmentations by generating an over-complete set of circular primitives using the detected major orientations. For the circular primitives such as cylinder, cone and torus, the intersection between each of them and a plane orthogonal to its axis is a circle. A plot of the circle's radius versus the displacement of the plane containing the circle is called the *profile curve*. The profile curve of a cylinder is a horizontal line segment and that of a cone is a oblique line segment. The profile curve of a torus is a circle whose center is off the z-axis. A sphere does not have any natural axis,

but if given any axis passing through its center, its profile curve can be defined similarly to the case of cylinder, cone and torus. The profile curve of a sphere is also a circular arc but its center is on the z-axis. Exploiting this property, we convert the 3D circular primitive detection in the original model into the 2D circle detection in its cross-sectional projection along the primitive's directions followed by segmenting the profile curve into line segments and circular arcs.

Projection Image Generation

Starting with the model's major orientations, we uniformly slice the model along each of them (see Fig. 2.3E top row). The points whose normals are parallel to the axis orientation have been grouped into planes (Section 2.3.3) and thus are excluded here. The thickness of each slice, denoted by τ, depends on the point cloud's density. In all of our experiments, we choose $\tau = \varepsilon = 0.5\% \, diagonal$, where $diagonal$ is the length of the diagonal of the model's bounding box. There are at most $\frac{diagonal}{\tau} = \frac{diagonal}{0.5\% \, diagonal} = 200$ slices from each of the major orientations. Each slice defines a plane orthogonal to a major orientation and containing the projection of all points within τ-distance. Examples of projection images are shown in Fig. 2.3E second row in black pixels.

After the projection images are generated, we will proceed to detect the circles on these images. The 2D circle detection is a classical problem and it has been extensively studied for many decades [57, 58, 59, 60, 61, 11, 62, 12]. In general, one could use one of these methods as a sub-routine for this task. We apply our circle detection [63] because it is much more robust than both the Circular Hough Transform and RANSAC approaches.

2D Circle Detection

Automatic circle detection is a fundamental problem in computer vision and has a wide variety of applications such as traffic sign detection, robot vision, pupil and iris localization, vectorization of hand-sketched drawings, automatic inspection of manufactured products

and components, people counting in surveillance video, etc. In consequence, the circle extraction problem has been extensively studied in the literature and most of the proposed methods belong to either of the two categories, Circular Hough Transform (CHT) and Random Sampling Consensus (RANSAC).

The CHT, in its original form [64, 65, 66], is the most universally applied approach for detecting circular shapes. Commonly, an edge map of the image is generated and an expected circle radius is given. The CHT collects the contributions, or *votes*, from the edge pixels (or *edgels* for short) for an accumulator describing the parameter space (or sometimes called the Hough space) which represents the circle's center. Then, circles are extracted by finding the local maxima of the parameter space. Despite of its popularity and simplicity, the CHT has several disadvantages. First of all, the CHT is not robust to noise because the accumulator may add up votes from noisy regions and finally return some false peaks. Morevoer, the memory demand for the Hough space is high and the computation time could also be an issue when the CHT updates the parameter space or searches for local maxima. If the target radius is unknown, the situation becomes worse as the CHT has to deal with a three dimensional parameter space. Another limitation of the CHT is the Hough grid's size. Too coarse grid can lead to a large number of votes being obtained falsely because many different structures can locate in a single bucket and too fine grid can lead to structures not being found. To overcome such problems, many modifications have been proposed to improve the CHT's performance [67, 68, 69, 70, 71, 59, 60] but they are still not robust to noise as noise are accumulated into the parameter space.

In addition to the CHT-based methods, the RANSAC-based [23] approaches are a viable alternative option. They essentially generate hypothesis of a circle and immediately test it rather than accumulate it. The Randomized Circle Detection (RCD) [58] is a typical approach in this class. It iteratively picks four edge points at random where three of them are for generating a circle candidate and the remaining one is for validation purpose. The process is repeated and the circle candidates with large support are returned. The RANSAC

paradigm, however, has two main disadvantages. First, it makes an early decision which can lead to a lot of false detection especially with presence of noise. Second, a substantial large number of possible candidates have to be generated and verified, which is not practical in many scenarios. For example, consider an image with 300 edgels (which is a relatively small number in real case), among which 100 edgels belong to a circle. According to the RCD, the probability of a 4-tuple of randomly chosen pixels that all come from the 100-edge-pixel circle is $P = \frac{100\times99\times98\times97}{300\times299\times298\times297} \approx 0.01185$. That is to say, in the worst case, $1/0.01185 \approx 84$ times should be run to just find one circle, not to mention that real image usually contains dozens of thousands of edgels. As a result, several adjustments have been proposed such as random line pooling [72], gradient evidence [73], symmetric property [74] for screening of circle candidates. Other refinement methods including generic algorithms [75] and electro-magnetism optimization [62] are designed to improve the accuracy. According to [76] and our experiments, RANSAC-based methodology yields a large set of false positives.

We propose a new method to detect circles from images. Our approach consists of two steps, circle candidate generation and verification. In particular, we start by line segment detection from the edge image, then use pairs of detected line segments to generate an initial set of circle candidates. Non-maximum suppression is applied to this initial set via mean-shift based clustering while the cluster centers are considered as the final set of circle candidates. These circle candidates are then individually verified by both the ratio between number of edgels to their radii and their *completeness*. Our contribution stands out from the existing methods due to the following:

- Our hypothesis generation is based on line segments which are much cleaner than edgels and hence increase the accuracy and robustness to noise.
- Our circle verification also differs from existing methods in that besides fitting error, it measures the *spanning angle* of the circular arc, a more natural way in circle detection which mimics human perception.

- We create CDBD, the first benchmark dataset for circle detection which consists of more than a thousand images with ground truth circles labeled by human.

Although we share some similar ideas with existing works such as [77, 78, 79] which links edgels to segments or arcs, finds the circle candidates and verifies them based on inliers and outliers, etc., the major difference is that in our work, we generate the candidates after considering a (large) number of candidates (obtained from all pairs of line segments) while in other works, candidates are generated locally from a (small) set of arcs or segments and those local circular arcs are merged together using various (still local) approaches with some parameters. These merging operations are unstable, harder to be controlled and do not guarantee error-free. This difference, though subtle, makes the whole algorithm much more robust. It is because under noise, a subset of edgels from a true circle may result in a very different circle. Moreover, our method has no problem with discontinuous circle while others may fail with it because fitting circle locally to a small arc could result in totally different circle from fitting "globally". Another issue is that the verification by inliers and outliers can reject (most of) the candidates with insufficient amount of supporting evidence, but it does not guarantee to prune the cases where candidates have enough supporting edgels, yet are false positives. The testing images in existing papers are too simple, so it is not difficult for them to deal with. However, the natural images (especially those in our CDBD) are complicated and the methods based on local estimation and linking produce lots of false detection (based on our empirical experiment).

The mathematical theory behind our method is that if A and B are two points on a circle centered at C, the line d_{AB} perpendicular to the chord AB at its middle point, or *sagitta*, must go through C. As a result, if we have two chords AB and MN, the intersection of d_{AB} and d_{MN} gives us the circle's center C with the radius r to be the average of the lengths of the line segments CA, CB, CM and CN. Our approach consists of the following steps (Fig. 2.8):

1. Circle candidate generation

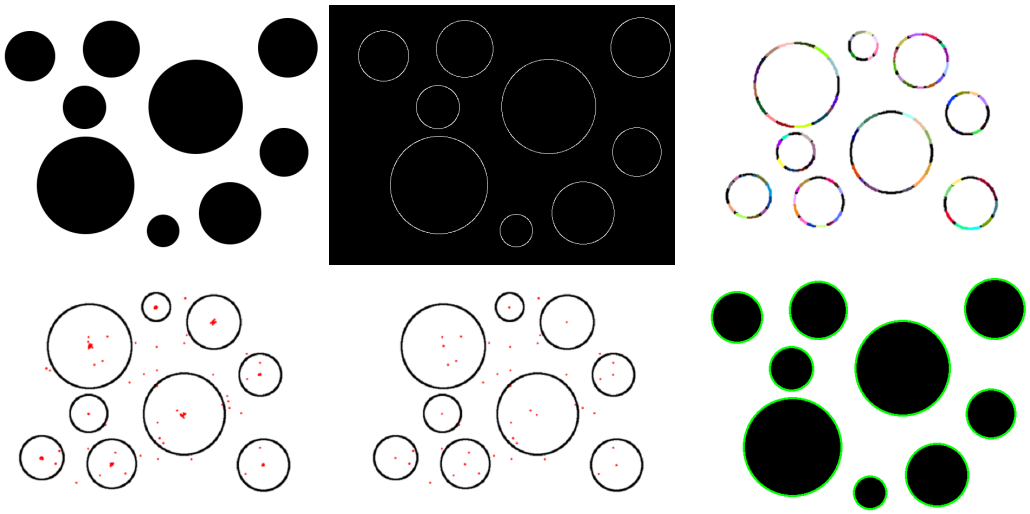


Figure 2.8: Illustration of our circle detection (*best view in electronic version*). *From left to right, top to bottom*: input image; edge map; line extraction; circles' centers generated from pairs of lines; circles' centers candidates; final detected circles.

(a) Given an image I, compute its edge map E.

(b) Compute the normal for each edgel on E.

(c) Detect line segments on E.

(d) For every pair of line segments (l_i, l_j), compute a circle candidate and verify it by the edgels on both l_i and l_j.

(e) Perform non-maximum suppression via mean-shift clustering on the generated circle candidates, with each cluster represented by its center.

2. Circle candidate verification based on supporting edgels and circle's completeness.

For the edge map extraction, we use the Canny's edge detector [13]. The normal of each edgel can be approximated by the image gradient, or by performing local PCA on the edgels themselves. We test both and they give similar performance. The next step of the algorithm is to detect line segments from these edgels, which is the linear approximation of the raw edgels to get more information about the local geometric structure. Another benefit of using line segments is their robustness to noise. There are several algorithms to

detect line segments from edgels and we choose the LSD [80] because of its false detection control. An example of line extraction is shown in Fig. 2.8 (first row, last column).

The algorithm continues by choosing all pairs of line segments (l_i, l_j), computing d_i and d_j as the sagittas, respectively. The intersection C of d_i and d_j gives the circle's center. The radius r is averaged by the distances from C to the four endpoints of l_i and l_j. A circle is verified against the edgels making up l_i and l_j based on a distance tolerance ε, normal tolerance α and inlier percentage $\gamma = 0.8$ (i.e. at least 80 percent of the edgels on both l_i and l_j are within ε distance and α degrees from the circle).

Fig. 2.8 (second row, first column) is an example of circle candidates generated from valid pairs of line segments. As we can see, choosing all pairs of line segments creates many circles which may have duplicates. To remove such these duplicates, we apply non-maximum suppression via mean-shift clustering [56]. Note that in our method, we perform a two-step clustering corresponding to the circle's centers and radii because the clustering approach usually performs better at a low dimensional space. The mode corresponding to each cluster represents the circle candidate. The final set of circle candidates is typically small enough to be considered individually (as shown in Fig. 2.8 (second row, second column)).

After all the circle candidates have been generated, they will be individually fitted using least-squares fitting (R. Bullock, 2006 at `http://www.dtcenter.org/met/users/docs/write_ups/circle_fit.pdf`) and verified against the supporting edgels based on two criteria: the number of support and the completeness. Recall that the inliers of a circle candidate are obtained from the edgels satisfying the distance and normal tolerances. For each circle candidate, we do the connected component analysis on its inliers and discard the small components whose sizes are less than 10 percent of the largest component. The reason for doing this is to purify the circle's inlier set. For the minimum number of edgels on a circle, if we use a global threshold T_g, the scale problem arises because circles with different radii have different circumferences. The global threshold T_g is

not suited for circles with small radii. To overcome this scale issue, we use a ratio threshold T_r similarly to the approach in [58] where a circle with radius r is expected to have $2\pi r T_r$ edgels. As a result, circles with large radii require more supporting points and vice versa.

In addition to the number of supporting edgels, the second criterion is the circle's completeness. Intuitively, complete circle is more reliable than incomplete circle. We define the circle completeness as the angular coverage of the supporting edgels, which is calculated over the connected components of the supporting edgels. We only accept circles whose completeness is at least T_c degrees (e.g. $T_c = 180^\circ$). We start by detecting all 360-complete circles first, then 270-complete circles, 180-complete-circles, etc. until T_c-complete circles. This heuristics experimentally works very well. The verified circles for the illustration example are shown in Fig. 2.8.

Our algorithm has four parameters: distance tolerance ε, normal tolerance α, circle ratio T_r and circle completeness T_c. In all of our experiments, we set ε to 0.5 percent of the image's minimum size, α to 20°, T_r to 0.6 and T_c to 180°. For comparison purpose, we compare our approach with other two popular paradigms, the Circular Hough Transform (CHT) [64, 66] and the Randomized Circle Detection (RCD) [58]. For the CHT, we use the built-in `imfindcircles` from `MATLAB` with sensitivity as 0.9. For the RCD, there is no publicly available implementation, so we implement the approach described in [58] with all the parameters set to the given default values. All the experiments were run on a PC with Intel Core i5 3.2 GHz CPU and 8 GB RAM.

According to our knowledge, there is no publicly available benchmark for circle detection. In order to make the evaluation of this fundamental problem more rigorous and objective, we create the Circle Detection Benchmark Dataset (CDBD). Our dataset consists of more than a thousand of images with labels "circles" and "round objects" from the ImageNet [81] and images from the papers listed in the references section. The CDBD includes synthesized, hand-sketched and natural images with various levels of complication,

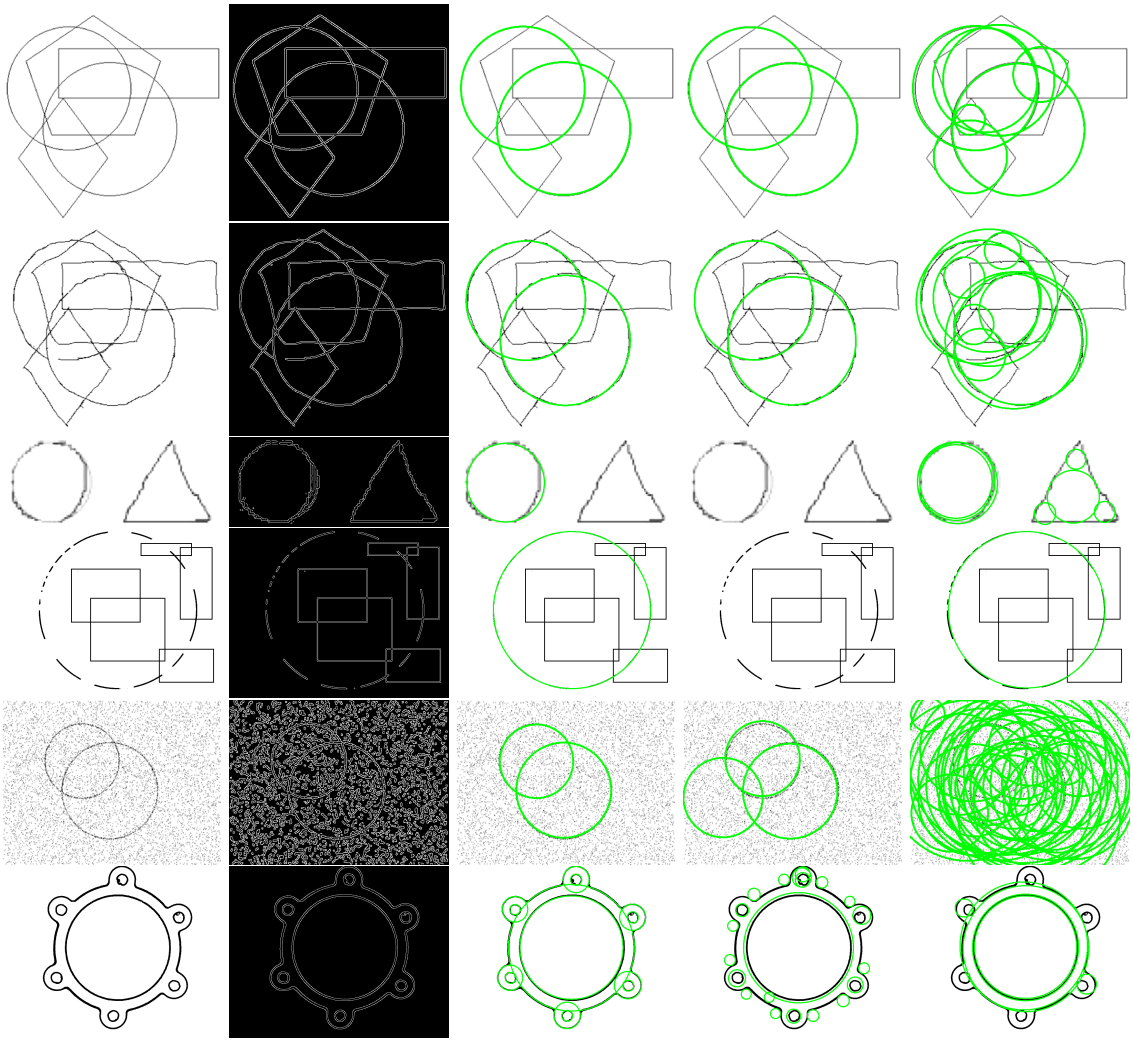


Figure 2.9: Comparison of the circle detection methods on synthesized images (images on rows 1, 2, 3, and 5 are copied from [11]). *First column*: input image; *Second column*: edge map; *Third column*: our approach; *Fourth column*: the Circular Hough Transform (CHT); *Last column*: the Randomized Circle Detection (RCD).

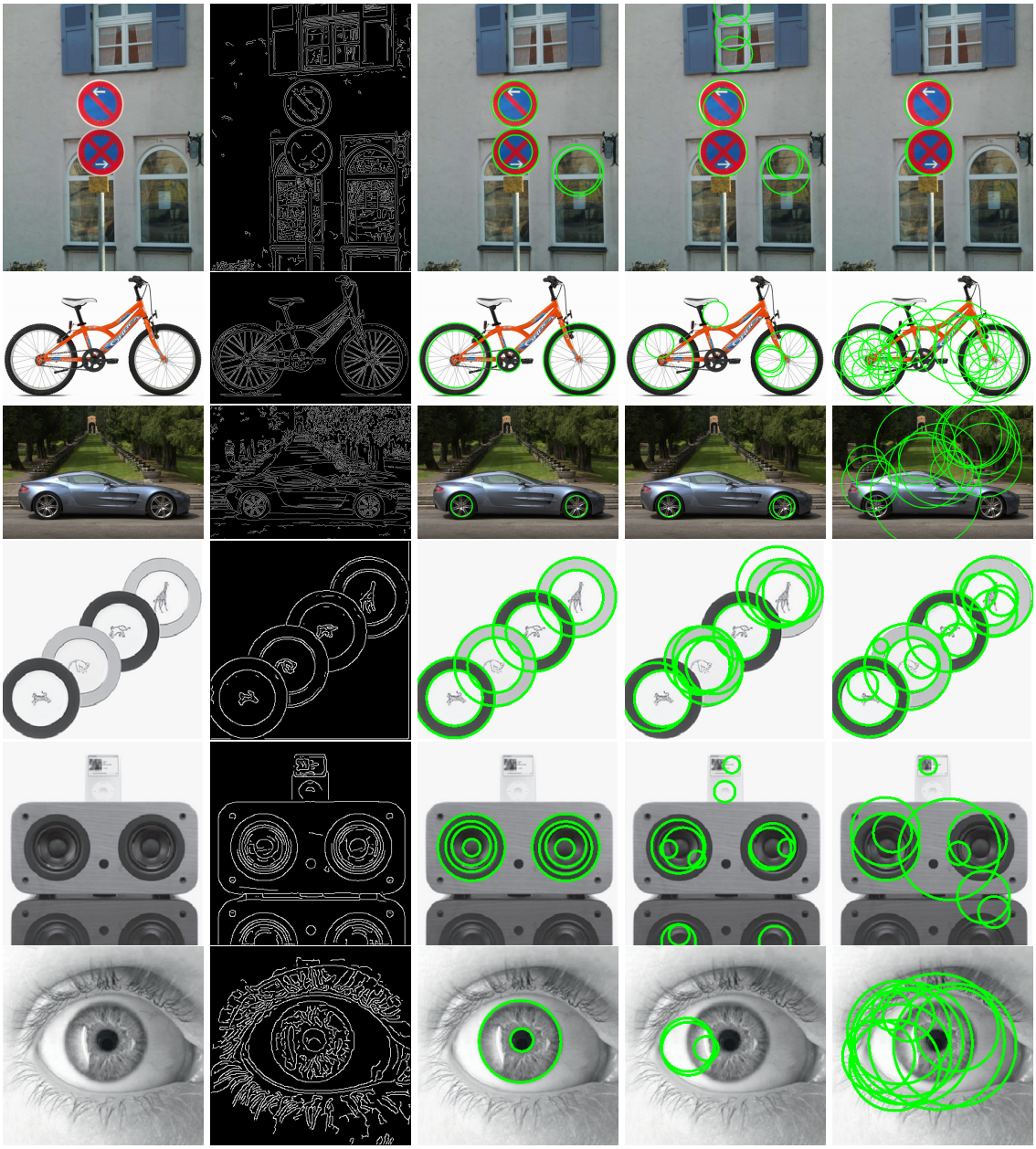


Figure 2.10: Comparison of the circle detection methods on natural images (images on rows 4, 5 and 6 are copied from [12]). *First column*: input image; *Second column*: edge map; *Third column*: our approach; *Fourth column*: the Circular Hough Transform (CHT); *Last column*: the Randomized Circle Detection (RCD).

noise and deformation. We asked several people to label all the circles in each image[1]. We will continue to enrich this dataset in the future.

Fig. 2.9 shows the behaviors of three circle detection algorithms on some of the synthesized images in the CDBD. As we can see, the RCD generally works well in clean images (such as those in rows 1, 4 and 6) because there is a high probability that within a small number of iterations, four edgels lying on the same circle are selected. However, when there are certain noise and/or deformations (rows 2, 3 and 5), the RCD usually returns a lot of false positives and sometimes a circle may never be detected in a reasonable number of RANSAC iterations (row 5). The CHT, on the other hand, struggles with the cocentric circles (row 6). It is able to tolerate some degree of deformation (row 2), but it does not work in noisy image (row 5) as noisy edgels can contribute a significantly high number of (irrelevant) votes which making the parameter space too evenly distributed and local maxima cannot be robustly located. Our circle detection works well under certain deformation and noise because it uses line segments which are robust to both deformation and noise.

Natural images are places where our algorithm really outperforms the other two (Fig. 2.10). The number of edgels from natural images is typically big (from a few thousands to a few dozens of thousands) and the complication increases tremendously, which challenges the other two methods as there are more uncertainties in the circle verification. Since our algorithm works on line segments, it can capture a certain degree of higher geometry information and reduce the solution space substantially. Moreover, the line segments help us to correctly compute the circles' parameters while in the RCD method, due to its early decision, circles' parameters (centers and radii) are not correctly estimated.

Fig. 2.11 illustrates the effect of the two parameters (T_r and T_c) in our framework. The circle ratio T_r controls the minimum edgel density while the circle's completeness T_c suppresses circles with small spanning angle. We realize that these two parameters, upon decreased, help more circles be detected or, in other words, increase the sensitivity

[1]We make our benchmark available at `https://github.com/trucleduc/Circle-Detection`

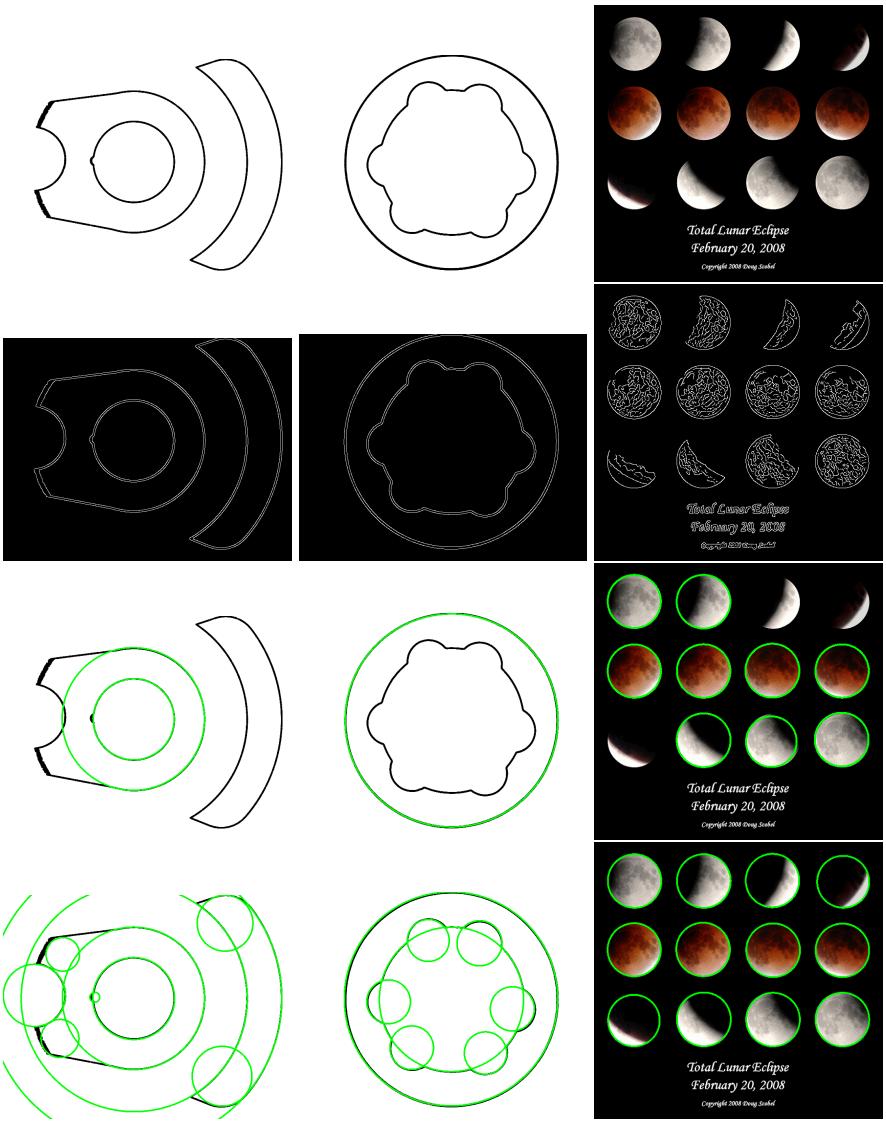


Figure 2.11: The effect of two parameters T_r and T_c. *First row*: input image (from CDBD); *Second row*: edge map; *Third row*: our circle detection with T_r and T_c are set to default values (i.e. $T_r = 0.6$ and $T_c = 180°$); *Last row*: our circle detection with $T_r = 0.4$ and $T_c = 60°$.

Figure 2.12: The effect of the edge detection on our circle detection. *Top left*: input image (from CDBD); *Second column*: Canny's edge detector [13] on top and Sobel's edge detector [14, 15] at bottom; *Third column*: our circle detection running on two corresponding edge maps.

	Ours	CHT	RCD
Mean precision	**63.25**	13.74	8.34
Mean recall	**73.37**	62.24	59.29
Run time (seconds)	7.46	9.21	**4.32**

Table 2.1: Comparison of three algorithms on our benchmark.

of our circle detection, but at the same time, could potentially lead to false detection. Our algorithm works directly on the edge image, so different edge detection algorithms produce different edge maps and hence affect our circle detection. Fig. 2.12 is an example where two popular edge detection algorithms (Canny's [13] and Sobel's [14, 15]) are applied separately. The Canny's edge map has ample details including weak edges thus allows more circles to be found. On the other hand, the Sobel's edge map only captures strong edges which exhibits the line detector and, certainly, our algorithm from revealing other circles.

For quantitative evaluation, the three algorithms (ours, the CHT and the RCD) are respectively executed against all images in the CDBD using the default parameters and their precision, recall as well as running time are recorded. Let C_g and C_r be the set of circles from ground truth labels (by humans) and from a circle detection algorithm, respectively. Recall that precision $= \frac{|C_g \cap C_r|}{|C_r|}$ and recall $= \frac{|C_g \cap C_r|}{|C_g|}$ (we use a pseudo count 1 to avoid zero division). The metric for comparing two circles is the Euclidean distance between their parameters (centers and radii). Two circles whose distance is less than 1 percent of the maximum image size are considered identical. Table 2.1 shows the average performance of our method against the two algorithms mentioned in the previous section. Our algorithm absolutely shines over the CHT and the RCD with respect to both mean precision and recall. Furthermore, the recalls of the CHT and the RCD are much higher than the precision indicating that they produce a lot of false positives and randomness while the precision our method keeps pace with its recall, which means that our algorithm is more robust and reliable. In terms of time performance, the unoptimized `MATLAB`'s implementation of our

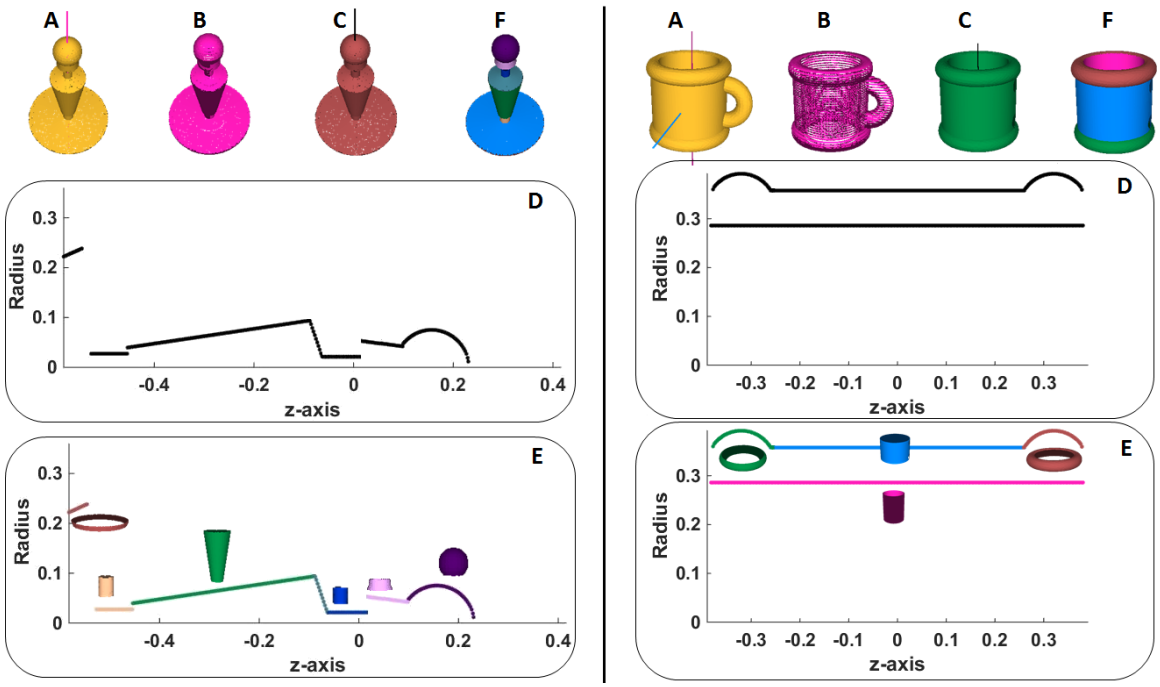


Figure 2.13: Two examples of profile curve analysis. **A**: Input models and detected orientations; **B**: Model's slicing along the orientation; **C**: Group of adjacent detected circles; **D**: Profile curve from **C**; **E**: Segmentation of profile curve into line segments and circular arc; **F**: Detected primitives in 3D.

algorithm is still faster than the highly optimized and built-in `imfindcircles` and, with trade-off between the running time and the accuracy, ours is comparable with the RCD.

Primitive Axis Estimation

Section 2.3.2 only estimates the axis directions with no information about the axis location. After all the circles are detected in the preceding section, we will group the adjacent circles with nearby circle centers into connected components. For each connected component of 2D circles we conduct line fitting to all the circle centers in the connected component, which not only gives the axis location but also refines the axis orientation.

Profile Curve Analysis

After the axis is estimated, a profile curve is constructed for each connected component. The vertical axis represents the radii of the circles in the connected component and the horizontal axis shows the z-values of the circles' centers. Figs. 2.3F and 2.13 show examples of the *profile curves* obtained from groups of co-centered circles. Cylinders and cones will be represented by line segments while spheres and torus by circular arcs in the profile curves, respectively. Thus, detecting line segments and circular arcs in the profile curves is equivalent to detecting circular primitives in the 3D space.

A circular arc could also be approximated by multiple short line segments. To reduce the ambiguity between a circular arc and line segments, we propose to first extract the circular arcs from the profile curve. We use the same method [63] to detect circular arcs. The circular arcs centered on z-axis correspond to spheres and those centered off z-axis correspond to torus (see Fig. 2.13).

Once the circular arcs are extracted, the remaining task is to detect all line segments in the remaining profile curve. In this work, we employ a region growing algorithm for line segment detection. The region growing is performed based on the circle radius (vertical axis of the profile curve), the circle center's displacement (horizontal axis of the profile curve) and the circle's average angle (measured by the average over the angles between the 3D normals of the associated points and the major orientation) which serves as the *gradient* of the curve (in fact, this angle equals to 90° minus the gradient of the curve). Other techniques such as the LSD [80] could also be used in place of the region growing. However, our experiments show that LSD sometimes could miss smaller line segments which our region growing algorithm correctly detects. Sample outputs of the line segments obtained from the profile curves are shown in Figs. 2.3F, 2.13E with their associated primitives.

Based on the profile curve information, the circular primitive's parameters can be easily computed. All circular primitives are extracted from each of the model's major orientations and individually fitted by the method in [82] which is known to be better than traditional

least-squares fitting due to its robustness to noise and degeneration. An additional verification step in 3D based on distance and normal errors of the 3D points and the primitives is added for eliminating the bad quality (or spurious) primitives.

2.3.5 Set Cover Optimization

So far we have generated a linear approximation (planar patches) P of the model and all of the hypothetical circular primitives. The set of hypothetical primitives, H, includes both the planar patches and the circular primitives. Note that the hypothetical primitives may overlap on each other. A patch $p \in P$ is said to belong to a hypothetical primitive H_i if the majority (e.g. 80%) of its points belong to H_i. We use both distance and normal constraints (with ε and α tolerances) to verify the point-primitive's membership.

It can be seen that a segmentation is an assignment of each patch in P to each hypothetical primitive in H, or, in other words, a choice among all subsets of H covering P. Based on the principle of the *Minimum Description Length*, we define a good segmentation as a minimal subset of H that fully covers P. Let us further denote a set of binary indicator variables where $x_i = 1$ means the i^{th} hypothetical primitive is selected in the minimal subset. The selection problem can be formulated by

$$\min \quad \sum_{i=1}^{|H|} x_i \tag{2.1a}$$

$$\text{subject to} \quad x_i \in \{0,1\} \qquad \forall i = 1, \ldots |H| \tag{2.1b}$$

$$\sum_{i:\ p \in H_i} x_i \geq 1 \qquad \forall p \in P \tag{2.1c}$$

The objective in (2.1a) clearly minimizes the number of selected hypothetical primitives. The constraint in (2.1c) asserts that the selection covers every element of the P. Another way of writing (2.1c) is $\bigcup_{i:\ x_i=1} H_i = P$.

The problem of identifying the smallest sub-collection from a set of collections whose

union equals a known universe (set cover) is a classical NP-complete combinatorial problem [83]. We apply the randomized rounding algorithm [84] to solve this optimization. The algorithm first computes an optimal fractional solution x to the linear programming relaxation of the original binary programming. After that the fractional solution must be converted to an integer solution (and thus a solution of the original problem). The main conversion technique is to use randomization, and then to use probabilistic arguments to bound the increase in cost due to the rounding (based on the probabilistic method from combinatorics). Probabilistic arguments are used to show the existence of discrete structures with desired constraints in (2.1b) and (2.1c).

2.3.6 Boundary Refinement

As a post-processing step, final primitives are grown to unlabeled regions using the *distortion minimizing flooding* algorithm [27, 30] with normal deviation as the error metric under the constraint of distance (ε) and normal (α) tolerances. The reasons for doing this are two-folds: to fill holes between the primitives' boundaries and to fix missing circles in the earlier step. After that, we smooth the boundary between pairs of adjacent primitives using Graph-Cut-based technique [85] similarly to [30].

2.4 Experimental Results

We run our algorithm on several mechanical models downloaded from the AIM@SHAPE Repository and Archive3D with various complication (see Table 2.2 for the list of our processed models). All experiments are executed on a PC with Intel Core i5 3.2 GHz CPU and 8 GB RAM. Our segmentation framework has three parameters: the distance (ε) and normal (α) tolerances for assessing a point on a primitive and the slice thickness (τ). They are fixed to the default values (ε is 0.5 percent of the diagonal of the point cloud's

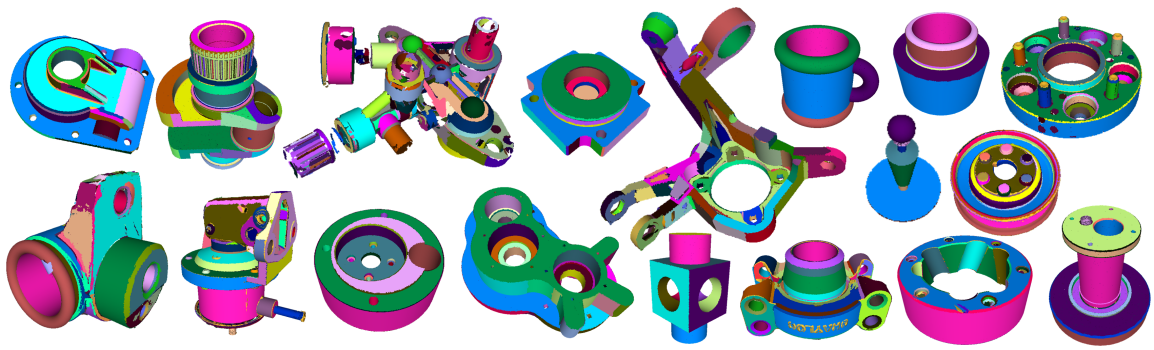


Figure 2.14: Projection of the point cloud onto the detected primitives obtained from our segmentation algorithm.

bounding box; $\alpha = 20°$; $\tau = \varepsilon$. For comparison purpose, we apply the RANSAC-based segmentation [7] and the GlobFit optimization [8] on the same data with parameters set to their default values.

Fig. 2.14 is the projection of the point cloud onto our segmented primitives, which is a rough reconstruction of the 3D models. As we can see, the projected points are highly consistent with the underlying surface, which illustrates the accuracy as well as the quality of our detected primitives.

Figs. 2.1 and 2.15 show a side-by-side comparison of our segmentation results with that of the RANSAC-based approach. We use random colors to differentiate between segmented primitives. For primitive type, we use red for plane, green for cylinder, blue for cone, cyan for sphere and olive for torus (gold color indicates unsegmented region). For simple and clean model such as the *block*, *cover-rear*, *crank*, *pump-carter* and *stator*, the RANSAC-based results are acceptable, though some primitive types are incorrect and some parts remain unsegmented. Moreover, RANSAC-based approach over-segments the *shaft* and *coupling-down* models. Our result, in contrast, not only identifies correct primitive types but also segments a better level of details.

More complicated models include the *grayloc*, *mechanical part*, *master-cylinder* and *stub-axle*. Our results are clearly better than the RANSAC's in terms of both level of details and segmentation quality. A lot of small parts with accurate primitive types can be

well-captured by our approach while the RANSAC-based method typically either misses or detects with wrong primitive types. The *grayloc*, *mechanical part*, *master cylinder* and *stub-axle* are very challenging because there are many blending regions and a significant amount of noise on both the points' positions the normals. This is the case where the local-based segmentation approaches are often trapped by local sub-optimal primitives (and have no general way to fix them). On the contrary, our algorithm really shines as it exhibits well-defined structural primitives from large to small, even on the curvy surface such as the *master cylinder*.

Fig. 2.1 shows a side-by-side comparison of the quality of the segmentation results between our approach, the RANSAC-based approach and the GlobFit approach. We project the associated points onto the segmented primitives and display the primitives' axes by the black lines. As we can see our method demonstrates its strength in discovering the global relationship between the primitives because we restrict the search space for detecting the primitive and thus avoid the locally optimal primitives. On the other hand, RANSAC-based approach is still subject to local estimation and may produce locally optimal primitives. If the initial RANSAC's segmentation is wrong, it is very hard, if not impossible, for the GlobFit approach to correct it and sometimes GlobFit even makes it worse (e.g. the *grayloc*, *master-cylinder*).

Fig. 2.16 shows the comparison between the RANSAC-based method and ours on the *joint* model under different levels of noise with numerical statistics in Table 2.4. Starting with a clean model, we gradually increase the (Gaussian) noise characterized by the noise level σ. With the clean model (first column) ($\sigma = 0$) or very little noise (second column) ($\sigma = 0.01$), both approaches give consistently good segmentation results. However, when noise increases, RANSAC-based approach tends to produce local optimal primitives (and even wrong primitive types) while our dimension reduction segmentation approach is still robust to a certain degree. Even when the model is heavily corrupted by noise (the last column) ($\sigma = 0.2$), our model's major orientations are still correctly estimated and some

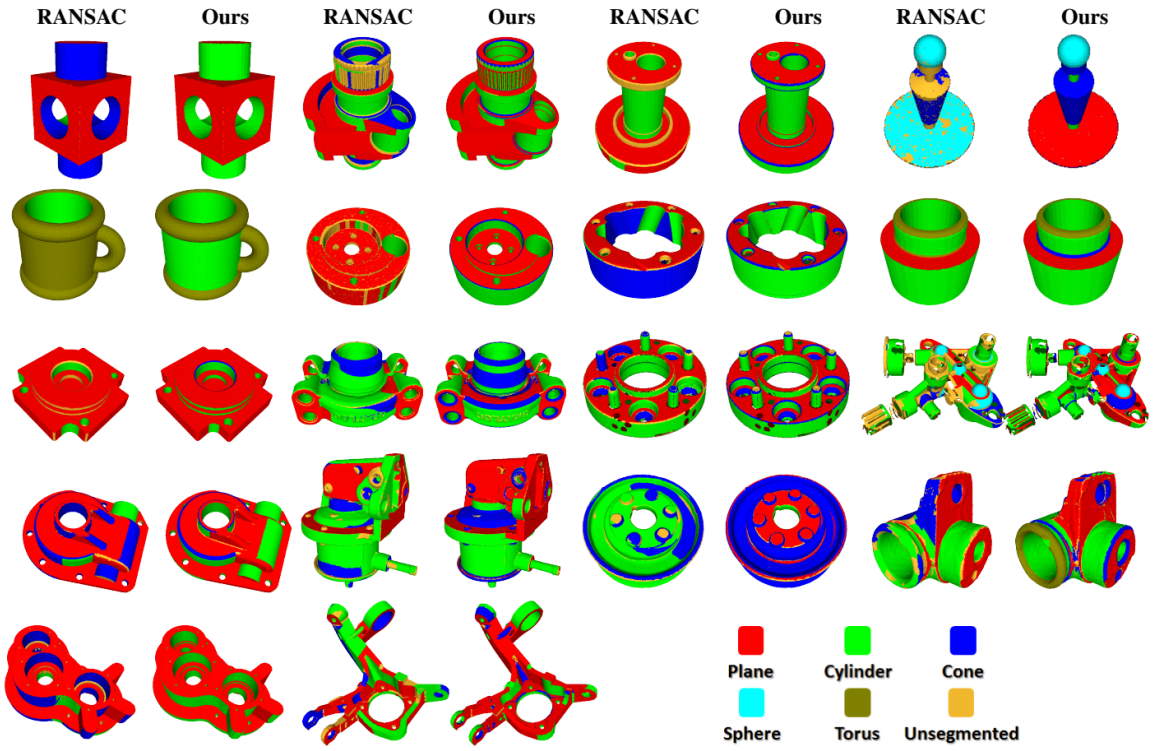


Figure 2.15: Comparison primitive type recognition between our segmentation and RANSAC-based segmentation [7]. **Odd columns**: RANSAC's result; **Even columns**: Our result.

of the primitives are extracted in good shapes.

In short, RANSAC-based method, as mentioned in the introduction often produces over or under segmentation and wrong primitive type. This is because the primitive parameters are calculated locally, which is extremely sensitive to noise, and the model selection is also local. Our approach, on the other hand, narrows the solution to the model's major orientations and by examining the profile curve obtained from the cross-sectional projection, our primitive's parameters are more accurately computed and the model selection is more robust. As a result, our approach can capture small details such as the conic transition between coaxial primitives (e.g. the transition between primitives in the *crank*, *mechanical part*,*master-cylinder* and *shaft* in Fig. 2.15).

For quantitative comparison, some error metrics such as the number of primitives, the coverage percentage, the distance error and the normal error are evaluated for our results,

Table 2.2: Comparison of primitive quality over processed models

Model name	# of primitives			Coverage (percentage)			Distance error ($\times 10^{-3}$)			Normal error (degrees)		
	(I)	(II)	(III)	(I)	(II)	(III)	(I)	(II)	(III)	(I)	(II)	(III)
block	14	14	14	**99.98**	**99.98**	**99.98**	**0.08**	0.37	0.69	**1.24°**	1.33°	1.43°
casting	61	41	40	**98.80**	97.93	84.90	**0.33**	0.53	1.97	**2.28°**	3.53°	5.03°
coupling-down	44	74	67	**99.99**	76.54	80.83	0.27	**0.07**	0.34	**1.07°**	3.84°	1.83°
cover-rear	45	28	28	**100.00**	87.79	87.79	**0.04**	0.11	0.15	**0.51°**	1.38°	3.11°
crank	169	84	83	**99.34**	87.73	80.83	0.83	**0.14**	0.15	**1.68°**	1.75°	1.89°
grayloc	112	60	60	**93.77**	92.60	92.60	**1.69**	4.22	10.64	**6.14°**	10.72°	15.14°
lamp	34	29	29	**100**	96.31	96.31	**0.75**	1.23	1.02	**4.32°**	8.82°	12.04°
master-cylinder	78	32	32	**97.19**	93.81	93.81	**1.66**	2.98	10.61	**8.21°**	10.51°	12.80°
mech. part	179	59	59	**94.05**	76.17	76.17	**1.17**	7.68	7.54	**7.23°**	8.81°	11.32°
mug	5	6	n/a	**99.99**	99.98	n/a	**0.28**	0.34	n/a	**2.47°**	2.50°	n/a
oil pump	155	108	108	**96.00**	86.31	86.31	1.85	1.15	**1.14**	**8.08°**	8.21°	8.20°
pulley	49	49	49	**96.76**	95.25	95.25	**3.60**	4.23	4.49	**9.11°**	12.00°	12.02°
pump-carter	63	57	57	**98.61**	92.87	92.87	0.30	**0.16**	2.30	2.63°	**1.84°**	7.07°
rolling-stage	42	18	18	**99.94**	91.36	91.36	**0.32**	1.16	3.75	**3.43°**	3.79°	6.87°
shaft	12	12	39	**99.80**	84.31	70.71	**0.59**	1.83	1.81	4.17°	4.77°	**3.17°**
stator	12	12	n/a	**100.00**	99.99	n/a	**0.47**	0.80	n/a	**2.87°**	3.12°	n/a
stub-axle	165	91	91	**98.49**	90.05	90.05	**0.68**	1.19	5.02	**4.17°**	4.77°	7.00°
wheel	58	12	12	97.44	**98.02**	**98.02**	**0.21**	0.59	0.42	**3.57°**	6.52°	4.59°

*(**I**): Ours; (**II**): RANSAC; (**III**): GlobFit

Table 2.3: Timings statistics (in seconds) of each step in our method on processed models

Mode namel	**# of points**	**(I)**	**(II)**	**(III)**	**(IV)**	**(V)**	**(VI)**	**(VII)**	**(VIII)**	**Total**
block	850,873	1.01	10.52	0.13	54.98	0.23	21.15	1.31	20.23	109.56
casting	911,321	7.14	101.82	0.16	20.27	0.86	1.62	2.35	51.07	185.29
coupling-down	1,190,493	1.76	14.53	0.16	64.87	0.21	35.54	6.13	13.76	136.96
cover-rear	990,663	1.96	9.63	0.14	43.01	0.12	26.86	3.26	3.74	88.72
crank	1,213,649	9.96	18.58	0.29	26.91	0.84	12.34	14.01	43.76	126.69
grayloc	1,272,891	9.99	270.39	0.32	185.29	0.96	5.78	4.12	112.82	589.67
lamp	212,365	4.13	28.42	0.08	35.58	0.85	4.32	0.25	57.90	131.53
master-cylinder	1,244,445	9.25	70.87	0.26	277.55	0.98	138.36	23.75	56.54	577.56
mech. part	529,006	4.02	258.91	0.23	103.37	0.52	112.23	36.47	79.75	595.50
mug	758,597	3.45	85.74	0.10	65.42	0.41	39.64	1.34	3.63	199.73
oil pump	1,020,244	7.81	147.11	0.68	175.40	0.86	5.90	6.17	81.89	425.82
pulley	1,366,550	10.72	209.38	0.12	29.10	0.69	1.05	1.01	92.18	344.25
pump-carter	1,900,511	3.29	54.74	0.17	69.66	0.90	50.75	7.90	23.45	210.86
rolling-stage	920,433	7.23	49.47	0.15	68.51	0.36	65.52	6.38	45.34	242.96
shaft	1,102,147	2.71	54.67	0.15	83.76	0.12	13.52	6.91	32.45	194.29
stator	1,777,363	3.86	85.96	0.20	23.45	0.14	18.42	4.50	5.64	142.17
stub-axle	1,266,906	9.34	215.74	0.35	204.45	1.52	59.43	32.57	69.65	593.05
wheel	282,534	2.13	63.39	0.09	29.53	0.61	16.43	3.18	9.23	124.59

*(**I**): Patch-based over-segmentation; (**II**): Orientation estimation; **III**: Plane detection; (**IV**): Circle detection; (**V**): Axis estimation; (**VI**): Profile curve analysis; (**VII**): Set cover optimization; (**VIII**): boundary refinement

Table 2.4: Numerical comparison of primitive quality for Fig. 2.16

Noise level	$\boldsymbol{\sigma = 0}$		$\boldsymbol{\sigma = 0.01}$		$\boldsymbol{\sigma = 0.02}$		$\boldsymbol{\sigma = 0.05}$		$\boldsymbol{\sigma = 0.1}$		$\boldsymbol{\sigma = 0.2}$	
	(I)	**(II)**	**(I)**	**(II)**	**(I)**	**(II)**	**(I)**	**(II)**	**(I)**	**(II)**	**(I)**	**(II)**
# primitives	12	12	12	13	12	14	24	41	27	83	40	112
Coverage	**100**	**100**	**99.70**	99.52	**99.66**	98.61	**96.08**	95.87	**91.19**	88.39	**67.54**	60.14
Distance error	**0.17**	0.32	**0.84**	0.92	**1.61**	1.86	**2.26**	3.60	**2.46**	5.90	**2.76**	8.18
Normal error	**1.07**	1.28	**1.13**	1.15	**1.16**	1.19	**1.42**	2.04	**1.95**	4.09	**4.31**	7.72

*(**I**): Ours; (**II**): RANSAC

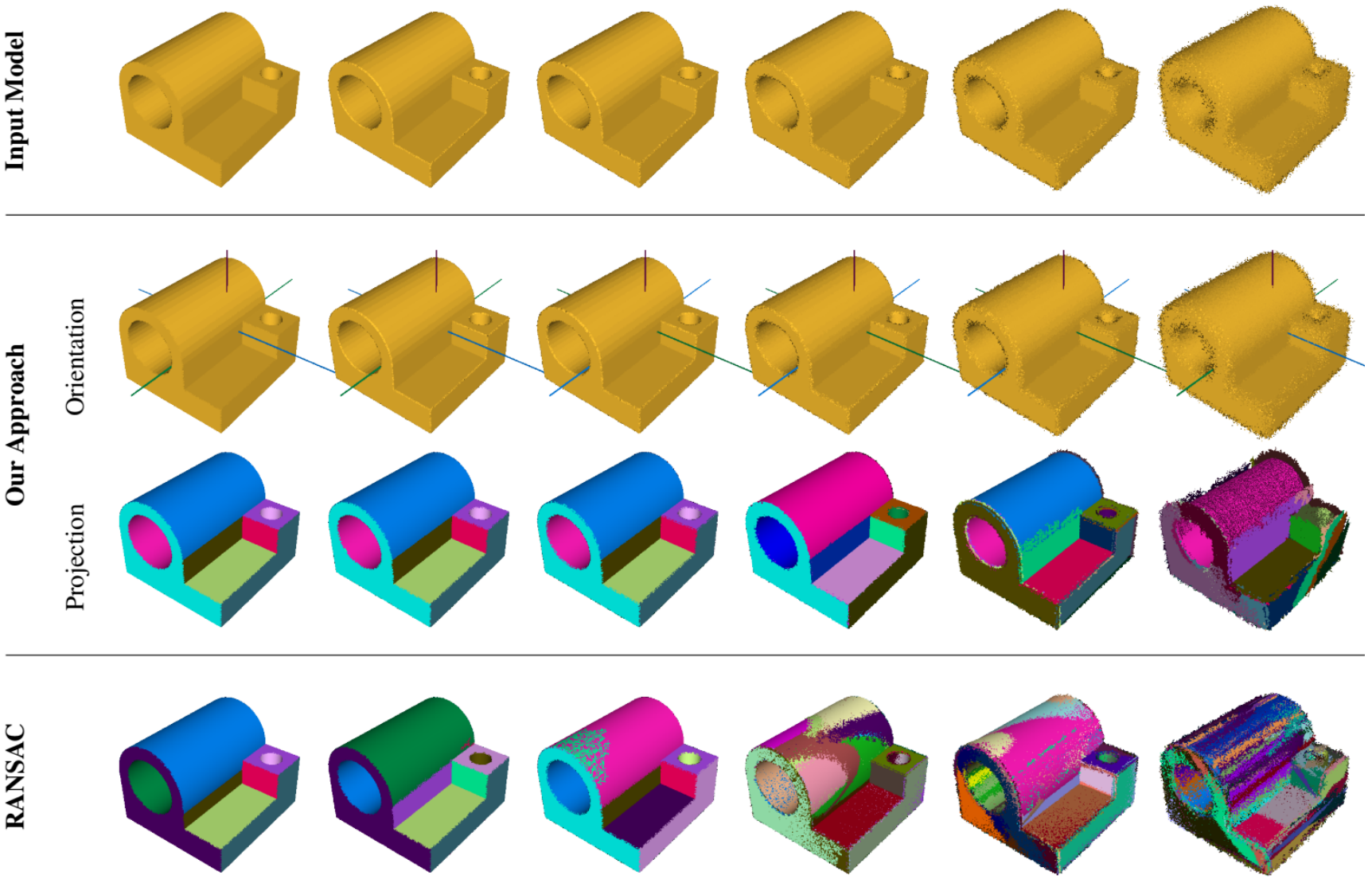


Figure 2.16: Comparison of the robustness between our segmentation and RANSAC-based segmentation [7] on the *joint* model. Noise (on both points' positions and normals) increases from left to right, starting with a clean model.

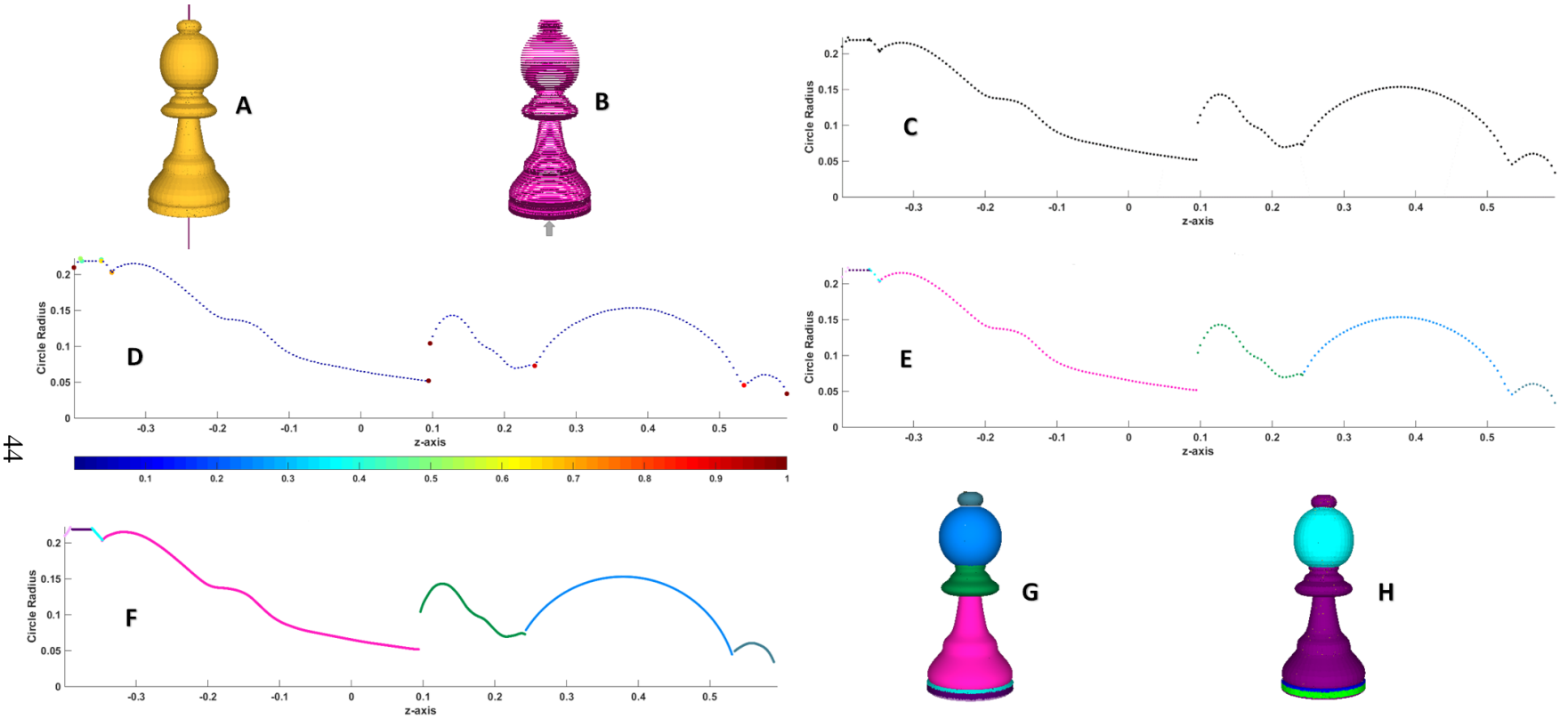


Figure 2.17: Illustration of segmenting profile curve involving surface of revolution. **A**: A *bishop* model with detected axis; **B**: Cross-sectional slicing; **C**: Profile curve from detected circles; **D**: Color map of the (approximated) profile curve's normal variation where critical points are marked bigger; **E**: Segmentation of profile curve; **F**: Profile curve fitted by line segments, circular arcs and cubic splines; **G**: Segmentation result; **H**: Segmentation colored by primitive type.

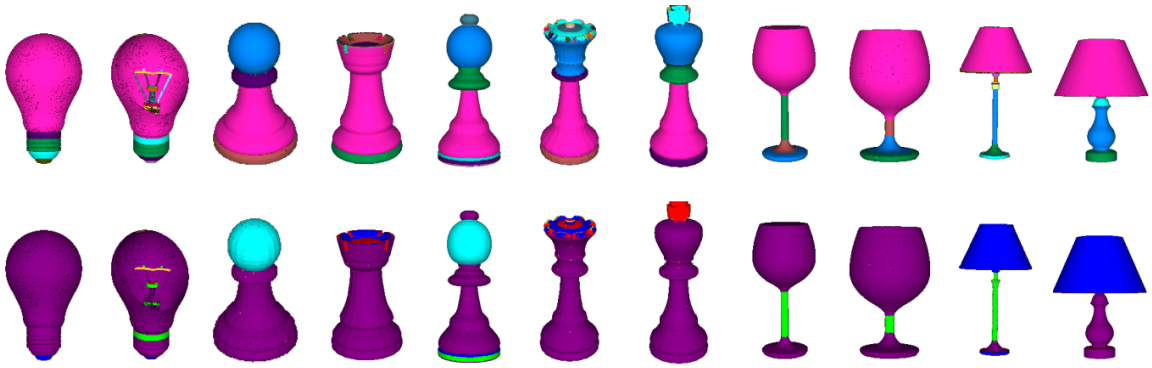


Figure 2.18: Our segmentation algorithm for some models with surface of revolution. **Top**: Random colors for different parts. **Bottom**: Segmentation colored by primitive type (surface of revolution is purple, others are colored similar to Fig. 2.3H).

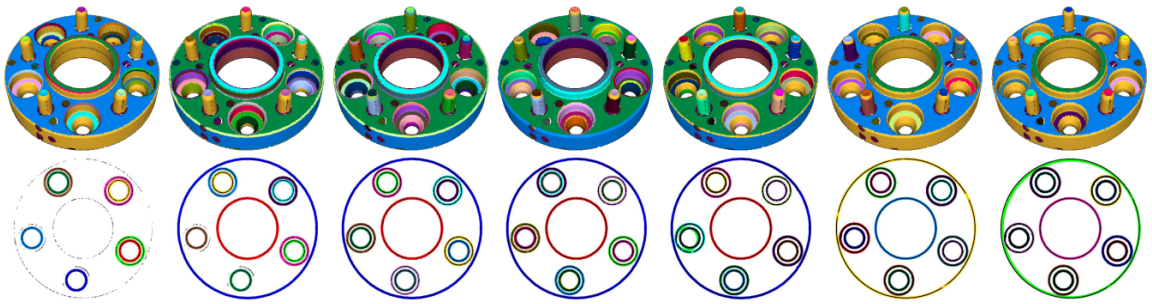


Figure 2.19: Effect of various values of the slice's thickness τ. **Leftmost**: too fine slicing makes the projection image too sparse for circle detection; **Rightmost**: too coarse slicing makes the profile curve too sparse for circular primitive detection.

RANSAC's and GlobFit's (see Table 2.2). There is no doubt that our method yields the best overall quality with very low variance (we cover over at least 90 percent of the model, our distance error is less than 0.5 percent of the diagonal of the point cloud's bounding box and our normal error is less than 10°).

Table 2.3 gives the performance of our algorithm with detailed timings on each step. The performance depends on the complexity of the model rather than its size. The axis estimation and profile curve analysis are very fast because in worst case scenario, they only have to deal with a relatively small input (i.e. grouping several hundreds of circles into connected components and fitting lines to each component for the axis estimation; the profile curve analysis processes as many as 200 points (see Section 2.3.4)). The cross-sectional circle detection is the most time-consuming stage in our framework. The Graph-Cut smoothing also accounts for a moderate amount of time. It is worth noticing that the major orientation estimation and cross-sectional circle detection steps are both highly amenable to parallelization and, hence can be made significantly faster. With our sequential implementation in an unoptimized `MATLAB` code, it takes less than 10 minutes to completely segment a 1M-point model.

The proposed method can be easily extended to surface of revolution such as hyperboloid, ellipsoid, paraboloid, etc. by extending the segmentation of the profile curve into polynomial curve segments or splines. Fig. 2.17 shows an example of the *bishop* model with cylinder, cone, sphere and surface of revolution together. Since surface of revolution is too flexible, it is sometimes ambiguous to differentiate between a surface of revolution and a combination of many cylinders, cones, sphere and tori. As a result, we opt to identify the *critical points* on the profile curve first. A critical point represents discontinuity in the first derivative. In order to quantify this measurement, starting with the profile curve constructed by cross-sectional circle detection, we approximate its normal variation as follows. For each point on the profile curve, we fit two lines to its *left* and *right* neighbors, respectively and use the angle difference between them as the normal variation. As we

can see from Fig. 2.17D (we normalize the normal variation to $[0, 1]$ interval), the points on smooth parts have low normal variation while the points at discontinuities in the first derivative have large normal variation. We can obtain the critical points by first detecting local maxima of the normal variation followed by thresholding (e.g. 0.4). After the critical points are identified, the profile curve can be broken into smaller parts where each part can be fitted by a line segment, circular arc or cubic splines (Fig. 2.17E and F). The segmentation result is shown in Fig. 2.17G and H. More examples involving surfaces of revolution can be found in Fig. 2.18.

2.4.1 Limitation and Future Work

Our method has some limitations. First, we currently use a uniform slicing along the major direction, which is not adaptive to the data's sampling rate or level of details. A too coarse slicing prevents us from detecting small details while a too fine one makes the projection image too sparse for circles to be detected (Fig. 2.19), both of which result in miss-detection. However, we could generate multiple hypothetical primitives based on various thresholds and put all them into the set-cover optimization (at the cost of extra computation). Similar strategy could be applied to the distance (ε) and normal (α) thresholds. In the future work, we plan to explore the adaptive sampling techniques so that the slice's thickness could be adapted to the data density.

Second, our method in its current form supports plane, cylinder, cone, sphere, torus and extendable to surface of revolution. It, however, can be further extended to generalized cylinders with curilinear axis. Indeed, instead of fitting a single straight line to circle centers within each connected component (Section 2.3.4), we could fit multiple line segments or even curvilinear axis which aims at generalized cylinders. Another potential future work is to fit ellipse or superellipse [86] instead of circles in the 2D cross-sectional image. With these extensions we believe the proposed framework can be extended to segment a much broader class of shapes.

2.5 Conclusion

We have presented our novel primitive-based segmentation algorithm for mechanical CAD models consisting of planes, cylinders, cones, spheres, tori and extendable to surfaces of revolution. Our approach first over-segments the input model, then estimates the model's major orientations and generates an over-complete set of planar and circular primitives which are then selected by the set cover optimization. For each orientation, we decompose the model into connected components of circular primitives of the same axis, then further divide each connected component into individual circular primitives by constructing a profile curve for each connected component and segmenting the profile curve into circular arcs and line segments. The final segmentation is obtained from the set cover optimization and finished by the boundary refinement. Experimental results on both synthetic and real scanned models show that our approach compares favorably with existing methods such as the RANSAC-based [7] and GlobFit [8] approaches in terms of robustness and accuracy. In the future, we plan to extend our method to handle a broader class of primitives as discussed in Section 2.4.1.

Acknowledgment

We wish to thank the authors of the AIM@SHAPE Shape Repository and the Princeton Shape Benchmark [87] for providing us the tested models. We also appreciate the authors of the RANSAC-based approach [7] and GlobFit [8] for making their code publicly available.

Chapter 3

A Multi-view Recurrent Neural Network for 3D Mesh Segmentation

This work introduces a multi-view recurrent neural netowrk (MV-RNN) approach for 3D mesh segmentation. Our architecture combines the convolutional neural networks (CNN) and a two-layer long short term memory (LSTM) to yield coherent segmentation of 3D shapes. The imaged-based CNN are useful for effectively generating the edge probability feature map while the LSTM correlates these edge maps across different views and output a well-defined per-view edge image. Evaluations on the Princeton Segmentation Benchmark dataset show that our framework significantly outperforms other state-of-the-art methods.

3.1 Introduction

Mesh segmentation is a classical, yet challenging problem in computer graphics for many decades. Unfortunately, the segmentation problem is ill-posed and there is no general objective measurement that can universally be applied in any case. Judging the quality of a segmentation largely depends on application. For instance, in a LiDAR scan of urban environment, a desired segmentation should distinguish between different instances of build-

ings, people, cars, trees, ground, etc. However, in a part-based annotation of a 3D model (e.g. human), the requirement is usually to segment head, torso, left/right arms, left/right legs and sometimes more details such as thumb, index finger, and so on depending on specific task. Consequently, in the scope of this work, we aim to tackle the mesh segmentation as a data driven approach. Given a training dataset of input mesh and the corresponding desired segmentation, we design a deep learning framework to learn the pattern of the segmentation given by the training dataset so that it can segment an unseen mesh. As a result, we make no geometric or topological assumptions about the shape, nor exploit any hand-crafted descriptors.

In this work, we propose a multi-view recurrent neural network (MV-RNN) deep learning framework to segment 3D model which significantly outperforms prior methods on the Princeton Segmentation Benchmark dataset [21]. It is worth mentioning that our goal is to partition the 3D model and not to do the semantic segmentation. In semantic segmentation, the two wings of an airplane are assigned a single label *wing*. On the other hand, in mesh segmentation, the two wings belong to two different regions and do not have semantic label. In general, semantic segmentation provides better understanding of a 3D model. However, mesh segmentation still has its merits such as guiding mesh processing algorithms including skeleton extraction [88, 46], modeling [89], morphing [90], shape-based retrieval [26] and texture mapping [91]. Moreover, in contrast to semantic segmentation which requires a fixed set of semantic labels, many mesh segmentation algorithms could be generalized to unseen object categories. As a result, instead of identifying surface area of the 3D model within a segment, we predict its boundary (or edge). The benefits of doing so are twofold. First, it is usually more expensive to obtain dense surface annotations than boundary annotations from humans. Second, we only have two semantic labels, i.e. boundary versus non-boundary, which is simpler for the framework to learn than using hundreds of semantic labels (e.g. hand, torso, leg, head, etc.). In fact, detecting 3D edges could be useful for other tasks such as suggestive contours [92, 93] and ridge-valley detection [94].

Our approach belongs to the multi-view paradigm which has been shown success recently for many visual recognition tasks such as classification and segmentation [95, 96, 97, 98]. Typically, in the multi-view segmentation, a 3D model is rendered with multiple views to generate multi-view images, each of which is fed-forward to a (shared weights) convolutional neural network to obtain densely labeled images before being mapped back to 3D. In general, a multi-view approach for segmentation must overcome several technical obstacles. Firstly, there must be enough views to minimize occlusions and cover the shape surface. This can be achieved by generating a large number of views equally distributed around the object. Secondly, shape parts can be visible from more than one view, thus, the proposed method should effectively correlate information from multiple views. The main drawback of the existing multi-view approaches such as the multi-view convolution neural network (MV-CNN) [95, 96] is that different views may not be correlated and hence a 3D area may correspond to totally different outcomes from different views. Let us take an example of a standing person rotating counter-clockwise (Fig. 3.1). When the view is front facing, the boundary between the torso and the right arm is a real boundary. At certain time, the right arm starts to be occluded. Then the boundary between the torso and the right arm is no longer real boundary, but the MV-CNN cannot distinguish them because it does not correlate the result over different views.

We propose MV-RNN to overcome this limitation by treating the sequential multiple views as a temporal sequence, and applying recurrent neural network to capture the redundancy between adjacent views. More specifically, in this work we employ the long short term memory (LSTM) as the recurrent neural unit. The multi-view outputs from CNN are correlated through a two-layer LSTM to obtain consistent fine detail responses for every view. Finally, the boundary pixels are back-projected onto 3D shape surface followed by region growing and Conditional Random Field (CRF) to obtain the final segmentation. The main contribution of our work is the MV-RNN, which is, to the best of our knowledge, the first network treating multiple views as a temporal sequence and applying LSTM to

Figure 3.1: Given a 3D model, we try to detect boundary between segments by using multi-view approach. We apply non-maximum suppression [16] to the MV-CNN results shown on the second row for visualization. The main drawback of MV-CNN is its inconsistency across multiple views (e.g. the elbow and arm regions). On the other hand, our MV-RNN could correlate multiple views and generate more coherent results.

correlate adjacent views. Moreover, since the proposed framework is purely data driven, it can be easily adapted or extended to other tasks in shape modeling such as suggestive contours [92, 93] and ridge-valley detection [94].

In the next section, we briefly discuss existing methods related to 3D segmentation with emphasis on deep learning. To make the work self-contained, we review the recurrent neural network in Section 3.3. Section 3.4 describes our approach in depth followed by experimental results in Section 3.5. Section 3.6 concludes our work.

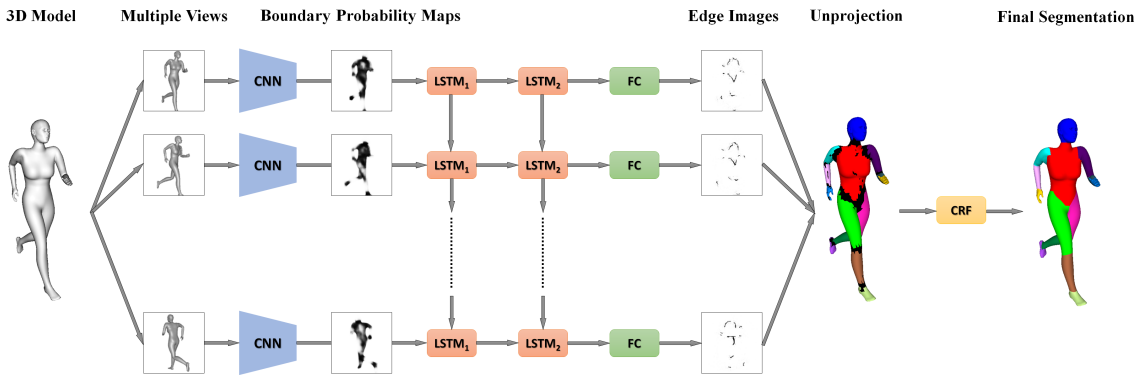


Figure 3.2: Overview of our MV-RNN approach. Given an input 3D mesh model, we render it with a sequence of ordered viewpoints. Each of view is passed through an identical (shared weights) CNN to obtain a boundary probability map, which is correlated by a two-layer LSTM followed by a fully connected layer. The consistent edge images from multiple views are unprojected back to 3D followed by a region growing and CRF for boundary smoothing.

3.2 Related Work

Hand-crafted features: Before the era of deep learning, people proposed many approaches (region growing [41, 42], hierarchical clustering [44, 45, 46], spectral clustering [48], k-means [47], normalized cut [99], random walk [52], heat walk [53], etc.) based on local features to segment a 3D model such as planarity of various forms, higher degree geometric proxies (cylinders, cones, spheres, etc.), dihedral angles between triangles [38], curvatures (Gaussian curvature or mean curvature) [39], geodesic distances on a mesh, slippage, symmetry, convexity, medial axis, shape diameter [40] and motion characteristics [20]. Shamir et al. [20], Agathos et al. [36] and Theologou et al. [37] gave a comprehensive overview of methodologies in 3D segmentation. In general, these approaches are usually built on some particular property of 3D objects and hence do not generalize well.

Image-based CNN: CNNs [100, 101, 102, 103] are currently the main stream in many visual recognition problems and have been extensively applied to image semantic segmentation [104, 105, 106, 107, 108]. For example, fully convolutional network (FCN) [105] was a breakthrough in deep learning based image semantic segmentation. In this approach, fully connected layers in a standard CNN are replaced by convolutions with large receptive

fields, and segmentation image is achieved using coarse class score maps obtained by feed forwarding an input image. However, the deconvolution part of the network responsible for upsampling is fixed to bilinear interpolation and only the CNN part of the network is fine-tuned. In contrast, Noh et al. [106] proposed the deconvolution network (DeconvNet) with unpooling layers followed by convolutions, which increases the network's capability to learn more complex deconvolution than using just bilinear interpolation.

The holistically nested-edge detection (HED) [109] casts the classical edge detection as a CNN-based problem. An interesting idea of this work is that the final edge map is fused from multiple edge maps obtained at different scales. The multi-scale edge maps are side outputs of a VGG-16 network [101] and hence the shallow edge maps give fine detail edges while the deeper ones capture the more salient edges. The final result is linearly combined from all edge maps at multiple scales. Our MV-RNN approach adopts HED as a sub-module for our CNN part thanks to its high performance on natural images.

Deep learning for 3D: While deep learning has been very popular in 2D images for many years, it has just been applied in 3D recently because unlike pixels in 2D images, 3D objects do not have regular structure. As a result, in the early period, people use deep learning as a tool to learn high level features from low level cues (usually hand crafted). The unsupervised shape segmentation proposed by Shu et al. [110] starts by over-segmenting the input model, computing patch-based local features and then uses stacked auto-encoder to learn high level features followed by Graph-Cut based segmentation. Guo et al. [111] compute local features at different scales for each triangle and arrange them into a rectangular image, which is feed forward through a convolutional neural network (CNN) to predict the semantic label for each triangle. Although these two frameworks use deep learning techniques (stacked auto-encoder, CNN) to learn high level features from local low level ones, they do not exploit the full potential of deep learning.

A natural extension from 2D image to 3D shape is to discretize the 3D object into 3D voxel and apply 3D convolutions on it. The 3D ShapeNet [4] used this approach for

3D object classification. Su et al. [95] was the first one to apply multi-view convolutional neural network (MV-CNN) for 3D recognition. The 3D shape is rendered in multiple views, each of which is passed through the identical image-based CNN. Features obtained from multiple views are combined via a view pooling (which is the max pooling) and then passed through another CNN to predict the final object label. Between volumetric and multi-view CNN, the later typically gives higher accuracy [97]. One reason might be due to the higher computation and memory cost of using 3D convolutions which in turn limits the image resolution [97]. A similar result has also been observed in other 3D data such as videos [112, 113, 114].

Xie et al. [96] used multi-view depth images via extreme learning machine to generate per-view segmentation and combine them via Graph-Cut. This method works pretty fast due to the easy training of the extreme learning machine but it does not give high accuracy. Later, Kalogerakis et al. [115] proposed a more complete multi-view framework. They first render the 3D model with different views, each of which is processed through a shared CNN before unprojected to 3D. The label consistency is solved by a conditional random field (CRF), which is part of a network and is optimized in an end-to-end manner. Although this approach uses the CRF to solve the consistency after unprojection to 3D, the semantic label images from multiple views are obtained in a max-pooling manner and they are still not correlated.

Recently, Su et al. proposed the PointNet [116] and SyncSpecCNN [117] which consume directly non-regular 3D data (point cloud and mesh, respectively). These networks demonstrate the flexibility of neural networks in many visual problems. However, in term of performance, these structures still fall behind MV-CNN approaches (if equipped large enough number of views) [116].

3.3 Background on Recurrent Neural Network

In contrast to normal feed-forward neural network which is a one-shot function, recurrent neural network (RNN) runs repeatedly through time which simulates human brain processing capability. An RNN is a composition of identical feed-forward neural networks, one for each moment, or step in time, which we will refer to as RNN cells. These cells operate on their own output, allowing them to be composed. They can also operate on external input and produce external output. Note that this is a much broader definition of an RNN depending on the choice of RNN cells (e.g. Vanilla RNN, LSTM, etc.). Here is the algebraic description of a Vanilla RNN cell.

$$s_t = \phi\left(Wx_t + Us_{t-1} + b\right) \tag{3.1}$$

where ϕ is the activation function (e.g. sigmoid, tanh, ReLU [118, 100]); Assuming d and h are the state input and output sizes, respectively, $s_t \in \mathbb{R}^h$ is the current state (and current output); $s_{t-1} \in \mathbb{R}^h$ is the prior state; $x_t \in \mathbb{R}^d$ is the current input; $W \in \mathbb{R}^{h\times d}$, $U \in \mathbb{R}^{h\times h}$ and $b \in \mathbb{R}^h$ are weights and biases.

Although being simple and quite powerful, Vanilla RNN has certain disadvantages. First, it is very difficult to exploit post information if information constantly morphs, which leads to the degeneration problem [103]. Second, gradient vanishing and exploding are common in training Vanilla RNN because we train it by the back-propagation over time algorithm. If the gradients explode, we cannot train our model. If they vanish, it is difficult for us to learn long-term dependencies, since back-propagation will be too sensitive to recent distractions.

To tackle the drawbacks of Vanilla RNN, the long short-term memory (LSTM) unit [119] was introduced to ensure the integrity of information thanks to its written memories. Furthermore, LSTMs use gates as a mechanism to selectively control and coordinate writing (i.e. the cell memory is written, read and forgot selectively).

Unlike Vanilla RNN, an LSTM network is well-suited to learn from experience to classify, process and predict time series when there are time lags of unknown size and bound between important events. Consequently, LSTM achieved the best known results in natural language text compression, unsegmented connected handwriting recognition. Recently, researchers have been integrating LSTMs to computer vision tasks such as image segmentation [120], activity recognition, image captioning, video description [114], 3D object reconstruction [98].

3.4 Multi-view Recurrent Neural Network (MV-RNN)

Given an input 3D shape, our goal is to segment it into parts based on the prior knowledge learned from a pre-segmented training dataset. We design a MV-RNN network to this end. Our network architecture is visualized in Fig. 3.2. It takes as input a set of images from multiple views which are equally distributed over the 3D model; segments these images by generating per-view boundary probability maps; correlates them by a two-layer LSTM followed by a fully connected layer and returns the consistent edges which are back projected to the 3D surface and finally integrated by a CRF. In the following sections, we elaborate the input to our network, its layers and the training procedure.

3.4.1 Input

The input to our whole algorithm is a 3D shape represented as a polygonal mesh. As a preprocessing step, we normalize and scale it to fit into the unit sphere. Then our algorithm renders the object in K different views (we set $K = 60$ based on our experiments). We first equally partition the unit sphere into K regions using [10]. These regions serve as camera locations. More importantly, to make these views learnable for LSTM, we arrange these locations in sequence so that adjacent locations are next to each other such as in Fig. 3.3. To

make all views oriented consistently, we choose the camera up vector pointing to a very far away fixed point (e.g. $[0, 0, 100]$). The camera always looks at the origin since the model is normalized.

In general, CNN is quite robust to lighting illumination, so we render shaded, grayscale images using Phong reflection model [121] with light source always behind the camera for every view. We also experimented with depth images (with HHA encoding [122]), normal images and/or combined them together but the result is not better than using the shading images alone. To make the training faster, we opt to use image resolution of 128×128 without sacrificing the overall segmentation accuracy of the framework.

In addition, for each camera setting, we store the 3D vertex corresponding to each pixel. The correspondence is determined by the proximity of the 3D point unprojected from the 2D pixel and the closest 3D vertex (the distance between them must be less than 10^{-3}, otherwise there is no corresponding 3D vertex with that pixel). The stored information is used for the back projection later on.

3.4.2 CNN Module

The shaded images produced in the previous step are processed through identical image-based CNN. There are many choices of CNN architecture such as FCN [105], DeconvNet [106] and HED [109]. We opt to choose HED because of its edge detection nature. Each HED module outputs a grayscale image of the same size as the input shaded image (i.e. 128×128), which is the boundary probability map. Specifically, in our implementation, we employ the HED architecture suggested in [109], which adopted the VGG-16 network [101] for dense prediction by truncating after the `pool5` layer and fusing multiple side outputs. Since the HED is trained on RGB color images, we need to replicate our shaded grayscale images into three channels.

Fig. 3.4 shows the boundary probability maps in multiple views (only four views are shown here). As we can see the probability maps are not well-localized nor consistent.

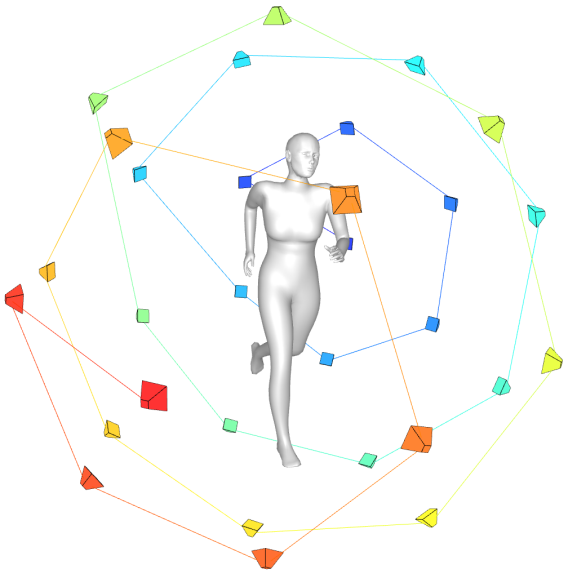


Figure 3.3: Multiple views ordered in a helix-like sequence.

The inconsistency problem cannot be solved by optimizing individual view alone, but by aggregating them together in a a more intelligent way. Recurrent neural networks (RNN) represent a type of neural networks with loop connections [123], which allow them to capture long-range dependency by gates and memory structures (such as LSTM [119]). In consequence, multiple views can be cast as time series which can be learned by such LSTM.

3.4.3 LSTM Module

As mentioned in Section 3.1, the goal of this layer is to correlate multiple views and generate consistent boundary maps. An LSTM network is well-suited here which treats view sequence as time series. First, we unroll the 2D boundary probability maps and ground truth boundary maps into vectors of size $128 \times 128 = 16{,}384$. A two-layer LSTM (with one LSTM stacked over the other) is deployed so that the first LSTM takes the sequence of ordered (unrolled) boundary probability maps, produces a sequence of hidden states for the second LSTM to eventually output the sequence of coherent boundary maps. We use the same number of hidden units (1024) for both peephole LSTMs [124] with the following

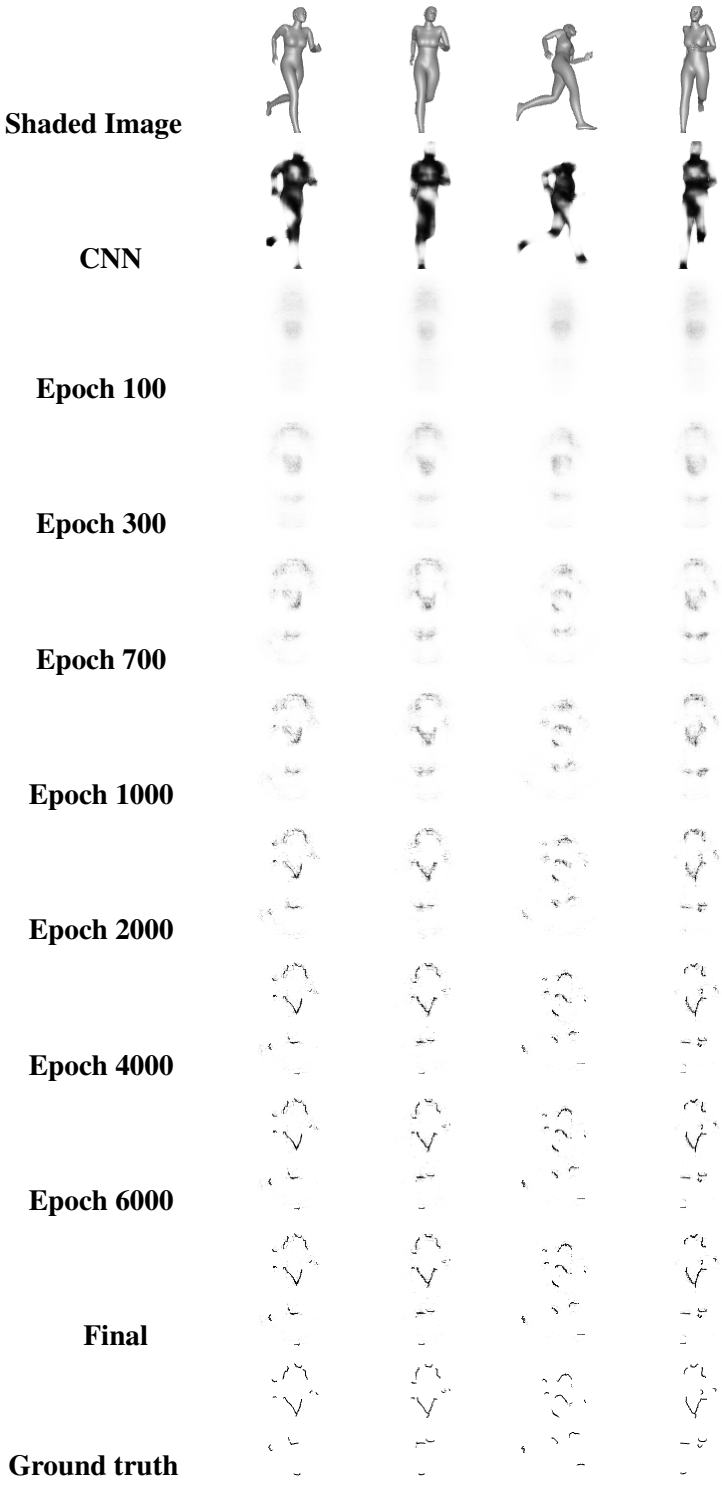


Figure 3.4: LSTM learning process (only four views are shown due to space limit). **First row**: Input shaded images to a CNN. **Second row**: Outputs from CNN. **Third to Tenth rows**: Edges returned from LSTM during training. **Last row**: Ground truth edges.

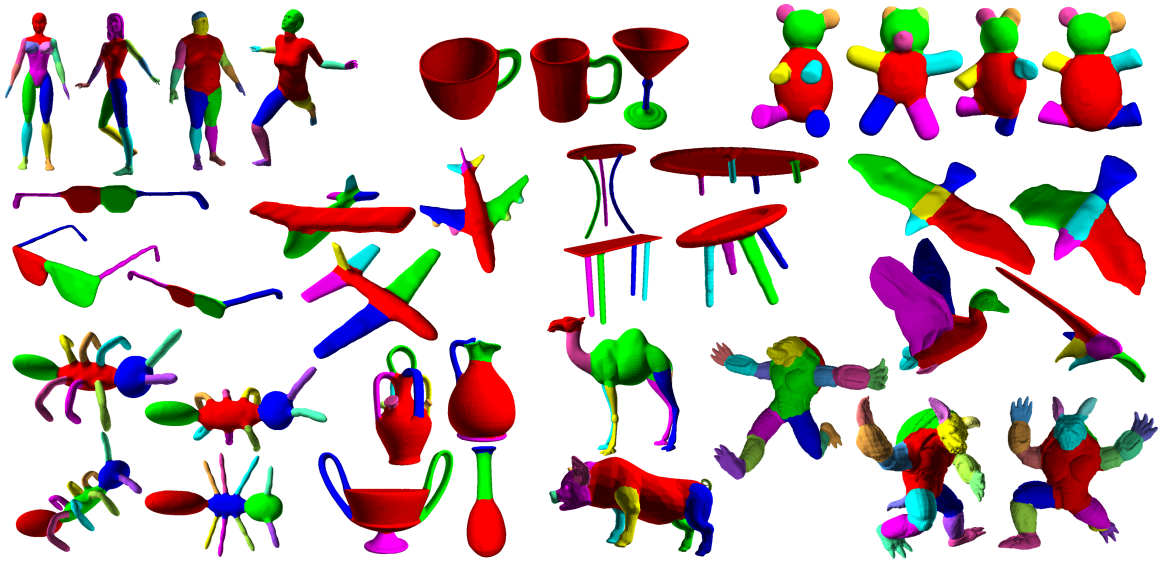
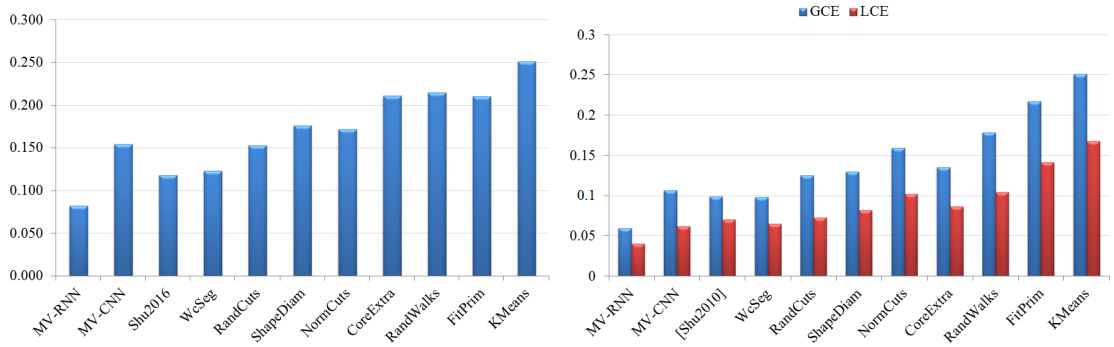
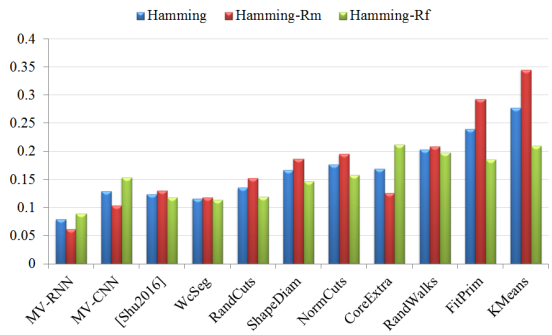


Figure 3.5: Representative segmentation results produced by our MV-RNN on PSB dataset.

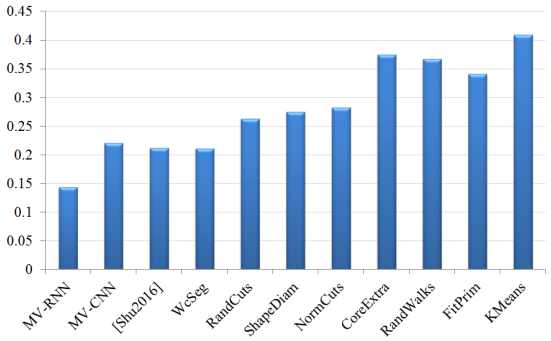


(a) Rand Index

(b) Consistency Error



(c) Hamming Distance



(d) Cut Discrepancy

Figure 3.6: Performance plots of different segmentation algorithms with respect to four evaluation metrics. Lower value is better.

updates.

$$i_t = \texttt{sigmoid}\left(W_i x_t + U_i c_{t-1} + b_i\right) \tag{3.2}$$

$$f_t = \texttt{sigmoid}\left(W_f x_t + U_f c_{t-1} + b_f\right) \tag{3.3}$$

$$o_t = \texttt{sigmoid}\left(W_o x_t + U_o c_{t-1} + b_o\right) \tag{3.4}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \texttt{tanh}\left(W_c x_t + b_c\right) \tag{3.5}$$

$$h_t = o_t \circ c_t \tag{3.6}$$

where $x_t \in \mathbb{R}^d$ is the current input; $h_t \in \mathbb{R}^h$ is the current output; c_t (and c_{t-1}) $\in \mathbb{R}^h$ are the current (and prior) memory state, $W_i, W_f, W_o, W_c \in \mathbb{R}^{h\times d}$, $U_i, U_f, U_o, U_c \in \mathbb{R}^{h\times h}$, $b_i, b_f, b_o, b_c \in \mathbb{R}^h$ are weights and biases and '$\circ$' denotes element-wise multiplication. In our case, $d = 128 \times 128 = 16{,}384$ and $h = 1024$. The output of the second LSTM is passed through a fully connected layer to map back to d-dimension edge image.

Fig. 3.4 illustrates how LSTM can help correct and correlate the edge probability maps produced from the MV-CNN. For example, the boundaries between the torso and two legs are quite different among four views, which may result in inconsistent edge information when unprojecting them to 3D mesh. However, as the LSTM consumes the whole view sequence, the edges at convergence are all consistent.

3.4.4 Training

We train our network in a two-stage approach. In the first stage, we train the HED module. We randomly rotate each 3D model in 16 different ways. The network takes as input a pair of two images, shaded image and ground truth boundary map. We use the sigmoid cross-entropy loss for all five side outputs and the fused output. The network is initialized from VGG-16 weights [101]. We use Adam optimizer [125] with fixed learning rate 10^{-7},

batch size of 16 and train for 100,000 iterations. The first stage training takes three days on an NVIDIA Titan X.

After the HED module is trained, it is fixed for training the LSTM module in the second stage. The two-layer LSTM takes as input a pair of sequences of boundary probability maps from the HED and ground truth boundary maps. We also use Adam optimizer [125] with initial learning rate 0.01 (as this optimization algorithm is able to compute adaptive learning rates for each parameter), batch size of 1 (due to memory limit) and train for 7000 epochs. Each view sequence is processed bidirectionally, which yields two sequences per shape. The second stage training takes three days on an NVIDIA Titan X.

3.4.5 Back Projection to 3D and Post-processing

The consistent boundary maps produced from LSTM network are back projected to 3D surface using the stored pixel-to-vertex information (see Section 3.4.1). It is possible that many pixels (typically from different views) map to the same vertex, so we take the maximum response as the final value. For each edge of the mesh model, we assign the boundary probability which is defined as the average of the boundary probabilities of the two vertices that it connects. Finally a binary boundary edge map is created by thresholding (we set the threshold as 0.5). These boundary edges function as the borders of the regions to be segmented. Thus, we use a simple region growing to find the initial segmentation with the boundary edges as blockers. A region with big enough area is considered as a segment. The polygons near the boundaries may be unlabeled due to projection error. Denote h_v as the initial label for polygon v, where $h_v = 0$ if v has no label. We expect that correct labels will be propagated to them via a CRF. Let V be the set of all polygons in a 3D shape, a CRF f with unary and pairwise terms operating on the surface representation is defined as follows.

$$E(f) = \sum_{v \in V} E_{\text{unary}}(f_v) + \sum_{(u,v) \in V^2} E_{\text{pairwise}}(f_u, f_v) \tag{3.7a}$$

Table 3.1: The Rand Index scores of segmentation for each category with different methods. Smaller is better.

Object Catergories	MV-RNN	MV-CNN	[Shu2016] [110]	WcSeg [126]	Rand Cuts	Shape Diam	Norm Cuts	Core Extra	Rand Walks	Fit Prim	KMeans
Human	**0.106**	0.196	0.116	0.128	0.131	0.179	0.152	0.225	0.219	0.153	0.163
Cup	0.100	0.100	**0.096**	0.171	0.219	0.358	0.244	0.307	0.358	0.413	0.459
Glasses	**0.066**	0.115	0.173	0.173	0.101	0.204	0.141	0.301	0.311	0.235	0.188
Airplane	**0.085**	0.157	0.150	0.089	0.122	0.092	0.186	0.256	0.248	0.166	0.211
Ant	0.021	0.044	**0.001**	0.021	0.025	0.022	0.047	0.065	0.068	0.086	0.131
Chair	0.051	0.078	**0.040**	0.103	0.184	0.111	0.088	0.187	0.156	0.212	0.213
Octopus	**0.022**	0.060	0.036	0.029	0.063	0.045	0.061	0.051	0.067	0.101	0.101
Table	0.072	0.091	**0.040**	0.091	0.383	0.184	0.093	0.244	0.131	0.181	0.369
Teddy	0.035	0.055	**0.024**	0.056	0.045	0.057	0.121	0.114	0.128	0.132	0.182
Hand	**0.076**	0.122	0.135	0.116	0.090	0.202	0.155	0.155	0.189	0.202	0.154
Plier	**0.054**	0.143	0.151	0.087	0.109	0.375	0.183	0.093	0.230	0.169	0.263
Fish	**0.146**	0.253	0.288	0.203	0.297	0.248	0.394	0.273	0.388	0.424	0.413
Bird	**0.059**	0.119	0.171	0.101	0.107	0.115	0.184	0.124	0.250	0.196	0.190
Armadillo	**0.060**	0.120	0.073	0.081	0.092	0.090	0.116	0.141	0.115	0.091	0.117
Bust	**0.162**	0.351	0.275	0.266	0.232	0.298	0.316	0.315	0.298	0.300	0.334
Mech	0.121	0.369	**0.073**	0.182	0.277	0.238	0.159	0.387	0.211	0.306	0.425
Bearing	0.080	0.104	**0.056**	0.122	0.124	0.119	0.183	0.398	0.246	0.188	0.280
Vase	**0.106**	0.216	0.212	0.161	0.133	0.239	0.236	0.226	0.246	0.257	0.387
FourLeg	**0.135**	0.213	0.140	0.152	0.174	0.161	0.208	0.191	0.218	0.185	0.193
Average	**0.082**	0.154	0.118	0.123	0.153	0.176	0.172	0.211	0.215	0.210	0.251

Table 3.2: Average cut discrepancy, hamming distance, consistency error scores of segmentation for each category with different methods. Smaller is better.

	MV-RNN	MV-CNN	[Shu2016] [110]	WcSeg [126]	Rand Cuts	Shape Diam	Norm Cuts	Core Extra	Rand Walks	Fit Prim	KMeans
Cut Discrepancy	**0.144**	0.220	0.212	0.211	0.263	0.275	0.282	0.375	0.367	0.341	0.409
Hamming	**0.075**	0.129	0.124	0.116	0.136	0.166	0.177	0.169	0.203	0.239	0.277
Hamming-Rm	**0.061**	0.104	0.130	0.118	0.152	0.187	0.195	0.126	0.209	0.293	0.345
Hamming-Rf	**0.089**	0.153	0.118	0.114	0.119	0.146	0.158	0.213	0.198	0.186	0.209
GCE	**0.060**	0.107	0.099	0.098	0.126	0.130	0.159	0.135	0.179	0.217	0.251
LCE	**0.041**	0.062	0.070	0.065	0.073	0.082	0.102	0.086	0.104	0.142	0.168

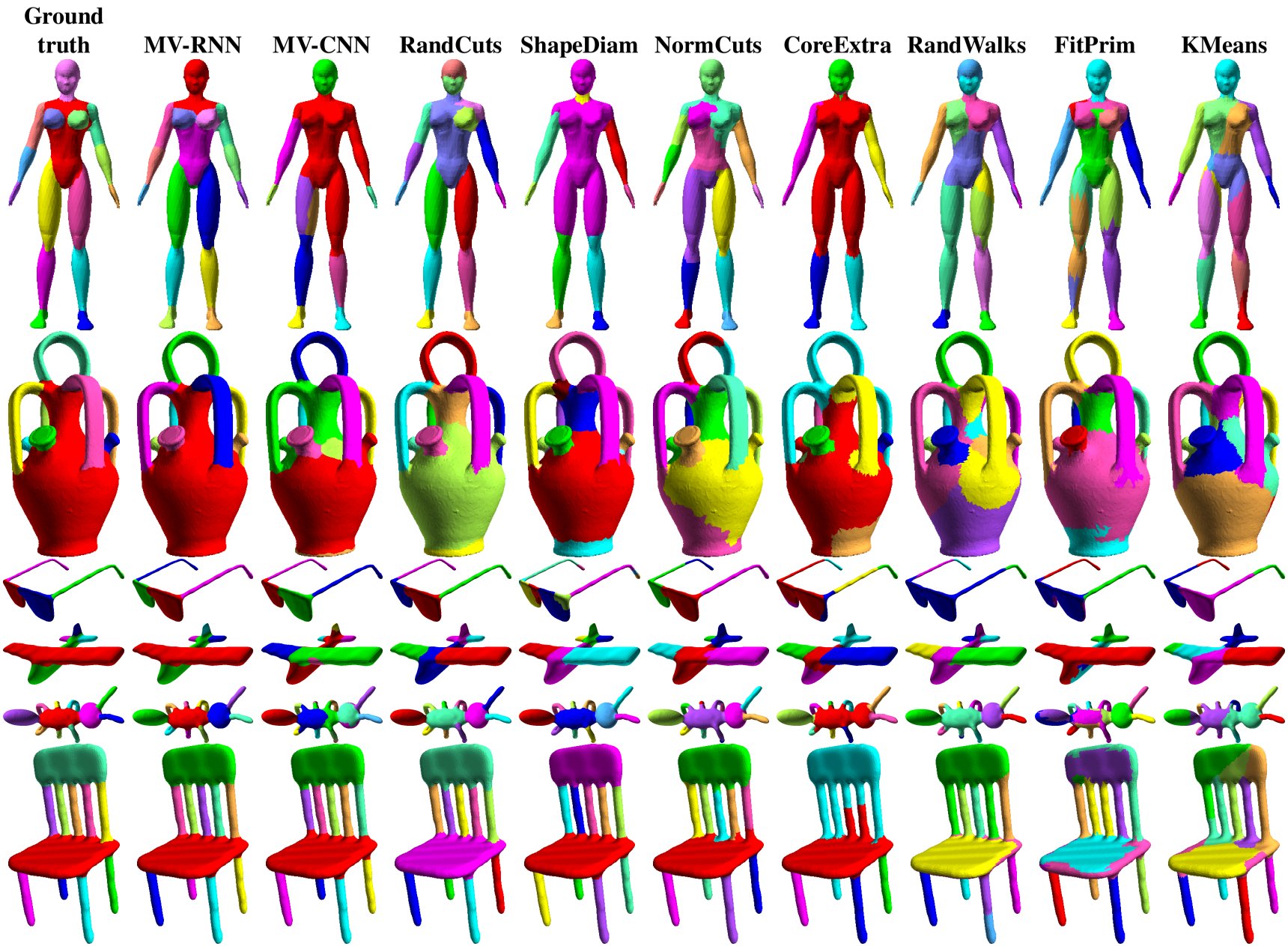


Figure 3.7: Comparison of segmentation algorithms.

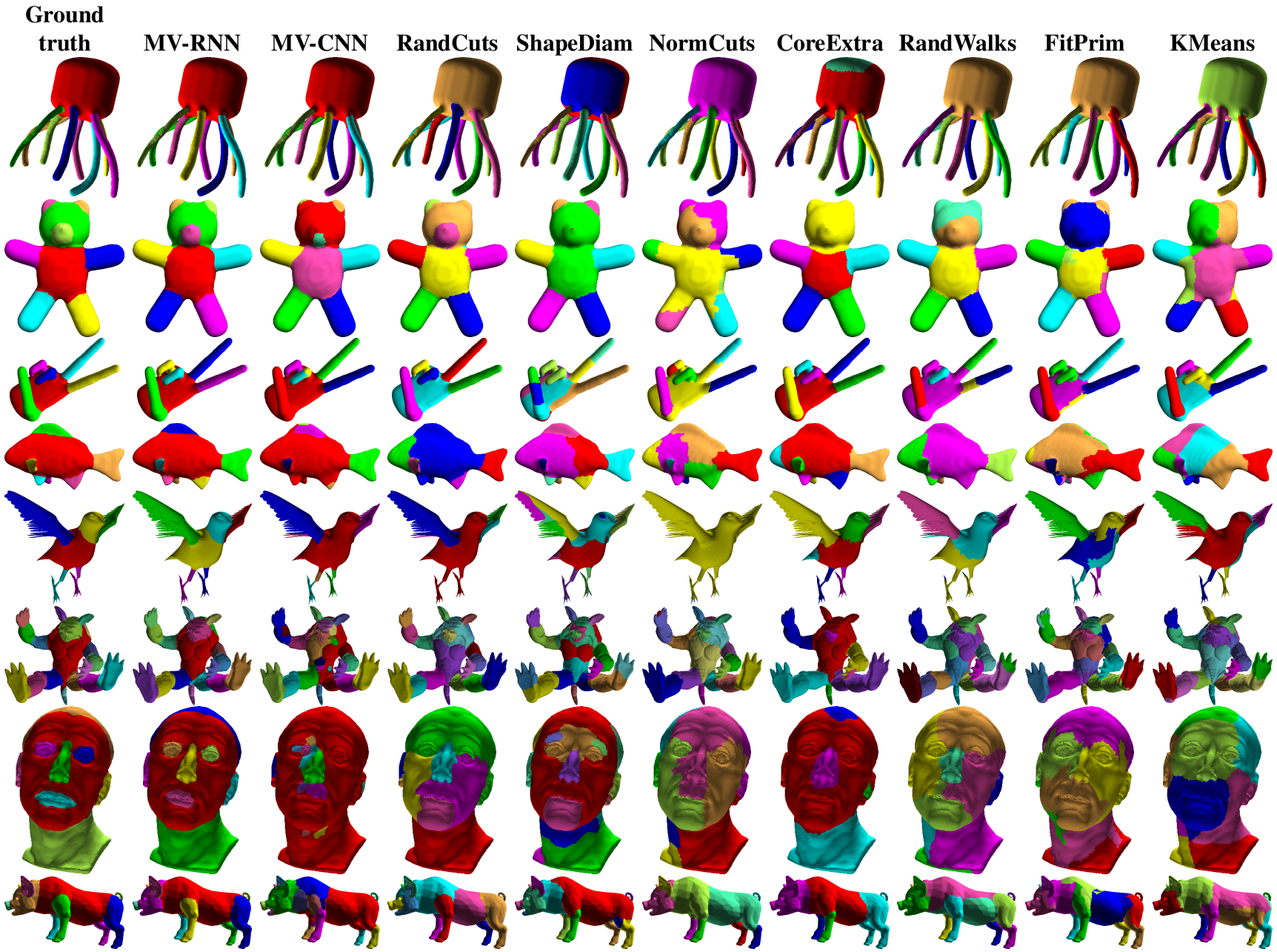


Figure 3.8: More comparisons of segmentation algorithms.

$$E_{\text{unary}}(f_v = l) = \begin{cases} 0,\ \forall l & \text{if } h_v = 0 \\ 0 & \text{if } h_v = l \\ \infty & \text{if otherwise} \end{cases} \tag{3.7b}$$

$$E_{\text{pairwise}}(f_u = l_u, f_v = l_v) = \begin{cases} e^{-d^2(u,v)} & \text{if } l_u \neq l_v \\ e^{-(1-d(u,v))^2} & \text{if } l_u = l_v \end{cases} \tag{3.7c}$$

where $d(u,v)$ is the geodesic distance [127, 128] between polygon u and polygon v. All distances are normalized to $[0,1]$.

The unary term tells that we only want to correct unlabeled polygons while the pairwise terms favor the same label for adjacent polygons. We use mean-field approximation [129] to solve (3.7a).

3.5 Evaluation

In this section, we present experimental validations and analyses of our approach. We test the segmentation algorithm on the well-known Princeton Segmentation Benchmark dataset [21]. This dataset has been intensively used to evaluate 3D shape segmentation and 3D shape retrieval algorithms. The dataset has 19 different object categories with 20 objects for each category which results in 380 models in total. For each category, we randomly select 16 models for training and 4 models for testing. Since there are multiple human generated segmentations for each model, we manually select one segmentation which is the most consistent among the object category. The ground truth edge images can be easily obtained by rendering the edges between different segments overlaid by the 3D shape with the same color as background. To further enhance the quality of the ground truth images, we use *polygon offset* in `OpenGL`. The ground truth edge images are used in training both the MV-CNN and the LSTM. Fig. 3.5 shows some representative segmentations of our

MV-RNN approach on this dataset.

To evaluate our segmentation method, we adopt four metrics that are defined by Chen et al. [21], including Rand Index, Cut Discrepancy, Hamming Distance and Consistency Error. Rand Index, named after William M. Rand, measures the similarity between two segmentations of the same shape. From a mathematical point of view, Rand Index is related to the accuracy, but is applicable even when class labels are not used. In this work, we use Rand Index Error, which equals to one minus the Rand Index. Cut Discrepancy is a boundary-based method evaluating the distance between different cuts. It sums the distances from points along the cuts in the computed segmentation to the closest cuts in the ground truth segmentation, and vice-versa. Hamming Distance, named after Richard Hamming, is a region-based method and measures the number of substitutions required to change one region into the other. Hamming Distance is directional, hence it includes missing rate (Rm) and false alarm (Rf) distances. Consistency Errors, whether the global version (GCE) or local version (LCE), are used to compute the hierarchical differences and similarities between segmentations, which are based on the theory that humans perceptual organization imposes a hierarchical tree structure on objects. Regarding all four metrics, smaller value indicates better result.

Comparison: We compare our method with the following segmentation algorithms:

- MV-CNN: we apply non-maximum suppression [16] on the boundary probability maps returned from the multi-view CNN (HED in this case) and unproject them back to 3D (without LSTM) followed by CRF. This serves as a baseline for multi-view paradigm.
- [Shu2016] [110]: an unsupervised 3D shape segmentation via stacked auto-encoders.
- WcSeg [126]; approximate convexity analysis.
- RandCuts [99]: randomized cuts.
- ShapeDiam [40]: shape diameter function.

- NormCuts [99]: normalized cuts.
- CoreExtra [130]: core extraction.
- RandWalks [52]: random walks.
- FitPrim [44]: fitting primitives.
- KMeans [131]: k-means.

Figs. 3.7 and 3.8 provide a side-by-side comparison of segmentations obtained from various algorithms. Although there are large shape variations, the absolute majority of our segmentation results are desirable and consistent with our perception. The baseline MV-CNN indeed yields better segmentations than some of the methods based on hand-crafted features such as k-means, fitting primitives, random walks. Due to the inconsistency of the boundary probability maps across multiple views, the MV-CNN is still not as good as the shape diameter function. However, the added LSTM has a significant contribution to the overall robustness, which vastly improves the nature of multi-view paradigm.

Numerical comparison: The Rand Index score statistics of our segmentation on the dataset, as well as those of other methods, are detailed in Table 3.1, from which we can see that our algorithm obtains an average Rand Index of 0.084 that outperforms the related algorithms. In addition to Rand Index, our MV-RNN also shines out of other methods with respect to other evaluation metrics (see Fig. 3.6 and Table 3.2). Comparing with the baseline MV-CNN, the LSTM in our framework indeed has a significant improvement because it correlates the outputs from CNN across multiple views.

Different number of views: We also experiment with various values of K. According to Fig. 3.9, using too few number of views is not good due to occlusion. As using more views equally distributed around the object, the object's surface area is more fully covered, hence we get higher accuracy (or lower Rand Index score). We choose $K = 60$ as a reasonable trade-off between accuracy and time/memory consumption.

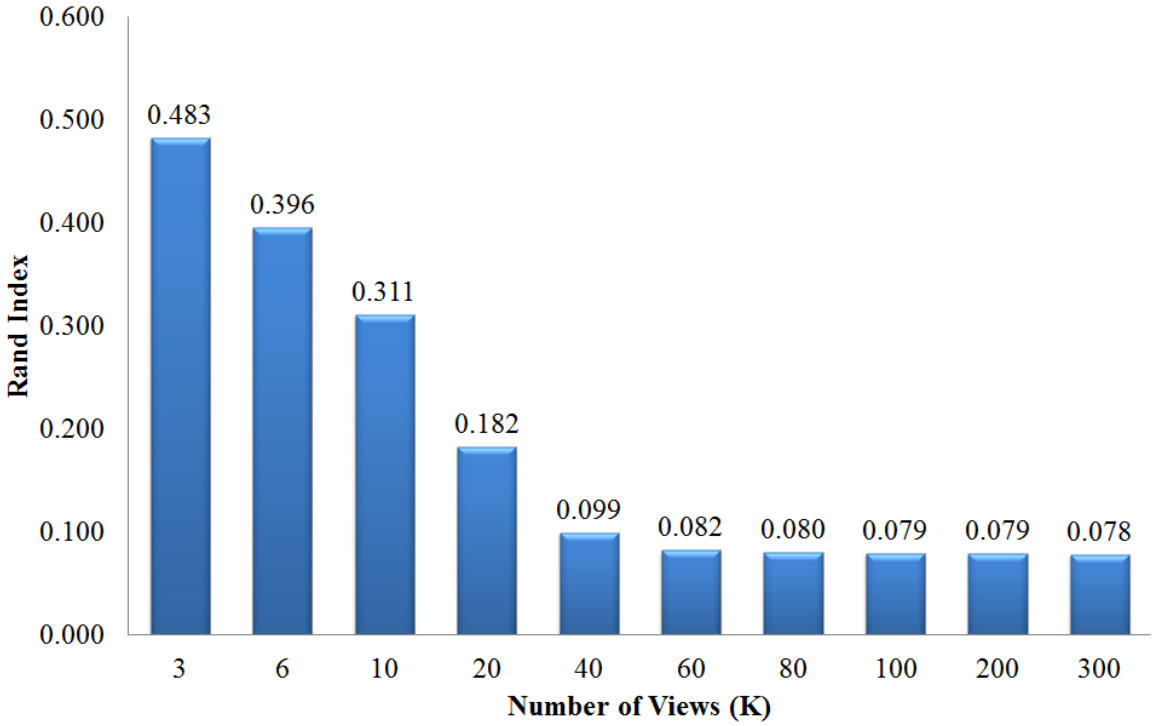


Figure 3.9: The Rand Index with respect to the number of views. We choose $K = 60$ as a reasonable trade-off between accuracy and time/memory usage.

3.5.1 Limitation

Because our approach belongs to the multi-view paradigm, it has a common occlusion issue. For example, the left and right thighs of the man in Fig. 3.10 are not separated due to occlusion (i.e. the area under torso is not revealed from any of $K = 60$ views). Increase the number of views could reduce the occlusions at the cost of more computations. Since we can easily computed occluded areas given the current set of views, we plan to use adaptive best view prediction to focus the camera on these areas, which is similar to the next-best-view prediction in 3D attention model proposed by Xu et al. [132].

3.6 Conclusion

We have presented our novel MV-RNN for 3D shape segmentation which combines the MV-CNN and LSTM to enhance the multi-view paradigm. To the best of our knowledge,

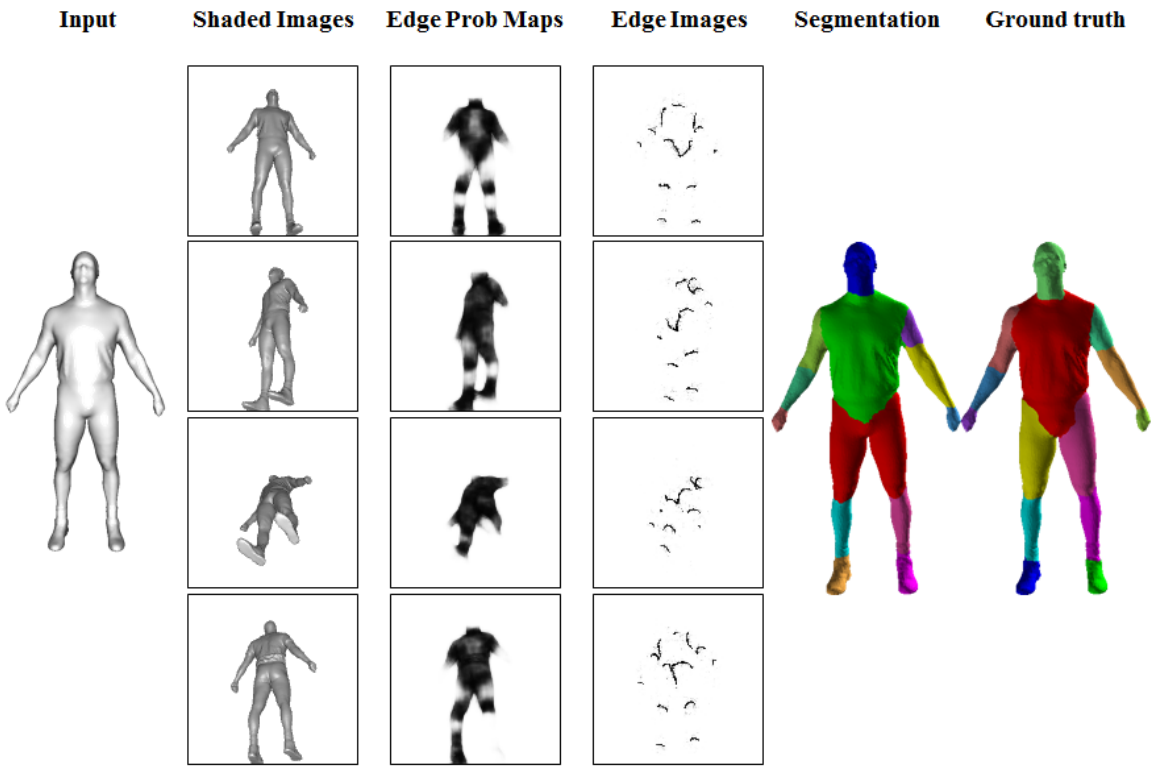


Figure 3.10: Limitation of our approach. The area under the torso is occluded and hence the left and right thighs are not separated although our MV-RNN can detect 2D edges correctly in all views.

we are the first group that treats multiple views as a temporal sequence and applies RNN to predict the edge images by aggregating the corresponding edge probability maps obtained by feed-forwarding a MV-CNN. Our MV-RNN detects 3D edges in an end-to-end manner and the segmentation is obtained as a post-processing. The 3D edges can be either semantic-based (e.g. semantic segmentation) or geometric-based (e.g. CAD model segmentation, suggestive contour, ridge and valley). According to our experimental results on the Princeton Segmentation Benchmark dataset, our MV-RNN compares favorably with other state-of-the-art methods on mesh segmentation.

In the future, we would like to conduct more experiments on different datasets such as those in [133, 117]. Additionally, our framework right now work on meshes only. In the future we would like to extend it to handle point clouds as well. The proposed framework is purely data-driven, thus in the future we would like to extend our method to other inter-

esting problems in shape modeling such as suggestive contours [92, 93] and ridge-valley detection [94].

Acknowledgment

We would like to acknowledge the authors of Princeton Segmentation Benchmark [21] who made the dataset public and provided evaluation toolbox. We also appreciate the authors of HED [109] for their edge detection network. Last but not least, we would like to thanks all the authors of other segmentation algorithms [110, 126, 99, 40, 130, 52, 44, 131] for their contribution of the segmentation results on the Princeton Segmentation Benchmark dataset.

Chapter 4

REDN: A Recursive Encoder-Decoder Network with Skip-Connections for Edge Detection

In this work, we introduce REDN: A Recursive Encoder-Decoder Network with Skip-Connections for edge detection in natural images. The proposed network is a novel integration of a Recursive Neural Network with an Encoder-Decoder architecture. The recursive network enables us to increase the network depth without increasing the number of parameters. Adding skip-connections between encoder and decoder helps the gradients reach all the layers of a network more easily and allows information related to finer details in the early stage of the encoder to be fully utilized in the decoder. Based on our extensive experiments on popular boundary detection datasets including BSDS500 [1], NYUD [2] and Pascal Context [3], REDN significantly advances the state-of-the-art on edge detection regarding standard evaluation metrics such as Optimal Dataset Scale (ODS) F-measure, Optimal Image Scale (OIS) F-measure, and Average Precision (AP).

4.1 Introduction

Edge detection has been a cornerstone and long-standing problem in computer vision since the early 1970's [134, 135, 136] and is essential for a variety of tasks such as object recognition [137, 138], segmentation [139, 1, 140, 141], etc. Initially considered as a low-level task, researchers now generally agree hat high-level visual context such as the perception of objects play an important role in edge detection [1].

Inspired by the success of deep convolutional neural networks (DCNN) in computer vision problems such as image classification [100, 101, 102], object detection [142], image segmentation [104, 107, 143, 105, 144, 106], normal estimation [145, 133], image captioning [114], etc., researchers have begun to utilize DCNN for low-level tasks such as edge detection [109, 146, 147, 148, 149, 150]. For example, Xie et al. [109] developed a HED network built upon the VGG-16 network [101] which hierarchically obtains edge images at multiple scales. Edges obtained from the initial levels are more localized while those from the deeper levels are more global. The final edge is a linear combination of all edge images at different scales. Later, Kokkinos et al. [140] explicitly applied HED [109] on the image pyramid. Yang et al. [146] developed a fully convolutional encoder-decoder network (CEDN) similar to Noh et al. [106]. The main drawback of these approaches is that the salient edges are obtained at the deeper layers with relatively lower resolution. Thus, the upsampled edge image tends to be blurry and less localized.

Several researchers also proposed the use of a refinement network for edge images at different hierarchies to achieve better edge detection results. For instance, Wang et al. [147] proposed a refinement module that fuses a top-down feature map from the backward pathway with the feature map from the current layer in the forward pathway, and further up-samples the map by a small factor of two, which is then passed down the pathway. Liu et al. [148] designed another type of refinement module, which uses all convolution layers at the same hierarchy to predict edge image at that level, to achieve a similar goal.

In this work, we propose a novel Recursive Encoder-Decoder Network with Skip Con-

nections (REDN) for edge detection in natural images. The proposed network is a novel integration of a Recursive Neural Network with an Encoder-Decoder architecture. Our encoder-decoder network is formed by DenseNet blocks [151], which are used to alleviate the vanishing-gradient problem, strengthen feature propagation and encourage feature reuse. The encoder network performs convolutions and poolings to produce a set of feature maps of different visual levels. The deeper the layer is, the higher level, more abstract, and less localized the features are. The encoder tends to learn more global and high-level features, and could ignore some finer information. The decoder network, which is topologically symmetric with the encoder network, first upsamples the feature maps by transposed convolutions (i.e. deconvolutions) followed by convolutions and finally returns the edge image with the same size as the input image. Nevertheless, since information related to finer details might be lost during the encoding stage, the decoded outputs are generally less detailed. As a result, edges generated by the encoder-dencoder network are usually blurry and less localized [146].

To overcome this limitation, in this work, we propose to add *skip-connections* [152] that connect one layer in the encoder to the corresponding layer in the decoder of the same level of hierarchy. Since features from the early encoder are forwarded to the later decoder, skip-connections provide sharper visual details. Skip-connections have been widely used in deep learning community such as U-Net [153], Deep Reflectane Map (DRM) [154], ResNet [103] and DenseNet [151]. According to [152, 151], skip-connections greatly improve gradient flow by allowing more even weight update in all of the layers.

We further enhance the network by adding a *feedback loop* between the output edge map and the input [155, 156]. The purpose of this is to enable iterative refinement of the edges using a single network model. Increasing recursion depth can improve performance without introducing new parameters for additional convolutions and deconvolutions. The whole network can be modeled jointly with shared parameters and optimized in an end-to-end manner.

Furthermore, in order to force the network to learn more salient edges, we propose a simple but very effective data augmentation scheme by conducting random Gaussian blur to the input images. This also helps to reduce potential over-fitting as the input images is augmented randomly in each iteration during training.

In summary, the main contribution of this work is to improve the deep learning algorithms used for edge detection by combining skip-connections and feedback loop into an encoder-decoder network, as well as a simple and effective Gaussian blurring based data augmentation. To the best of our knowledge, we are the first group applying the recursive network in low-level tasks such as edge detection. Our REDN experimentally demonstrates state-of-the-art results on popular boundary detection datasets including BSDS500 [1], NYUD [2] and Pascal Context [3].

4.2 Related Work

The literature of edge detection is very expansive. We will only be able to highlight a few representative works that are closely related to our work.

The early pioneering edge detection methods (e.g. [15, 14, 135, 157, 158, 13, 159, 160]) focused on low-level cues such as image intensity or color gradients. A complete overview of various low-level edge detectors can be found in [161, 162]. For example, the well-known Canny edge detector [13] finds the peak gradient orthogonal to edge direction. In general, these low-level edge detectors are not very robust and may generate many false positives or false negatives.

In the past decade, people have explored machine learning techniques for more accurate edge detection especially under more challenging conditions [163, 164, 165, 166, 167, 1, 168, 169]. For example, Dollar et al. [164] used a boosted classifier to independently label each pixel using its surrounding image patch as input. Zheng et al. [165] combined low, mid, and high-level cues to achieve improved results for object-specific edge detection.

Arbelaez et al. [1] combined multiple local cues into a global framework based on spectral clustering. Ren and Bo [170] further improved the method of [1] by computing gradients across learned sparse codes of patch gradients. Lim et al. [171] proposed an edge detection approach that classifies edge patches into sketch tokens using random forest classifiers. Sketch tokens are learned using supervised mid-level information in the form of hand drawn contours in images. Dollar et al. [172] learned more subtle variations in edge structure and lead to a more accurate and efficient algorithm. This structured edge detection method was considered one of the best method for edge detection thanks to its state-of-the-art performance and relatively fast speed.

Recently deep learning approaches become very popular and researchers have attempted to deploy it to edge detection. It is widely believed that accurate detection of edges requires object-level understanding of the image, an area in which deep learning is best known for. Kivinen et al. [173] applied mean-and-covariance restricted Boltzmann machine (mcRBM) architecture [174] to edge detection and obtained competitive results. Starting from candidate contour points from Canny edge detector [13], DeepEdge [175] extracts patches at four different scales and simultaneously run them through the five convolutional layers of the AlexNet [100]. These convolutional layers are connected to two separately-trained network branches. The first branch is trained for classification, while the second branch is trained as a regressor. At testing time, the scalar outputs from these two sub-networks are averaged to produce the final score. DeepContour [176] classified image patch of size 45×45 into background or one of the clustered shape classes by a 6-layer convolutional neural network. The disadvantage of both DeepEdge and DeepContour is that at testing time, it operates on the input image in a sliding window fashion (due to the fully-connected layers), which restricts the receptive size of the network to only a small image patch and thus may lose global information.

Inspired from FCN [105], Xie et al. [109, 177] proposed the HED network which can be trained in an end-to-end manner. An interesting idea of this work is that the final edge

map is fused from multiple edge maps obtained at different scales. The multi-scale edge maps are side outputs of a VGG-16 network [101] and hence the shallower edge maps give finer detail edges while the deeper ones capture the more salient edges. The final result is linearly combined from all edge maps at multiple scales. The main drawback of this network is that salient edges are typically learned in the deeper layers, hence they are of low-quality when being up-sampled - edges are blurry and do not stick to actual image boundaries. Later, Kokkinos [140] proposed the Deep-Boundaries network, which is essentially a multi-scale HED [109]. As being claimed by Kokkinos [140], the explicit use of multiple scale improves the accuracy of edge detection. However, because being built upon the HED [109] and fed by down-sampled images, Deep-Boundaries also suffers from the same issue as the HED.

To solve the issue of low quality salient edges, Wang et al. [147] and Liu et al. [148] proposed the CED and RCF, respectively. Both papers proposed an extra network to synthesize the high resolution edge maps from low resolution ones instead of trivially using bilinear interpolation. For example, CED's refinement module fuses a top-down feature map from the backward pathway with the feature map from current layer in the forward pathway, and further up-samples the map by a small factor ($2\times$), which is then passed down the pathway.

Maninis et al. [149] proposed the Convolutional Oriented Boundaries (COB) which demonstrated state-of-the-art performance in edge detection. From a single pass of a base convolutional neural network, COB obtains multiscale oriented contours, combines them to build Ultrametric Contour Maps at different scales and finally fuses them into a single hierarchical segmentation structure.

Our REDN architecture is based on an encoder-decoder network with significant improvements. Firstly, we use DenseNet blocks within each convolution group. Secondly, we add skip-connections between encoder and decoder, which helps the gradient to more easily reach all the deep layers of a network. Additionally, finer details in the early stage

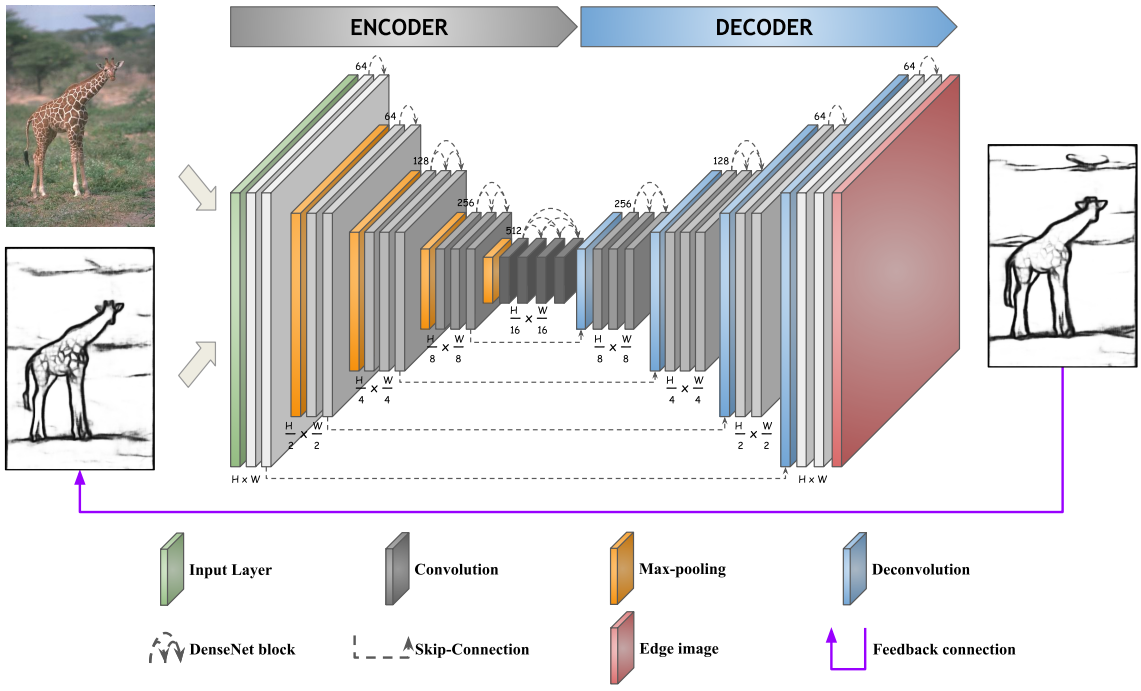


Figure 4.1: The architecture of our Recursive Encoder-Decoder Network with Skip Connections (REDN). Encoder-decoder network is at the heart of our design which consists of DenseNet blocks. There are four skip-connections that connects one layer of the encoder to a corresponding layer of the decoder. The feedback connection enables a deeper network with no extra parameters.

of the encoder are preserved to be used in the decoder. Thirdly, our recursive network is used with convolutions to further increase the network depth with the same number of parameters. In the next section, we will describe our network architecture in depth followed by evaluation results.

4.3 Recursive Encoder–Decoder Network with Skip-Connections

Fig. 4.1 shows the architecture of our REDN. Our network takes as input an RGB image and a recursive edge image, concatenates them (in the depth channel) and passes through an encoder-decoder network. The encoder consists of 5 blocks of DenseNet [151]. The decoder is symmetric with the encoder with max-pooling replaced by transposed convolution (i.e. deconvolution). Skip-connections connects corresponding layers of encoder and

decoder at the same hierarchy. The decoder outputs an edge image of the same resolution as the input image, which serves as a recursive input to replace the edge image in the network (feedback loop). There are L iterations and $L = 0$ indicates no feedback loop at all. In contrast to DeepEdge [175] and DeepContour [176] which can only be applied on image patch of fixed size due to the use of fully-connected layers, our REDN does not contain any fully-connected layer, and can consume images of any size. In the following sections, we elaborate the REDN in more details and discuss the training and testing procedures.

4.3.1 Training Formulation

We denote our input training dataset by $S = \{(X_i, Y_i)\}_{i=1}^{N}$ where X_i denotes raw input image patch (we use patch size of 256×256 during all experiments) and Y_i denotes the corresponding binary ground truth edge map for image patch X_i. The goal of the network is to produce edge maps approaching the ground truth. Let $\mathbf{W}$ be the collection of all network parameters for simplicity. The network runs through L iterations, each of which produces an edge map $f^{(l)}(X_i|\mathbf{W})$ ($l = 0, \ldots, L$). Thus, $f^{(L)}(X_i|\mathbf{W})$ is the final output of the REDN. Consequently, the ultimate goal is to minimize the loss between the final edge map and the ground truth, or

$$\min_{\mathbf{W}} \quad \mathscr{L}\left(f^{(L)}(X_i|\mathbf{W}), Y_i\right) \tag{4.1}$$

where $\mathscr{L}$ is the loss function, a weighted cross-entropy that will be discussed later.

Nevertheless, training such a deep network is not trivial when $L \geq 1$. Adapting the idea of deeply supervised network training [109, 177], we also regularize the network by adding multiple losses for all $f^{(l)}(X_i|\mathbf{W})$. The goal now is to minimize the following.

$$\min_{\mathbf{W}} \quad \sum_{l=0}^{L} \alpha_l \mathscr{L}\left(f^{(l)}(X_i|\mathbf{W}), Y_i\right) \tag{4.2}$$

where $\{\alpha_l\}_{l=0}^{L}$ are weights for edge maps at each iteration. We set $\alpha_l = l + 1$ to force the

Figure 4.2: Original images (top row) are augmented with Gaussian noise to force the network to extract stronger edges.

network to focus on the edge maps at later iterations.

4.3.2 Testing Formulation

During testing, given image X, we obtain the edge map predictions at all iterations of REDN, i.e. $f^{(l)}(X|\mathbf{W}), l = 0,\ldots,L$. The final edge map is defined as the last one.

$$\hat{Y}_{\text{RED}} = f^{(L)}(X_i|\mathbf{W}) \tag{4.3}$$

Alternatively, one may define the final edge map as a weighted combination of all edge maps with learnable weights γ as follows.

$$\hat{Y}_{\text{REDN}} = \frac{\sum_{l=0}^{L} \gamma_l \mathscr{L}\left(f^{(l)}(X_i|\mathbf{W}), Y_i\right)}{\sum_{l=0}^{L} \gamma_l} \tag{4.4}$$

Empirically, when the network is trained properly, we do not notice significant difference between these two formulations (4.3) and (4.4) both visually and quantitatively. Therefore, we opt to use (4.3) for simplicity.

4.3.3 Network Architecture

Encoder

The encoder extracts features from input image, so we need an architecture that is deep and can efficiently generate perceptually multi-level features. Inspired from the recent success of DenseNet [151] on image classification, we design our encoder by stacking 5 DenseNet blocks. The first block consists of two 5×5 convolution layers with 64 kernels for each, followed by a similar second block with max-pooling layer in between which downsamples the feature maps and hence forces the network to learn good global features. Starting from the third block, we double the number of kernels for each successive block, which results in a 512-dimension feature maps after the fifth block. Moreover, we also increase the number of convolution layers to 3, 3 and 4 for the third, fourth and fifth blocks, respectively for more powerful architecture. Every convolution layer in the encoder composes of a convolution layer, a batch normalization layer [178] and a leaky rectified unit activation [179] (with leaking coefficient of 0.1) in this order.

Decoder

The decoder maps the learned features to another space and eventually reaches the edge image. This network is symmetric with the encoder with 5 DenseNet blocks. We use transposed convolutions (or deconvolutions) to upsample the feature maps corresponding to max-pooling in the encoder. The transposed convolutions are initialized as bilinear filters which purely serve as upsample filters. At the last layer, the decoder returns an edge prediction from the 64-channel layer via convolution. To facilitate the training, we also use batch normalization and leaky rectified unit in the same way as in the encoder except for the last layer which only consists of a convolution followed by a sigmoid activation.

Skip-connections

The encoder progressively extracts and down-samples features, while the decoder upsamples and combines them to construct the output. The sizes of feature maps are exactly mirrored in our network. We concatenate early encoded features (from the encoder) to the corresponding decoded features (from the decoder) at the same spatial resolution, in order to obtain local sharp details preserved in early encoder layers. There are four of such skip-connections corresponding to four different level of hierarchies, which are called mirror-links. Mirror-link is a form of skip connection which has been proven effective in many deep network such as ResNet [103] and DenseNet [151]. Besides the sharpness, these skip-connections could also regulate gradient flow and allow better trained network.

Feedback loop

This is a recursive connection similar to the Recurrent Neural Network (RNN). In contrast to RNN in which recurrence targets temporal sequence and tries to learn temporal changes, our feedback loop refines the edge map progressively without introducing more network parameters. At the beginning, there is no edge image generated, so the initial edge image is set as a blank image (i.e. zero-image). After the first pass through the encoder-decoder network, the output edge map is recursively fed back to the input and repeatedly processed through the shared encoder-decoder network. The whole REDN is jointly optimized in an end-to-end manner. Due to memory limit, we only conduct experiments for $L = 2$.

4.3.4 Loss function

We use weighted sigmoid cross-entropy function to compute the loss between our predicted edge $\hat{Y}_{\text{REDN}}$ (or other intermediate edge images $f^{(l)}(X_i|\mathbf{W})$, $l = 0,\ldots,L$) and the ground

truth edge image Y as follows.

$$\mathscr{L}\left(\hat{Y}_{\text{REDN}}, Y_i\right) = -(1-\beta)\sum_{j\in Y_+}\log\hat{Y}_{\text{REDN}} - \beta\sum_{jinY_-}\log\left(1-\hat{Y}_{\text{REDN}}\right) \tag{4.5}$$

where Y_+ and Y_- denote edge and non-edge pixels, respectively and $\beta = \frac{|Y_+|}{|Y|}$ to balance the relative importance of these two classes.

4.3.5 Implementation

We implement our framework using the publicly available TensorFlow [180].

Hyper-parameters

In contrast to fine-tuning CNN for image classification, adapting CNN for pixel-wise output requires special care. Even with the proper initialization or a pre-trained model, sparse ground truth distributions coupled with conventional loss functions lead to difficulties in network convergence. Through experimentation, we choose the following hyper-parameters: mini-batch size of 8, convolutional filters randomly initialized by Gaussian distribution with zero-mean and standard deviation of 0.01, convolutional biases all zero-initialized, deconvolutions initialized as bilinear filters, weight decay of 10^{-6}, training epochs equal 500. Furthermore, we use Adam optimizer [125] with initial learning rate 10^{-4}. As mention earlier, we extract image patches of size 256×256 for training but use the whole image during testing.

Data augmentation

Data augmentation has proven to be a crucial technique in training deep neural networks. For each training image, we randomly sample 500 patches, each of size 256×256, which is a kind of image cropping. We further randomly flip the training image horizontally.

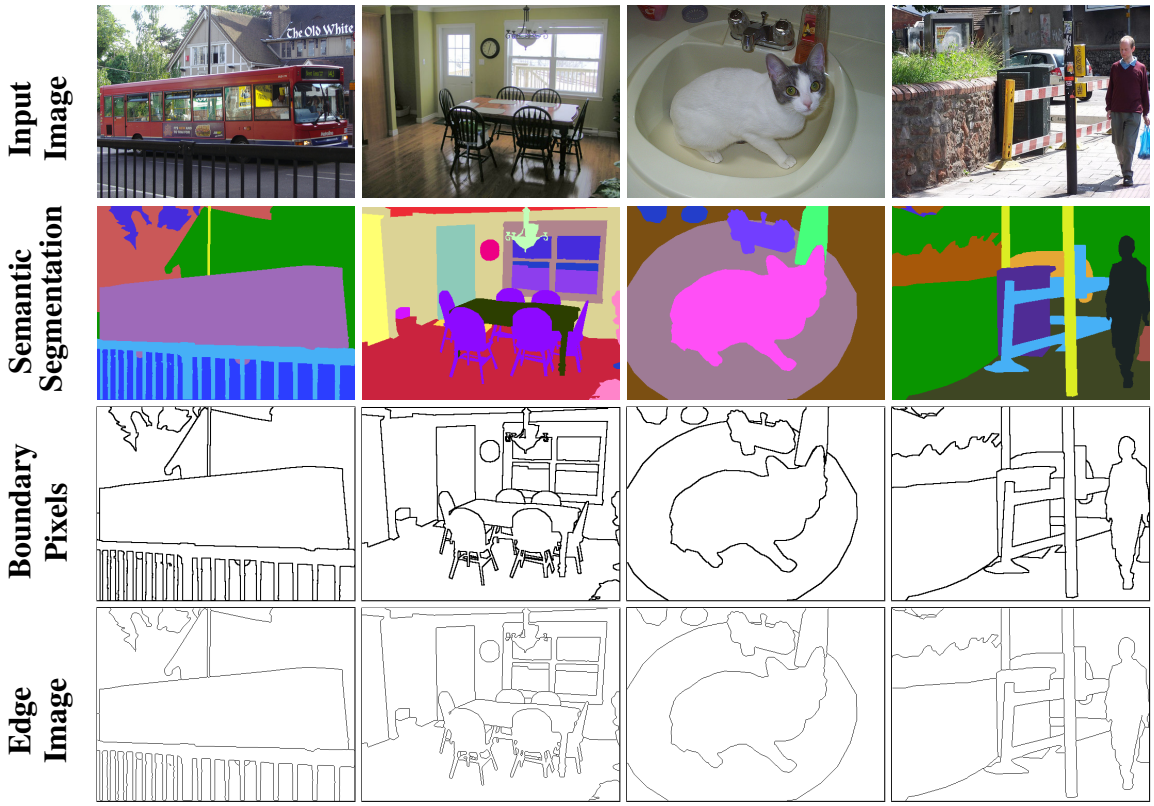


Figure 4.3: Ground truth edge image generation. From an input image (first row) with ground truth semantic segmentation (second row), we identify all boundary pixels (third row) and then apply image thinning (e.g. `MATLAB`'s `bwmorph`) to obtain the ground truth edge image (last row).

These together lead to an augmented training set that is a factor of 500 times larger than the unaugmented set.

In addition, we add random Gaussian noise (black-and-white noise) to the training images by sampling from a Gaussian distribution with zero-mean and standard deviation of 20 (assuming image intensities are within $[0, 255]$) (Fig. 4.2). This data augmentation forces the network to learn the stronger edges such as object contours over finer texture ones. This augmentation also helps combat over-fitting because each training image is augmented differently in each iteration.

Table 4.1: Datasets and Parameters

Dataset	# train	# test	`maxDist`
BSDS500 [1]	300	200	0.0075
NYUD-v2 [2]	795	654	0.011
Pascal Context [3]	7605	2498	0.0075

Running time

Training ranges from 4 hours for the BSDS500 dataset with 300 images to 50 hours for the Pascal Context dataset with 7605 images on a single Titan-X GPU. REDN produces an edge response for an image of size 512×512 in about 270 milliseconds including interface overhead (e.g. image loading), which is approximately 3.4 frames/second. This is significantly more efficient then existing CNNs such as DeepEdge [175], DeepContour [176] and COB [149].

4.4 Evaluation

This section presents the performance of our REDN on the well-known datasets for edge detection such as BSDS500 [1], NYUD [2] and Pascal Context [3] (see Table 4.1). We adopt three standard evaluation metrics commonly used for edge detection, fixed contour threshold ODS F-score, per-image best threshold OIS F-score, and average precision AP [1]. We compare our method against popular state-of-the-art methods including both the non-deep learning and deep learning approaches. For a fair quantitative comparison, we apply a standard non-maximal suppression technique [172] to all edge maps generated by all methods to obtain thinned edges before evaluation.

4.4.1 Datasets

We evaluate our algorithm on BSDS500 [1], NYUD [2] and Pascal Context [3] datasets using standard metrics such as ODS/OIS F-measure and AP. The BSDS500 dataset has

Figure 4.4: Side-by-side comparison of edge detection algorithms. All edge images are originally returned by the algorithms before non-maximum suppression.

Table 4.2: BSDS500 [1] test evaluation

Method	ODS	OIS	AP
Canny [13]	0.600	0.640	0.580
MShift [181]	0.601	0.644	0.493
EGB [182]	0.610	0.640	0.560
ISCRA [183]	0.724	0.752	0.783
gPb-owt-ucm [1]	0.726	0.757	0.696
Sketch Tokens [171]	0.727	0.746	0.780
SCG [170]	0.739	0.758	0.773
SE [172]	0.746	0.767	0.803
OEF [184]	0.749	0.772	0.817
MCG [185]	0.747	0.779	0.759
LEP [186]	0.757	0.793	0.828
DeepNets [173]	0.738	0.759	0.758
N4-Fields [187]	0.753	0.769	0.784
DeepEdge [175]	0.753	0.772	0.807
CSCNN [188]	0.756	0.775	0.798
DeepContour [176]	0.756	0.773	0.797
HED-fusion [177]	0.782	0.804	0.833
HED-late-merging [177]	0.788	0.808	0.840
CEDN [146]	0.788	0.804	0.834
COB [149]	0.793	0.820	0.859
CED [147]	0.803	0.820	0.871
RCF [148]	0.806	0.823	–
REDN (ours)	0.808	0.828	0.827

edge annotation ground truth while the others do not. The NYUD and Pascal Context datasets are primarily for semantic segmentation. To obtain the ground truth edges, we first identify all the boundary pixels, treat them as a binary image and then apply image thinning using `MATLAB` function `bwmorph` (see examples in Fig. 4.3).

BSDS500

The Berkeley Segmentation Dataset and Benchmark (BSDS500) [1] consists of 200 training, 100 validation and 200 testing images. We use the training and validation sets (300 images) for training our REDN. Each colored image is of size 481×321 or 321×481 and is manually annotated ground truth contours. We simply overlay all annotations followed

Table 4.3: NYUD-v2 [2] test evaluation

Method	ODS	OIS	AP
gPb-owt-ucm [1]	0.726	0.757	0.696
Silberman et al. [2]	0.658	0.661	n/a
SE [172]	0.685	0.699	0.679
MCG-B [185]	0.652	0.681	0.613
HED-RGB [177]	0.720	0.734	0.734
HED-HHA [177]	0.682	0.695	0.702
HED-RGB-HHA [177]	0.746	0.761	0.786
ResNet50-RGB-HHA [149]	0.745	0.762	0.792
ResNet50-RGB [149]	0.746	0.761	0.789
ResNet50-RGBD [149]	0.683	0.699	0.681
RCF-RGB [148]	0.729	0.742	–
RCF-RGB-HHA [149]	0.757	0.771	–
COB-PC [149]	0.710	0.735	0.734
COB-RGB [149]	0.778	0.799	0.814
COB-RGB-HHA [149]	0.784	0.805	0.825
REDN (ours)	0.793	0.813	0.832

Table 4.4: Pascal Context [3] test evaluation

Method	ODS	OIS	AP
SE [172]	0.533	0.568	0.496
LEP-B [186]	0.570	0.636	0.547
MCG-B [185]	0.554	0.609	0.528
HED [109]	0.688	0.707	0.704
CEDN [146]	0.702	0.718	0.744
COB [149]	0.750	0.781	0.773
REDN ($L = 0$) (no data aug.)	0.744	0.769	0.771
REDN ($L = 2$) (no data aug.)	0.759	0.784	0.784
REDN ($L = 2$) (with data aug.)	0.761	0.785	0.787

Table 4.5: Cross-dataset performance of REDN

Train	Test	ODS	OIS	AP
Pascal Context	BSDS500	0.755	0.781	0.828
	NYUD	0.732	0.766	0.783
BSDS500	Pascal Context	0.643	0.653	0.681
	NYUD	0.627	0.653	0.703

by image thinning to obtain a single ground truth image. Unlike other image-to-image deep learning framework such as HED [109] which resizes the input image to a fixed size

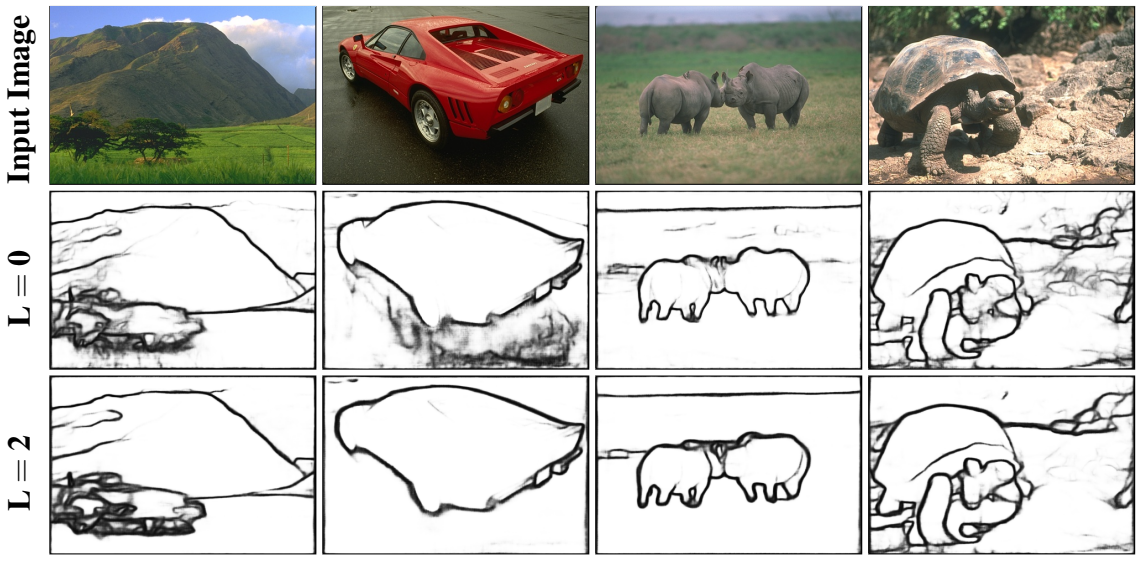


Figure 4.5: The recursive network ($L = 2$) improves the results of encoder-decoder network with skip-connections ($L = 0$) by cleaning noisy edges and enhance stronger ones.

of 400×400, our REDN runs on original image without resizing. We use padding to make image dimension fit after convolutional, pooling and deconvolutional layers and crop the output to get the result of the original dimension.

NYUD

The NYUD dataset [2], was used for edge detection in [170, 122], has 1449 RGB-D images of indoor scenes (which are quite different from outdoor scenes of the BSDS500 [1]). As a result, it is more challenging because the edges are more cluttered and there are more variations. Here we use the setting described in [172] and evaluate our REDN on data processed by [122]. The NYUD dataset is split into 795 training and 654 testing images. These splits are carefully selected such that images from the same scene are only in one of these sets. All images are of size 640×480. This dataset also has depth image and although our REDN is easily extensible to RGB-D image, we do not use this information for our experiment. HED [177] has three networks accepting RGB, depth encoded HHA [122] and RGB-HHA, respectively. Consequently, we include the results of all these three network versions in Table 4.3. For a fair comparison, during evaluation we increase the

maximum tolerance allowed for correct matches of edge predictions to ground truth from 0.0075 to 0.011 as used in [122, 172, 177].

Pascal Context

The Pascal Context dataset [3] contains carefully localized pixel-wise semantic annotations for the entire image on the PASCAL VOC 2010 detection trainval set. It contains 10,103 images, which is approximately 20 times larger than the BSDS500 dataset, span over 459 semantic categories. Images in this dataset have various sizes and are quite challenging due to the increased scene complexities.

4.4.2 Visual Comparison

Fig. 4.4 shows side-by-side comparison between different boundary detection algorithms. As we can see, the non-deep learning methods such as SE [172] and gPb-owt-ucm [1] produce sharp and clean edges in areas with high-contrast but fail at low-contrast regions because they only use local features and thus do not have a object-level understanding.

The HED [109], which uses the features from VGG-16 network [101], performs much better and is able to capture objects even in low-contrast cases and it is not easily confused by object's interior boundary. Its weakness remains in the blurry and less localized edge responses, which may prevent it from recovering the sharp details.

Our REDNs results are generally cleaner, sharper and more accurate. Additionally, our results capture more global boundaries. For example, in the airplane image (second to last row of Fig. 4.4), only the most salient edges of the plane are retained.

Fig. 4.5 illustrates the benefits of the feedback loop in our network. In a pure encoder-decoder network without feedback loop (i.e. $L = 0$), the results are blurry in the fine texture regions. However, with recursive network, these errors are cleaned up and salient edges are enhanced.

4.4.3 Quantitative Comparison

For numerical comparison, Table 4.2 shows the F-measure of various edge detection algorithms on BSDS500 dataset. It is obvious that our REDN is better than other methods regarding ODS/OIS F-measure with ODS $= 0.808$, OIS $= 0.828$ while providing reasonable AP $= 0.827$. Table 4.3 provides the numerical statistics of the tested algorithms on the NYUD dataset. As we can see, although our REDN only takes as input RGB image and ignores depth information, it is still better than HED [177], RCF [148] and COB [149] network which relies on both RGB and depth information. We set a new state-of-the-art edge detection on the NYUD dataset at ODS $= 0.793$, OIS $= 0.818$ and AP $= 0.832$.

The Pascal Context dataset is significantly larger and more challenging than the first two. From Table 4.4, without recursive network (i.e. $L = 0$), our REDN will be similar to CEDN [146], except the skip-connections. As we can see from the table, skip-connections boost the ODS F-measure from 0.702 to 0.744, which is a huge improvement from CEDN even though it is still marginally behind COB [149]. However, with feedback loop (i.e. $L = 2$), REDN edges out COB to achieve ODS F-measure of 0.759. Furthermore, with our novel data augmentation of adding random Gaussian noise, REDN manages to push the results a little further at ODS $= 0.761$, OIS $= 0.785$ and AP $= 0.787$.

4.4.4 Cross-dataset Evaluation

To further demonstrate the generalization capability of our network, we train our model with one dataset and test it with another dataset. Table 4.5 shows the performance of our method on BSDS500, NYUD and Pascal Context datasets. The performance of a pretrained model is expected to be lower than that of a fine-tuned one. Our pretrained model yields a high precision but a low recall due to its object-selective nature between any two datasets. Furthermore, since BSDS500 dataset is pretty small and less diversified than the other two, model trained on it results in bigger drops in performance when tested on NYUD and Pascal

Context.

4.5 Conclusion

We have proposed a method to substantially improve deep learning-based boundary detection performance. Our REDN adds skip-connections into the encoder-decoder network, which sharpens and preserves more details at the later layers, and a feedback loop, which allows progressive improvement of the edge image. We propose a novel data augmentation scheme use Gaussian blurring that can force the network to learn more salient edges as well as reduce the potential over-fitting. Our system is fully end-to-end trainable and operates in approximately 3.4 frames per second, a speed of practical relevance. As measured on the standard datasets such as BSDS500, NYUD and Pascal Context, our REDN significantly outperforms other state-of-the-art approaches and sets new records in all three evaluation metrics.

Acknowledgment

We would like to thank the authors of BSDS500 dataset [1], NYUD dataset [2] and Pascal Context dataset [3] for providing the benchmark data as well as evaluation toolbox. Additionally, we really appreciate the authors of all edge detection algorithms [13, 182, 1, 171, 170, 172, 184, 173, 187, 175, 188, 176, 177, 140] for providing source code and/or benchmark results for comparison.

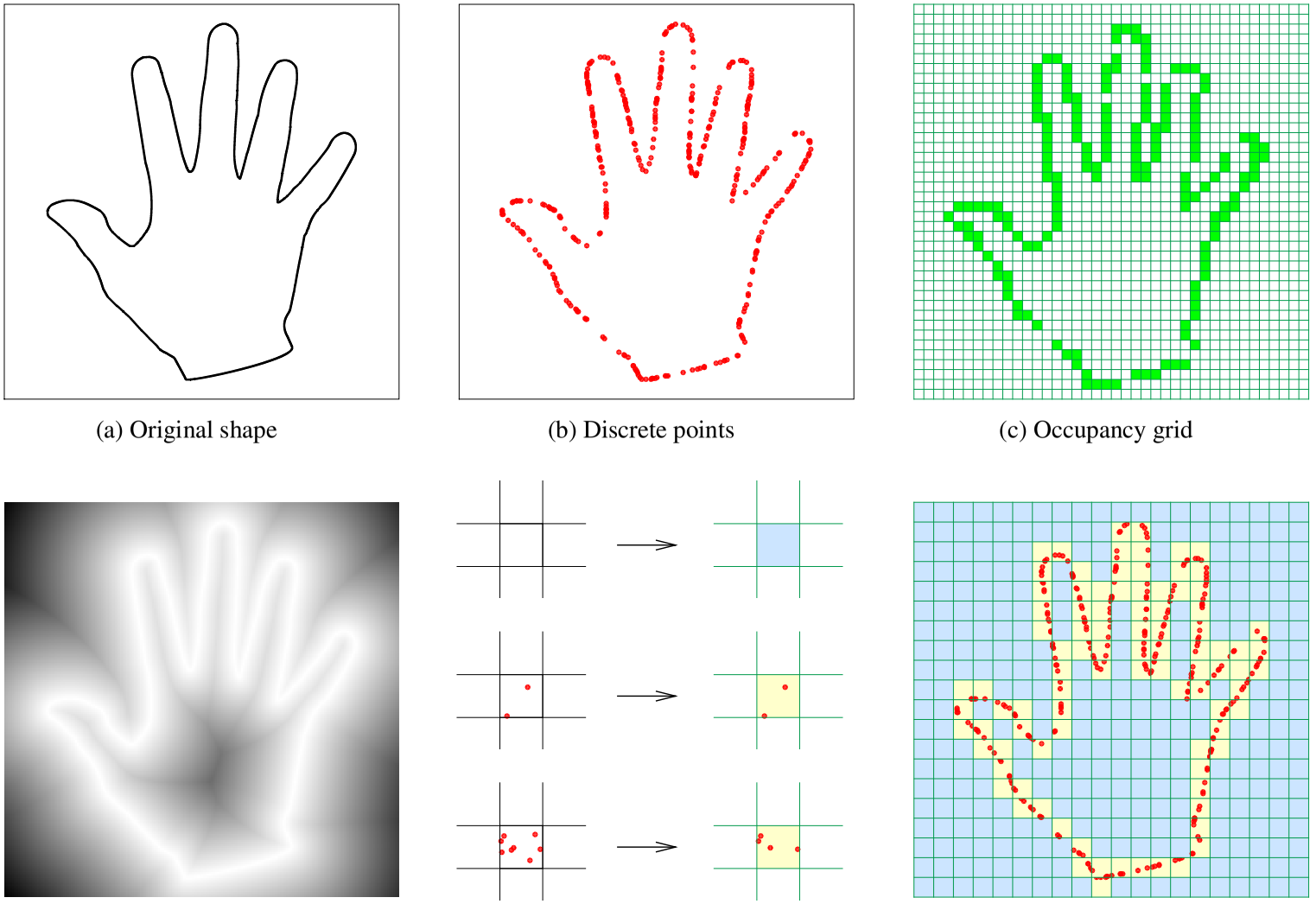
Chapter 5

PointGrid: A Deep Network for 3D Shape Understanding

Volumetric grid is widely used for 3D deep learning due to its regularity. However the use of relatively lower order local approximation functions such as piece-wise constant function (occupancy grid) or piece-wise linear function (distance field) to approximate 3D shape means that it needs a very high-resolution grid to represent finer geometry details, which could be memory and computationally inefficient. In this work, we propose the PointGrid, a 3D convolutional network that incorporates a constant number of points within each grid cell thus allowing the network to learn higher order local approximation functions that could better represent the local geometry shape details. With experiments on popular shape recognition benchmarks, PointGrid demonstrates state-of-the-art performance over existing deep learning methods on both classification and segmentation.

5.1 Introduction

Deep learning has become a universal tool for many visual recognition tasks ranging from classification to segmentation, especially ConvNets for 2D images [100, 101, 102, 103,



(a) Original shape (b) Discrete points (c) Occupancy grid

(d) Distance field (e) Point quantization ($K = 4$) (f) PointGrid ($K = 4$)

Figure 5.1: A 2D illustration of the proposed PointGrid, which is a hybrid 3D shape representation between discrete points (b) and volumetric grid (c) and (d). Points within each grid cell will be quantitized (e), so that both occupied (yellow) and empty (blue) cells have exactly K points (f).

104, 105, 106, 107, 108] thanks to its weight sharing and other kernel optimizations of 2D convolutions. It is therefore natural that a lot of researchers currently aim at the adaptation of deep ConvNets to 3D models. Such adaptation is, however, non-trivial due to the nature of 3D data representations. Currently the 3D geometry shape representation consists of point, mesh and volumetric grid. Mesh is extremely irregular and hence it is very hard to design a framework to directly learn from it. Point is flexible but it is unorganized. Volumetric grid is regular, which enables many researchers to utilize either occupancy grid or distance field as a mean of data representation and learn 3D convolutional networks from it.

Belonging to the volumetric grid, VoxNet and its variants [4, 189, 190, 191, 97, 192] is the most straightforward approach which transforms a 3D model into an occupancy grid. However, naive implementation of VoxNet does not scale well for dense 3D data because computational and memory requirements grow cubicly with the 3D grid resolution. A typical VoxNet takes as input a grid of size $64 \times 64 \times 64$ which is incapable of exploiting the rich and detailed geometry of the original 3D data. To resolve these issues, Kd-Net [193], O-CNN [194] and Oct-Net [195] in many respects mimic ConvNets but use the kd-tree or oct-tree structure to form the computational graph and apply 3D convolutions level by level, to share the learnable parameters, and to compute a sequence of hierarchical representations in a feedforward bottom-up fashion. These approaches exploit the sparsity of 3D data and can adaptively allocate computational and memory resources with respect to the data density. However, due to the use of more complicated data structures, it is generally not a simple task to implement these networks efficiently.

Recently Qi et al. [116] proposed PointNet that can consume unorganized point sets in 3D. In this network, all 3D points share the same set of multi-layer perceptrons which independently transform individual points. However, a single max-pooling layer is the only global operation in PointNet, which limits its ability to examine contextual neighborhood structure of the points.

In this work, we propose the *PointGrid*, a 3D convolutional network that is an integration of point and grid, a hybrid model that can better represent the local geometry shape details (Fig. 5.1). The proposed method scales better than volumetric grid and avoid information loss at the same time. PointGrid has an embedding volumetric grid that has the regular structure which allows 3D convolutions to extract global information hierarchically. In each grid cell, we sample a constant number of points (e.g. K) to overcome the grid size limitation. We expect the sampling points within the grid cell can better represent the local geometry shape details while the grid scales well with respect to data size as it only scales linearly in K, not cubicly as in pure volumetric grid. Later, we also show that PointGrid does not require a high resolution grid to perform well and a grid of $16 \times 16 \times 16$ is experimentally sufficient, which is substantially smaller than a typical $64 \times 64 \times 64$ grid of VoxNet. As a result, PointGrid (see Fig. 5.2) is simpler and faster in both training and testing. By experiments, PointGrid compared favorably with state-of-art methods including PointNet [116], PointNet++ [196], Kd-Net [193], O-CNN [194] and Oct-Net [195], with a smaller memory footprint.

5.2 Related Work

This section briefly goes over some of the existing approaches for 3D classification and segmentation, which can generally be categorized into two classes: learning on hand-crafted features or deep learning.

Hand-crafted features: The traditional approaches typically first extract local features such as planarity of various forms, higher degree geometric proxies (cylinders, cones, spheres, etc.), dihedral angles between triangles [38], curvatures (Gaussian curvature or mean curvature) [39], geodesic distances on a mesh [127, 128], slippage [49], symmetry, convexity, medial axis [197], shape diameter [40] and motion characteristics [20], shape contexts [198], spin images [199], etc. After that, people either directly apply machine

learning approaches on these features (e.g. k-NN, random forest, SVM [33], JointBoost classifier [200], correspondence analysis [201, 6]) or employ some local and greedy methods such as region growing [41, 42], hierarchical clustering [44, 45, 46], spectral clustering [48], k-means [47], normalized cut [99], random walk [52] and heat walk [53]. Shamir et al. [20], Agathos et al. [36] and Theologou et al. [37] gave a comprehensive overview of methodologies in 3D segmentation. In general, these approaches are often based on certain prior assumptions of some particular property of the 3D object and hence may not generalize well.

Deep learning: While deep learning has been very popular in 2D images for many years, it has just been applied in 3D recently because unlike pixels in 2D images, 3D objects do not have regular structure. As a result, in the early period, people use deep learning as a tool to learn high level features from low level cues (usually hand-crafted). The unsupervised shape segmentation proposed by Shu et al. [110] starts by over-segmenting the input model, computing patch-based local features and then uses stacked auto-encoder to learn high level features followed by Graph-Cut based segmentation. Guo et al. [111] compute local features at different scales for each triangle and arrange them into a rectangular image, which is fed forward through a convolutional neural network (CNN) to predict the semantic label for each triangle. Although these two frameworks use deep learning techniques (stacked auto-encoder, CNN) to learn high level features from local low level ones, they do not exploit the full potential of deep learning.

Recently researchers have started to either transform 3D data into regular form or refined convolution operations to adapt to the 3D's irregularity. Voxelization and multi-view are the most common representatives of the former approach. VoxNet and its variants [4, 189, 190, 191, 97, 192] discretizes the 3D bounding box into 3D occupancy grid, then applies 3D convolutions in a similar way as in 2D images. Among these approaches, VRN [191] uses Voxception-ResNet which mimics the Inception [102] and ResNet [103] and achieves state-of-the-art results for 3D object classifcation. The main drawback of the vol-

umetric approach is the information loss due to the voxelization as well as memory and computation consumptions as they increases cubicly with respect to the voxel's resolution. Kd-Net [193] and Octree-Net [194, 195] are designed to resolve them by skipping the computation on empty cells and focusing on informative ones. However, these networks are hard to be implemented efficiently.

Su et al. [95] was the first one to apply multi-view convolutional neural network (MV-CNN) for 3D recognition. The 3D shape is rendered in multiple views, each of which is passed through an identical image-based CNN. Features obtained from multiple views are combined via a view pooling (which is the max-pooling) and then passed through another CNN to predict the final object label. Xie et al. [96] used multi-view depth images via extreme learning machine to generate per-view segmentation and combine them via Graph-Cut. This method works pretty fast due to the easy training of the extreme learning machine but it does not give high accuracy. Later, Kalogerakis et al. [115] proposed a more complete multi-view framework. They first render the 3D model with different views, each of which is fed through a shared CNN before unprojected back to 3D. The label consistency is solved by a Conditional Random Field (CRF), which is part of the network and is optimized in an end-to-end manner. Le et al. [202] proposed a MV-RNN approach which treats multi-view images as a temporal sequence and uses recurrent neural network to correlate them. Although multi-view approaches generally give compelling results, it has several limitations. First, we need to carefully choose the rendering pipeline such as image resolution with respect to data sampling density, lighting, blending and keep track of the camera parameters. Second, each view only contains partial information and it is not trivial to correlate across views. Third, multi-view approaches are limited to model only the object's surface and cannot capture 3D internal structures.

Representatives for the later direction of adapting to the 3D irregularity include PointNet [116], PointNet++ [196], SpecCNN [117]. Su et al. [116] proposed PointNet as the first neural network which directly consumes 3D point clouds. PointNet is pretty fast and

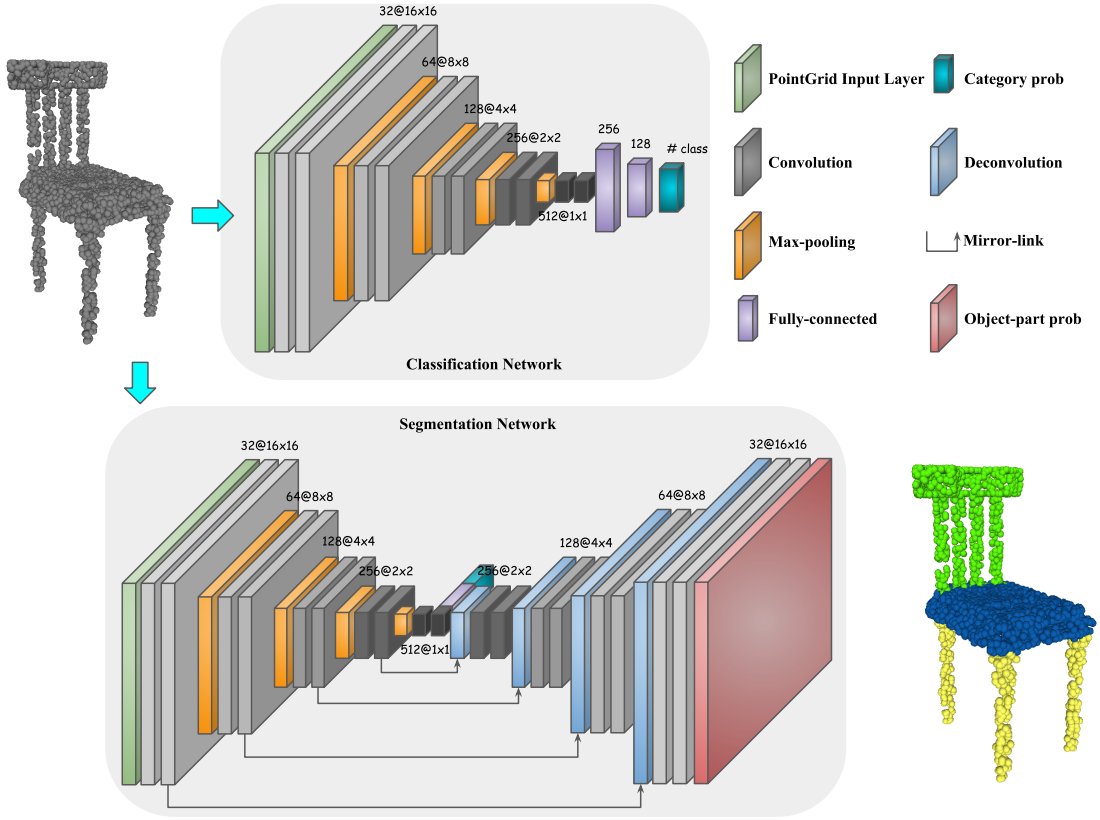


Figure 5.2: The architecture of PointGrid. Starting from the grid obtained from our sampling strategy, both classification and segmentation networks share the feature extraction (encoder) part. Segmentation network uses skip connections to preserve information of different hierarchical levels. All convolutions, deconvolutions and fully connected layers include batch normalization and ReLU (except object category and object-part segmentation layers). The notion 32@16x16 means there are 32 convolutional filters and the spatial dimension is 16×16. The network is visualized in 2D.

robust to rigid transformation and points' ordering. Its main limitation is relying on only the max-pooling to have context information. Later, PointNet++ was developed to compensate this weakness. Yi et al. [117] proposed SpecCNN uniquely designed for polygonal mesh. SpecCNN converts convolution to multiplication in spectral domain by Fourier analysis.

5.3 PointGrid

Volumetric grid is widely used for 3D deep learning due to its regularity. However the use of relatively lower order local approximation functions such as piece-wise constant

function (occupancy grid) or piece-wise linear function (distance field) to approximate 3D shape means it needs a very high-resolution grid to represent finer geometry details, which could be memory and computationally inefficient.

In this work, we propose the PointGrid, a 3D convolutional network that incorporates a constant number of points within each grid cell thus allowing the network to learn higher order local approximation functions that could better represent the local geometry shape details (Fig. 5.1).

We now introduce our PointGrid, starting with the discussion of its input format, then discussing its architecture for classification, and finally discussing how to use it for semantic segmentation of 3D point clouds.

5.3.1 Input Layer

The new deep architecture works with 3D grid constructed for 3D point clouds. We normalize the point cloud to the unit box $[-1,1]^3$ and this is the only preprocessing step in our framework. In contrast to VoxNet which uses occupancy grid as the primary representation of the 3D structure, we stack the points' coordinates as features for each cell. As a result, a cell with K points will have features $3K$ for the corresponding x, y and z coordinates. However, each cell has different number of points and constructing such grid is infeasible for sharing 3D convolutional kernels.

To solve this issue, we use a simple yet effective sampling strategy which we call *Point Quantization*, to keep a fixed number of K points in each cell. More specifically, if there are more than K points, we randomly sample K of them. If there are less than K points, we sample with replacement K of them. We pad zeros to cells with no point. This mimics the commonly used zero-padding to compensate the boundary loss in convolutions. The value of K theoretically can be approximated by the total number of points (P) divided by the number of cells in the grid (N^3) (i.e. $K \approx \frac{P}{N^3}$). When working with point clouds of size $P = 1024$, we empirically set $K = 4$ with a grid size of $16 \times 16 \times 16$ for the best trade-off

between accuracy and performance.

5.3.2 Classification Network

The classification network of PointGrid is illustrated in 2D in Fig. 5.2, which extracts features from input grid. Hence, we need the architecture to be deep and efficiently generate perceptually multi-level features. PointGrid consists of several blocks of convolutions followed by max-pooling to represent different hierarchical feature representations. Each convolution layer includes a $3 \times 3 \times 3$ kernel with stride 1 convolution, a batch normalization [178] and a rectified linear unit (ReLU) [118, 100]. The first block use 32-filter convolutions and they are doubled in each successive block. The pooling layers not only provide another form of translation invariance but also serve to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. All of our pooling layers are max-pooling which reduce the grid size half in each spatial dimension. After several convolutional and max pooling layers, as usual, the high-level reasoning in our PointGrid is done via fully connected layers. PointGrid has two fully connected layers, each of which consists of a fully connected layer, a ReLU activation and a dropout layer (with dropout rate 0.3). Finally, another fully connected layer followed by a softmax to regress to the probability of each category. The number of nodes in this layer equals to the number of object categories in the dataset.

5.3.3 Segmentation Network

Our segmentation network shares the feature extraction (or encoder) from the classification network (see Fig. 5.2) and decodes the extracted features to build the segmentation. It is intuitive that labeling object parts depends on the object category so we include both the high-level features in the last fully connected layer and the object category probability for

better global features. The decoder is almost symmetric to the encoder with convolutions replaced by deconvolutions (or sometimes referred as transposed convolutions). We optimize both networks simultaneously. The final loss is a linear combination of classification loss and segmentation loss.

The classification network progressively extracts and down-samples features, while the segmentation counterpart upsamples and combines them to construct the output. The sizes of feature maps are exactly mirrored in our network. We link early encoded features (from the classification network) to the corresponding decoded features (from the segmentation network) at the same spatial resolution, in order to obtain local sharp details preserved in early encoder layers. A mirror-link is a short notation for concatenation a copy followed by convolution. Besides the sharpness, mirror-links could also make the training converge faster.

The segmentation network produces $K+1$ labels for each cell in the 3D grid with K labels correspond to K points in that cell and one additional cell-level label. To obtain the ground truth labels for object parts at the cell-level, we take the majority label among labels of points in each cell. Cells with no point (and hence are filled with zeros) are labeled as "no label" and so are all the points within those cells. In testing, if there are less than or equal to K points in each cell, we use the corresponding K labels for each of them. Otherwise, the remaining points take the cell-level label.

5.3.4 Implementation details

Data augmentation

During training, before sampling to input grid, we augment the point cloud on-the-fly by randomly rotating the object along the up-axis and jittering the position of each points by a Gaussian noise with zero mean and 0.02 standard deviation. The sampling layer of our PointGrid also serves as an additional data augmentation.

Table 5.1: Object classification results on ModelNet40 [4].

Method	Input	Accuracy Overall	Accuracy Avg. Class
SPH [203]	mesh	–	68.2
3DShapeNets [4]	volume	84.7	77.3
VoxNet [189]	volume	85.9	83.0
Subvolume [97]	volume	89.2	86.0
VRN (simple) [191]	volume	91.3	–
VRN (ensemble) [191]	volume	95.5	–
LFD [4]	image	–	75.5
MVCNN [95]	image	–	90.1
FusionNet [204]	volume & image	90.8	–
Set-Conv [205]	point	90.0	–
PointNet [116]	point	89.2	86.2
PointNet++ [196]	point	91.9	–
Kd-Net [193]	point	91.8	–
O-CNN [194]	point	90.6	–
PointGrid (ours)	point	92.0	88.9

Table 5.2: Object classification results on ShapeNet-55 [5].

Method	Input	Accuracy Overall	Accuracy Avg. Class
PointNet [116]	point	83.2	78.2
PointGrid (ours)	point	86.1	80.5

Table 5.3: Accuracy of PointGrid's alternative structures on ModelNet40 [4].

Grid \ K	1	2	4	8	16
$4 \times 4 \times 4$	77.3	80.8	81.9	85.9	84.7
$8 \times 8 \times 8$	87.1	87.9	87.5	88.3	87.4
$16 \times 16 \times 16$	90.0	91.9	92.0	91.4	91.0
$32 \times 32 \times 32$	91.7	92.2	92.4	92.7	92.4

Training

We set batch size as 32, batch normalization initial decay as 0.5, batch normalization decay clipping as 0.99. The weights for classification and segmentation losses are 0.2 and 0.8, respectively. Both losses are cross-entropy. We use Adam optimizer [125] with initial learning rate of 10^{-4}. Adam realizes the benefits of both AdaGrad [206] and RMSProp [207]. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). We implement our PointGrid using the public deep learning library TensorFlow [180] and train it using Nvidia Titan X for 8 hours for classification and 20 hours for segmentation (equivalent to 100 epochs). PointGrid consumes 4.0GB and 8.4GB of memory for classification and segmentation network, respectively. These figures are for grid size $N = 16$, $K = 4$ for each cell and batch size of 32.

5.4 Experiments

We now discuss some experimental results of applying PointGrid to shape classification and object-part segmentation benchmark datasets. For classification, we evaluate PointGrid with various hyper-parameters such as the grid size (N) and the number of points per cell (K).

5.4.1 Shape Classification

Datasets: The ModelNet40 [4] benchmark contains 12,311 CAD models from 40 man-made object categories, which have been extensively used for 3D shape classification. The dataset is split into the training (9843 models) and the testing (2468) sets. ShapeNet-Core55 [5] benchmark contains a total of 51,190 3D models with 55 categories and 204 subcategories. The models are normalized to a unit length cube and have a consistent upright

Table 5.4: Object-part segmentation results on ShapeNet-part [6].

Method	# shapes		Wu	Yi	3D	Point-	Kd-Net	SpecCNN	O-CNN	PointGrid
	test	total	[201]	[6]	CNN	Net [116]	[193]	[117]	[194]	(ours)
mean IoU			–	81.4	79.4	83.7	77.2	84.7	85.9	86.4
airplane	341	2690	63.2	81.0	75.1	83.4	79.9	81.6	85.5	85.7
bag	14	76	–	78.4	72.8	78.7	71.2	81.7	87.1	82.5
cap	11	55	–	77.7	73.3	82.5	80.9	81.9	84.77	81.8
car	158	898	–	75.7	70.0	74.9	68.8	75.2	77.0	77.9
chair	704	3758	73.5	87.6	87.2	89.6	88.0	90.2	91.1	92.1
earphone	14	69	–	61.9	63.5	73.0	72.4	74.9	85.1	82.4
guitar	159	787	–	92.0	88.4	91.5	88.9	93.0	91.9	92.7
knife	80	392	–	85.4	79.6	85.9	86.4	86.1	87.4	85.8
lamp	286	1547	74.4	82.5	74.4	80.8	79.8	84.7	83.3	84.2
laptop	83	451	–	95.7	93.9	95.3	94.9	95.6	95.4	95.3
motor	51	202	–	70.6	58.7	65.2	55.8	66.7	56.9	65.2
mug	38	184	–	91.9	91.8	93.0	86.5	92.7	96.2	93.4
pistol	44	283	–	85.9	76.4	81.2	79.3	81.6	81.6	81.7
rocket	12	66	–	53.1	51.2	57.9	50.4	60.6	53.5	56.9
skateboard	31	152	–	69.8	65.3	72.8	71.1	82.9	74.1	73.5
table	848	5271	74.8	75.3	77.1	80.6	80.2	82.1	84.4	84.6

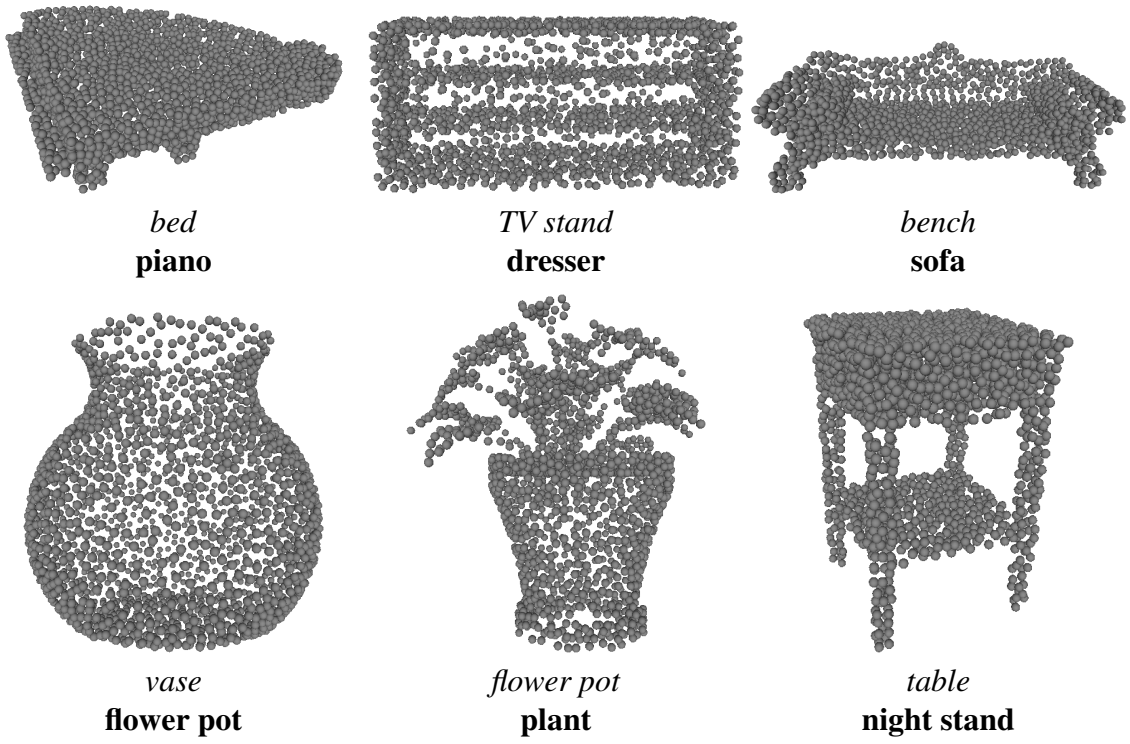


Figure 5.3: Some wrongly classified models. Predicted and actual labels are italicized and highlighted, respectively.

orientation. 70% of the dataset is used for training, 10% for validation, and 20% for testing.

Sampling point cloud: Given a triangular mesh model, 3D point cloud is computed as follows: firstly, a given number of triangles are sampled with the probability proportional to their surface areas. Then, for the sampled triangle a random point was taken by the following sampling equation.

$$(1-\sqrt{r_1})A+\sqrt{r_1}(1-r_2)B+r_2\sqrt{r_1}C \tag{5.1}$$

where A,B,C are the coordinates of the triangle's vertices and r_1,r_2 are random real numbers with $r_1,r_2 \sim U(0,1)$.

The whole sampling procedure thus closely approximated uniform sampling of model surfaces. For each model, we uniformly sample 1024 points on the surface.

Results: Our PointGrid gets 92.0% and 86.1% overall accuracy on ModelNet40 and ShapeNet-

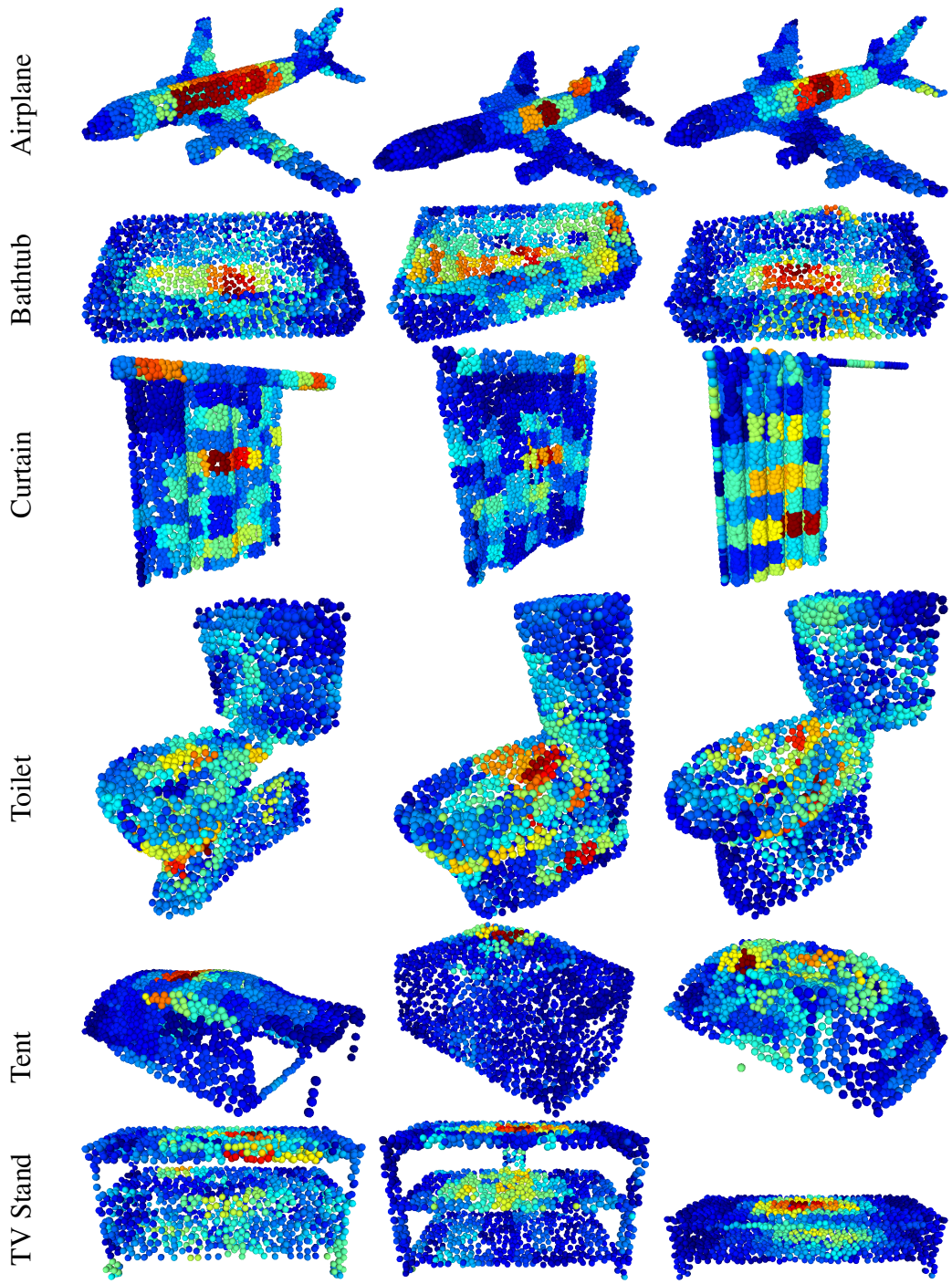


Figure 5.4: Visualization of object saliency. Magnitude of the gradient of the probability w.r.t. input grid is populated to points. Red indicates highly salient regions.

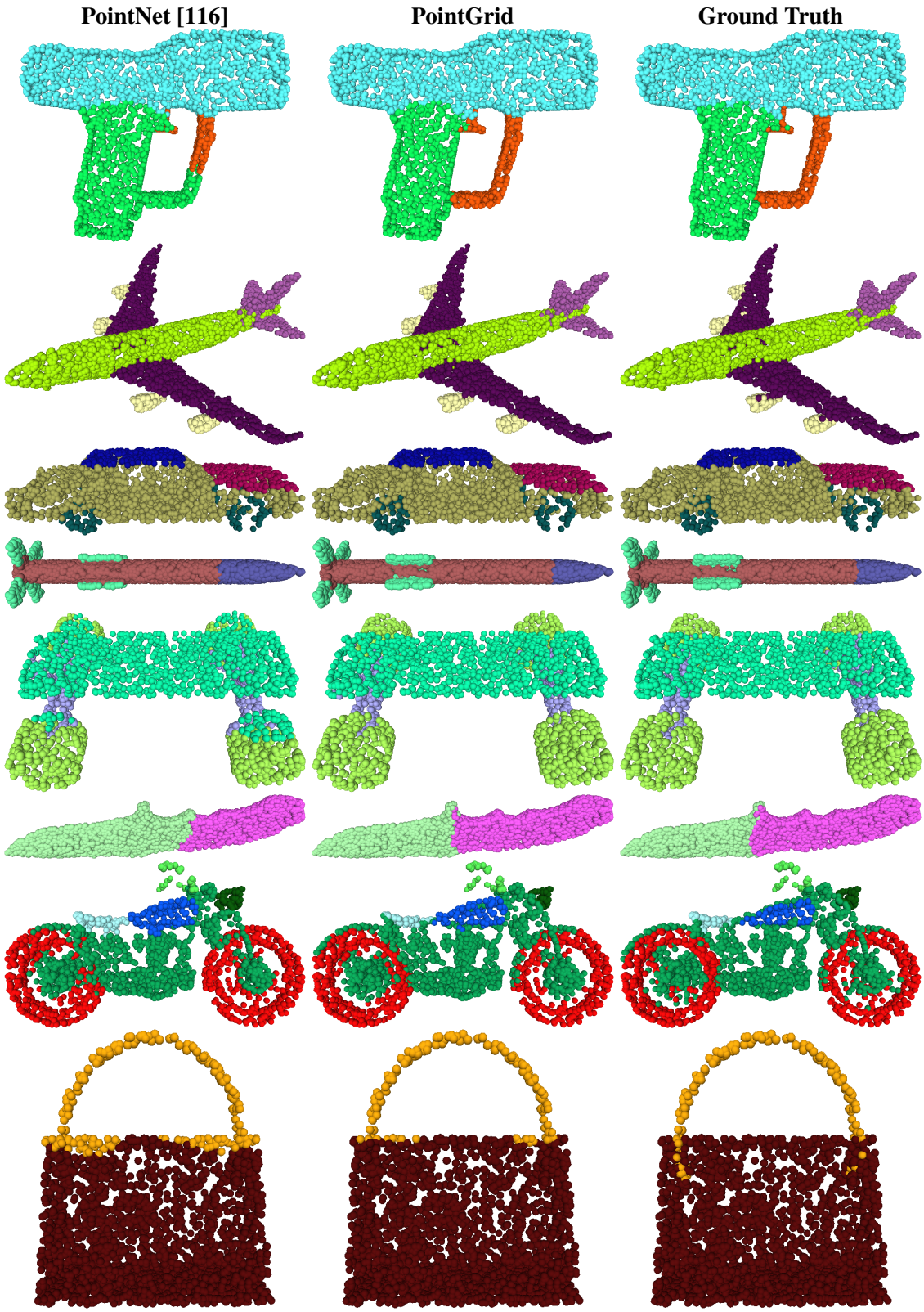


Figure 5.5: Comparison between PointGrid and PointNet on object-part segmentation. This result is based on grid size $16 \times 16 \times 16$ and $K = 4$.

Core55 datasets, respectively. Fig. 5.3 shows some wrongly classified models. As we can see, there are still some ambiguities in categories whose appearances could be similar such as bench versus sofa and flower pot versus vase.

Comparison: In Table 5.1, we compare our method with a representative set of state of the art methods. In the category of "accuracy overall", our PointGrid performs better than the majority of other voxel-based approaches such as 3DShapeNet [4], VoxNet [189], Subvolume [97] and VRN (single) [191] while being worse than the VRN-ensemble [191] which involves an ensemble of 6 models each trained separately over the course of 6 days on NVidia Titan X. Comparing with methods applied to point cloud, despite of its simplicity, our network edges out Set-Conv [205], PointNet [116], PointNet++ [196], Kd-Net [193] and O-CNN [194]. One of the reason may be because our method better captures points' contextual neighborhood and hence learns better high-level features. There is still a small gap between our method and multi-view based method (MVCNN [95]) in the "average class accuracy" which may be due to the fact that MVCNN used a higher resolution to represent an object (80 views of 224×224 image versus $16 \times 16 \times 16 \times 4$ in our PointGrid).

Table 5.2 shows a side-by-side comparison between our PointGrid and PointNet [116]. Our method outperforms PointtNet [116] in both categories, 2.9% overall accuracy and 2.3% average-class accuracy.

Alternative network architecture: We conduct experiments with alternative PointGrid's architectures (on N and K) and report their overall accuracy for the object classification on ModelNet40 dataset [4]. According to Table 5.3, increasing the grid resolution (with extra memory and computational cost) increases the accuracy as it captures finer details. However, the improvement keeps decreasing each time we double the size of the grid. This might be due to the relatively lower sampling resolution of the 3D data in the experiment where only 1024 points are sampled for each model. Regarding to the number of points per cell K, again keep using more points is not always better. This may also be due to fact that with the lower sampling rate of the 3D data, the point cloud has already been well repre-

sented with K points per cell, and further increasing K could result in randomly duplicating data which could make it harder for the convolution to extract good features. We choose $N = 16$ and $K = 4$ for the best trade-off between accuracy and the memory/computation usage.

Visualization: Fig. 5.4 shows the magnitude of the gradient of the highest predicted probability with respect to the input cell with blue to red indicates low to high magnitude. We populate the gradient magnitude from each cell to all the underlying points. Loosely speaking, a large gradient's magnitude indicates that a small change of points within that cell results in a large change in its classification probability. Therefore, this figure could be thought as saliency map of the objects. It is interesting that to classify an object, PointGrid looks for some tube structure for airplane, curvy surface for curtain, roof for tent and flat horizontal surface for TV stand.

5.4.2 Object-part Segmentation

Dataset: We evaluate our architecture for object-part segmentation on ShapeNet-part dataset from [6], which augments a subset of the ShapeNet models with semantic part annotations. It contains 16,881 shapes represented as separate point clouds from 16 categories with per point annotation (with 2 to 6 parts per category and 50 parts in total). We use the same training/testing split provided by [116, 117]. In this dataset, both the object categories and the object parts within the categories are highly imbalanced, which poses a challenge to all methods including ours.

Comparison: Fig. 5.5 displays some examples of segmentation results of our PointGrid compared with those of PointNet [116]. As we can see, our results are visually better in most cases. For example, our PointGrid can separate out the wheels from the body of a motorcycle while PointNet cannot. Similar observations can be made for other models such as the pistol, skateboard and cap.

Table 5.4 numerically compares the segmentation performance of our PointGrid against

Table 5.5: Average testing time on ModelNet40 [4].

	Classification	**Segmentation**
PointNet [116]	8.98ms	28.37ms
3D CNN (32^3)	27.50ms	64.32ms
3D CNN (64^3)	49.12ms	136.54ms
PointGrid (16^3)	14.91ms	48.94ms

other deep learning approaches. Evaluation metric is per-category and mean IoU on points. In the category of "mean IoU", our PointGrid achieves the best accuracy. In individual categories, we rank the best in airplane, car, chair and table; the second best in bag, earphone, guitar, lamp, mug and pistol; the third in cap, motor, rocket and skateboard. Our method places the fourth in laptop, and the fifth in knife. As we can see, our method performs better when there is more data in the category such as airplane, chair, table, etc. Note that we only use grid size of $16 \times 16 \times 16$ where all the other volumetric methods use at least $32 \times 32 \times 32$ grid. Moreover, we do not have any post-processing step such as boundary refinement by Conditional Random Field as is done in O-CNN [194].

Time complexity: Table 5.5 shows average inference time of our method on both classification and segmentation tasks (We test our method on individual data sequentially). For comparison, we run the authors' code of PointNet [116]. Since the 3DShapeNet [4] does not conduct segmentation, we extend the network in [4] to a fully convolutional network by omitting the fully connected layers and symmetrically adding the deconvolution layers. As we can see, PointGrid with grid resolution of 16^3 is faster than 3D CNN methods such as [4] with grid resolution of 32^3, and still outperforms it in the accuracy as shown in our work. It is slower than PointNet but it performs better than PointNet in both classification and segmentation.

5.5 Conclusion

In this work we propose a new deep learning architecture PointGrid suitable for 3D visual recognition tasks such as 3D classification and semantic segmentation. PointGrid is a hybrid representation of point and grid that can better capture local geometric details while exhibits easy-to-learn regular structure. Experiments on widely used benchmark datasets [4, 5, 6] show that PointGrid compares favorably over existing deep learning methods [201, 6, 116, 193, 117, 194] on both classification and segmentation with significantly smaller memory footprint than other volumetric approaches.

Currently in the "point quantization" step of Section 5.3.1, if there are more than K points, we randomly sample K of them. Although it works well in our experiment, we would like to explore other advanced sampling techniques, e.g. furthest sampling [208], which may provide better representation of the local shape.

Acknowledgement

We would like to thank the authors of ModelNet40 [4] and ShapeNet [5, 6] datasets. We also appreciate Su et el. [116], Wang et al. [194] for making PointNet and O-CNN publicly available.

Chapter 6

Summary

I have presented my work towards applying machine learning for 3D object classification and segmentation. For engineering CAD models with a predefined set of geometric primitives such as planes, spheres, cylinders, cones, tori, we propose to a hypothesis generation and optimization approaches. The hypothesis generation reduces the dimensionality by first detecting the model's major orientations along which the model is sliced, detecting circles in each slice, ensembling them and analyzing the profile curve to produce geometric primitives. The generated primitives are selected via the set cover optimization.

For generic objects with free-form shapes, we proposed two approaches, MVRNN and PointGrid. The former approach is multi-view based and more suitable for surface data. MVRNN generates segmentation of the model for each view and correlates them via recurrent neural network. The later approach is 3D-based and better utilizes 3D information of the input data. PointGird is, therefore, better represents the 3D local geometry of the shape. Both of these approaches demonstrate favorably results compared with other state of the art non-deep learning and deep learning methods.

In the future, we plan to extend both MVRNN and PointGrid for more memory efficient and better sampling strategy to handle large scale data.

Bibliography

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 5 2011.

[2] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *IEEE European Conference on Computer Vision*, 2012.

[3] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 891–898, 2014.

[4] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *Computing Research Repository - arXiv*, 2015.

[6] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics*, 35(6):210:1–210:12, 11 2016.

[7] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, 6 2007.

[8] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52:1–52:12, 2011.

[9] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. Fast and robust detection of crest lines on meshes. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*, pages 227–232, 2005.

[10] Paul Leopardi. A partition of the unit sphere into regions of equal area and small diameter. *Electronic Transactions on Numerical Analysis*, 25, 6 2006.

[11] Bodi Yuan and Min Liu. Power histogram for circle detection on images. *Pattern Recognition*, 48(10):3268–3280, 2015. Discriminative Feature Learning from Big Data for Visual Recognition.

[12] Cuneyt Akinlar and Cihan Topal. Edcircles: A real-time circle detector with a false detection control. *Pattern Recognition*, 46(3):725–740, 3 2013.

[13] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 11 1986.

[14] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[15] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. Never published but presented at a talk at the Stanford Artificial Project, 1968.

[16] Piotr Dollar and C. Lawrence Zitnick. Structured forests for fast edge detection. In *IEEE International Conference on Computer Vision*, 2013.

[17] Roseline Beniere, Gerard Subsol, Gilles Gesquiere, Francois Le Breton, and William Puech. Recovering primitives in 3d cad meshes. *Processing of the International Society for Optics and Photonics*, 7864:78640R–78640R–9, 2011.

[18] Roseline Beniere, Gerard Subsol, Gilles Gesquiere, Francois Le Breton, and William Puech. A comprehensive process of reverse engineering from 3D meshes to cad models. *Computer Aided Design*, 45(11):1382–1393, 2013.

[19] Moritz Tenorth, Stefan Profanter, Ferenc Balint-Benczedi, and Michael Beetz. Decomposing cad models of objects of daily use and reasoning about their functional parts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5943–5949, 2013.

[20] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 9 2008.

[21] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. *ACM Transactions on Graphics*, 28(3):73:1–73:12, 7 2009.

[22] Tamas Varady, Ralph R Martin, and Jordan Cox. Reverse engineering of geometric modelsan introduction. *Computer Aided Design*, 29(4):255–268, 1997.

[23] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 6 1981.

[24] Hossam Isack and Yuri Boykov. Energy-based geometric multi-model fitting. *International Journal of Computer Vision*, 97:123–147, 4 2012.

[25] Y. Zhang, J. Paik, A. Koschan, M.A. Abidi, and D. Gorsich. Simple and efficient algorithm for part decomposition of 3-d triangulated models based on curvature analysis. In *International Conference on Image Processing*, volume 3, pages III–273–III–276 vol.3, 6 2002.

[26] Emanoil Zuckerberger, Ayellet Tal, and Shymon Shlafman. Polyhedral surface decomposition with applications. *Computers & Graphics*, 26(5):733–743, 2002.

[27] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, 8 2004.

[28] Jianhua Wu and Leif Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[29] Dong-Ming Yan, Yang Liu, and Wenping Wang. Quadric surface extraction by variational shape approximation. In Myung-Soo Kim and Kenji Shimada, editors, *GMP*, volume 4077 of *Lecture Notes in Computer Science*, pages 73–86, 2006.

[30] Dong-Ming Yan, Wenping Wang, Yang Liu, and Zhouwang Yang. Variational mesh segmentation via quadric surface fitting. *Computer Aided Design*, 44(11):1072–1082, 11 2012.

[31] Aron Monszpart, Nicolas Mellado, Gabriel Brostow, and Niloy Mitra. Rapter: Rebuilding man-made scenes with regular arrangements of planes. *ACM Transactions on Graphics*, 34(4):103:1–103:12, 7 2015.

[32] İlke Demir, Daniel G. Aliaga, and Bedrich Benes. Coupled segmentation and similarity detection for architectural models. *ACM Transactions on Graphics*, 34(4):104:1–104:11, 7 2015.

[33] Aleksey Golovinskiy, Vladimir G. Kim, and Thomas Funkhouser. Shape-based recognition of 3D point clouds in urban environments. *International Conference on Computer Vision*, 9 2009.

[34] Irene Reisner-Kollmann, Christian Luksch, and Michael Schwarzler. Reconstructing buildings as textured low poly meshes from point clouds and images. In *Eurographics*, pages 17–20, 4 2011.

[35] Rongqi Qiu, Qian-Yi Zhou, and Ulrich Neumann. Pipe-run extraction and reconstruction from point clouds. *IEEE European Conference on Computer Vision*, 8691:17–30, 2014.

[36] Alexander Agathos, Ioannis Pratikakis, Stavros Perantonis, Nikolaos Sapidis, and Philip Azariadis. 3d mesh segmentation methodologies for cad applications. *Computer Aided Design and Applications*, 4(6):827–841, 2007.

[37] Panagiotis Theologou, Ioannis Pratikakis, and Theoharis Theoharis. A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation. *Computer Vision and Image Understanding*, 135:49–82, 2015.

[38] Dong Xiao, Hongwei Lin, Chuhua Xian, and Shuming Gao. Cad mesh model segmentation by clustering. *Computers & Graphics*, 35(3):685–691, 2011. Shape Modeling International (SMI) Conference 2011.

[39] Guillaume Lavoue, Florent Dupont, and Atilla Baskurt. A new cad mesh segmentation method based on curvature tensor analysis. *Computer-Aided Design*, 37(10):975–987, 2005.

[40] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259, 2008.

[41] Miguel Vieira and Kenji Shimada. Surface mesh segmentation and smooth surface extraction through region growing. *Computer Aided Geometric Design*, 22:771–792, 2005.

[42] A. Jagannathan and E.L. Miller. Three-dimensional surface mesh segmentation using curvedness-based region growing approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2195–2204, 12 2007.

[43] A.P. Mangan and R.T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 10 1999.

[44] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22:181–193, 2006.

[45] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Processing of the Symposium on Interactive 3D Graphics*, I3D '01, pages 49–58, 2001.

[46] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–961, 7 2003.

[47] Hitoshi Yamauchi, Seungyong Lee, Yunjin Lee, Yutaka Ohtake, Alexander G. Belyaev, and Hans-Peter Seidel. Feature sensitive mesh segmentation with mean shift. In *In Processing of the International Conference on Shape Modeling and Applications*, pages 238–245, 2005.

[48] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 8 2000.

[49] Bing Yi, Zhenyu Liu, Jianrong Tan, Fengbei Cheng, Guifang Duan, and Ligang Liu. Shape recognition of cad models via iterative slippage analysis. *Computer Aided Design*, 55(0):13–25, 2014.

[50] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics*, pages 259–263, 2004.

[51] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 7 2005.

[52] Yu-Kun Lai, Shi-Min Hu, Ralph R. Martin, and Paul L. Rosin. Fast mesh segmentation using random walks. In *Processing of the ACM Symposium on Solid and Physical Modeling*, SPM '08, pages 183–191, 2008.

[53] William Benjamin, Andrew Wood Polk, S.V.N. Vishwanathan, and Karthik Ramani. Heat walk: Robust salient segmentation of non-rigid shapes. *Computer Graphics Forum*, 30(7):2097–2106, 2011.

[54] Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or. Generalized cylinder decomposition. *ACM Transactions on Graphics*, 34(6):171:1–171:14, 10 2015.

[55] Ying He, Hongyu Wang, Chi-Wing Fu, and Hong Qin. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics*, 33(3):369–380, 2009. IEEE Int. Conf. on Shape Model. and App.

[56] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 8 1995.

[57] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116, 1988.

[58] Teh-Chuan Chen and Kuo-Liang Chung. An efficient randomized algorithm for detecting circles. *Computer Vision and Image Understanding*, 83(2):172–191, 2001.

[59] Ali Ajdari Rad, Karim Faez, and Navid Qaragozlou. Fast circle detection using gradient pair vectors. In *In the Digital Image Computing: Techniques and Applications*, pages 879–887, 2003.

[60] Heung-Soo Kim and Jong-Hwan Kim. A two-step circle detection algorithm from the intersecting chords. *Pattern Recognition Letters*, 22(6-7):787–798, 5 2001.

[61] Chun-Ta Ho and Ling-Hwei Chen. A fast ellipse/circle detector using geometric symmetry. *Pattern Recognition*, 28(1):117–124, 1995.

[62] Erik Cuevas, Diego Oliva, Daniel Zaldivar, Marco Pérez-Cisneros, and Humberto Sossa. Circle detection using electro-magnetism optimization. *Information Sciences*, 182(1):40–55, 1 2012.

[63] Truc Le and Ye Duan. Circle detection on images by line segment and circle completeness. In *IEEE International Conference on Image Processing*, pages 3648–3652, 9 2016.

[64] P.V. Hough. Method and means for recognizing complex patterns, 12 1962. US Patent 3,069,654.

[65] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1 1972.

[66] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Computing*, 8(1):71–77, 2 1990.

[67] Raymond K.K. Yip, Peter K.S. Tam, and Dennis N.K. Leung. Modification of hough transform for circles and ellipses detection using a 2-dimensional array. *Pattern Recognition*, 25(9):1007–1022, 1992.

[68] Dimitrios Ioannou, Walter Huda, and Andrew F. Laine. Circle recognition through a 2d hough transform and radius histogramming. *Image and Vision Computing*, 17(1):15–26, 1999.

[69] Shih-Hsuan Chiu and Jiun-Jian Liaw. An effective voting method for circle detection. *Pattern Recognition Letters*, 26(2):121–133, 1 2005.

[70] J. Illingworth and J. Kittler. The adaptive hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):690–698, 9 1987.

[71] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: Randomized hough transform (rht). *Pattern Recognition Letters*, 11(5):331–338, 5 1990.

[72] Y. C. Cheng and Y. S. Liu. Polling an image for circles by random lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):125–130, 1 2003.

[73] Kuo-Liang Chung, Yong-Huai Huang, Shi-Ming Shen, Andrey S. Krylov, Dmitry V. Yurin, and Ekaterina V. Semeikina. Efficient sampling strategy and refinement strategy for randomized circle detection. *Pattern Recognition*, 45(1):252–263, 2012.

[74] Yong-Huai Huang, Kuo-Liang Chung, Wei-Ning Yang, and Shih-Hsuan Chiu. Efficient symmetry-based screening strategy to speed up randomized circle-detection. *Pattern Recognition Letters*, 33(16):2071–2076, 12 2012.

[75] Victor Ayala-Ramirez, Carlos H. Garcia-Capulin, Arturo Perez-Garcia, and Raul E. Sanchez-Yanez. Circle detection on images using genetic algorithms. *Pattern Recognition Letters*, 27(6):652–657, 4 2006.

[76] Wei Lu and Jinglu Tan. Detection of incomplete ellipse in images with strong noise by iterative randomized hough transform (irht). *Pattern Recognition*, 41(4):1268–1279, 4 2008.

[77] X. Hilaire and K. Tombre. Robust and accurate vectorization of line drawings. *PAMI*, pages 890–904, 6 2006.

[78] B. Lamiroy and Y. Guebbas. Robust and precise circular arc detection. In *8th Int. Conf. on Graph. Recog: Achieve. Challen., and Evo.*, volume 6020, pages 49–60, 2009.

[79] S. Bera, P. Bhowmick, and B. B. Bhattacharya. Detection of circular arcs in a digital image using chord and sagitta properties. In *8th Int. Conf. on Graph. Recog: Achieve. Challen., and Evo.*, volume 6020, pages 69–80, 2009.

[80] R.G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, 4 2010.

[81] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 1–42, 4 2015.

[82] Gabor Lukacs, Ralph Martin, and Dave Marshall. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In *IEEE European Conference on Computer Vision*, pages 671–686, 1998.

[83] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin Heidelberg, 5th edition, 2012.

[84] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[85] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[86] Xiaoming Zhang and Paul L. Rosin. Superellipse fitting to partial data. *Pattern Recognition*, 36(3):743–752, 2003.

[87] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Shape Modeling International*, 6 2004.

[88] S. Biasotti, S. Marini, M. Mortara, and G. Patane. An overview on properties and efficacy of topological skeletons in shape modeling. In *Shape Modeling International*, pages 245–254, 5 2003.

[89] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Transactions on Graphics*, 23(3):652–663, 8 2004.

[90] Malte Zockler, Detlev Stalling, and Hans-Christian Hege. Fast and intuitive generation of geometric shape transitions. *The Visual Computer*, 16(5):241–253, 2000.

[91] Bruno Levy, Sylvain Petitjean, Nicolas Ray, and Jerome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 7 2002.

[92] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, 7 2003.

[93] Michael Burns, Janek Klawe, Szymon Rusinkiewicz, Adam Finkelstein, and Doug DeCarlo. Line drawings from volume data. *ACM Transactions on Graphics*, 24(3):512–518, 8 2005.

[94] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Transactions on Graphics*, 23(3):609–612, 8 2004.

[95] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *IEEE International Conference on Computer Vision*, 2015.

[96] Zhige Xie, Kai Xu, Wen Shan, Ligang Liu, Yueshan Xiong, and Hui Huang. Projective feature learning for 3d shapes with multi-view depth images. *Computer Graphics Forum*, 34(7):1–11, 2015.

[97] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas Guibas. Volumetric and multi-view CNNs for object classification on 3d data. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[98] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *IEEE European Conference on Computer Vision*, pages 628–644, 2016.

[99] Aleksey Golovinskiy and Thomas Funkhouser. Randomized cuts for 3d mesh analysis. *ACM Transactions on Graphics*, 27(5):145:1–145:12, 12 2008.

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[101] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2014.

[102] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 6 2015.

[103] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 6 2016.

[104] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 8 2013.

[105] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE International Conference on Pattern Recognition*, 11 2015.

[106] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE International Conference on Computer Vision*, 2015.

[107] A. Sharma, O. Tuzel, and D. W. Jacobs. Deep hierarchical parsing for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 530–538, 6 2015.

[108] Seunghoon Hong, Junhyuk Oh, Honglak Lee, and Bohyung Han. Learning transferrable knowledge for semantic segmentation with deep convolutional neural network. *Computing Research Repository - arXiv*, 2015.

[109] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *IEEE International Conference on Computer Vision)*, pages 1395–1403, 12 2015.

[110] Zhenyu Shu, Chengwu Qi, Shiqing Xin, Chao Hu, Li Wang, Yu Zhang, and Ligang Liu. Unsupervised 3d shape segmentation and co-segmentation via deep learning. *Computer Aided Geometric Design*, 43:39–52, 2016. Geometric Modeling and Processing 2016.

[111] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics*, 35(1):3:1–3:12, 12 2015.

[112] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 6 2014.

[113] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

[114] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 4 2017.

[115] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 3d shape segmentation with projective convolutional networks. *Computing Research Repository - arXiv*, 2016.

[116] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Computing Research Repository - arXiv*, 2016.

[117] Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3d shape segmentation. *Computing Research Repository - arXiv*, 2016.

[118] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *IEEE International Conference on Machine Learning*, pages 807–814, 2010.

[119] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[120] Zhen Li, Yukang Gan, Xiaodan Liang, Yizhou Yu, Hui Cheng, and Liang Lin. Lstm-cf: Unifying context modeling and fusion with lstms for rgb-d scene labeling. In *IEEE European Conference on Computer Vision*, pages 541–557, 2016.

[121] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 6 1975.

[122] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *IEEE European Conference on Computer Vision*, pages 345–360, 2014.

[123] Jurgen Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1:403–412, 1989.

[124] F. A. Gers and E. Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 11 2001.

[125] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *IEEE International Conference for Learning Representations*, 2015.

[126] Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-Or. Shape segmentation by approximate convexity analysis. *ACM Transactions on Graphics*, 34(1):4:1–4:11, 12 2014.

[127] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *ACM Transactions on Graphics*, pages 203–212, 2001.

[128] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24(1):1–27, 1 2005.

[129] Philipp Krahenbuhl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *IEEE International Conference on Neural Information Processing Systems*, pages 109–117, 2011.

[130] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8):649–658, 2005.

[131] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, 21(3):219–228, 2002.

[132] Kai Xu, Yifei Shi, Lintao Zheng, Junyu Zhang, Min Liu, Hui Huang, Hao Su, Daniel Cohen-Or, and Baoquan Chen. 3d attention-driven depth acquisition for object identification. *ACM Transactions on Graphics*, 35(6):238:1–238:14, 11 2016.

[133] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–547, 6 2015.

[134] Y. Chien. Pattern classification and scene analysis. *IEEE Transactions on Automatic Control*, 19(4):462–463, 8 1974.

[135] J. R. Fram and E. S. Deutsch. On the quantitative evaluation of edge detection schemes and their comparison with human performance. *IEEE Transactions on Computers*, 24(6):616–628, 6 1975.

[136] Guner S. Robinson. Color edge detection. *Optical Engineering*, 16(5):165479–165479, 1977.

[137] S. Ullman and R. Basri. Recognition by linear combinations of models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):992–1006, 10 1991.

[138] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1):36–51, 1 2008.

[139] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1):7–27, 6 2001.

[140] Iasonas Kokkinos. Pushing the boundaries of boundary detection using deep learning. In *International Conference on Learning Representations*, 2016.

[141] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[142] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 6 2016.

[143] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *International Conference on Learning Representations*, 2015.

[144] Alex Kendall, Vijay Badrinarayanan, , and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *Computing Research Repository - arXiv*, 2015.

[145] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2650–2658, 2015.

[146] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, and Ming-Hsuan Yang. Object contour detection with a fully convolutional encoder-decoder network. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[147] Yupei Wang, Xin Zhao, and Kaiqi Huang. Deep crisp boundaries. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1724–1732, 2017.

[148] Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Kai Wang, and Xiang Bai. Richer convolutional features for edge detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 5872–5881, 2017.

[149] K.K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool. Convolutional oriented boundaries: From image segmentation to high-level tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[150] Ming-Yu Liu Srikumar Ramalingam Zhiding Yu, Chen Feng. CASENet: Deep category-aware semantic edge detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2017.

[151] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[152] Jian Shi, Yue Dong, Hao Su, and Stella X. Yu. Learning non-lambertian object intrinsics across shapenet categories. *Computing Research Repository - arXiv*, 2016.

[153] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[154] Fritz M Gavves E Tuytelaars T Rematas K, Ritschel T. Deep reflectance maps. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[155] Xi Peng, Rogerio S. Feris, Xiaoyu Wang, and Dimitris N. Metaxas. A recurrent encoder-decoder network for sequential face alignment. In *IEEE European Conference on Computer Vision*, pages 38–56, 2016.

[156] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1637–1645, 2016.

[157] David Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London Series B*, 207:187–217, 1980.

[158] V Torre and T Poggio. On edge detection. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 8(2):147–163, 2 1986.

[159] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 7 1990.

[160] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 9 1991.

[161] K. Bowyer, C. Kranenburg, and S. Dougherty. Edge detector evaluation using empirical roc curves. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 1, page 359, 1999.

[162] Djemel Ziou and Salvatore Tabbone. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.

[163] David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, 5 2004.

[164] Piotr Dollar, Zhuowen Tu, and Serge Belongie. Supervised learning of edges and object boundaries. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1964–1971, 2006.

[165] S. Zheng, Z. Tu, and A. L. Yuille. Detecting object boundaries using low-, mid-, and high-level information. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 6 2007.

[166] Julien Mairal, Marius Leordeanu, Francis Bach, Martial Hebert, and Jean Ponce. Discriminative sparse image models for class-specific edge detection and image interpretation. In *IEEE European Conference on Computer Vision*, pages 43–56, 2008.

[167] Iasonas Kokkinos. Boundary detection using f-measure-, filter- and feature- (f3) boost. In *IEEE European Conference on Computer Vision*, pages 650–663, 2010.

[168] Nicolas Widynski and Max Mignotte. A particle filter framework for contour detection. In *IEEE European Conference on Computer Vision*, pages 780–793, 2012.

[169] M. Leordeanu, R. Sukthankar, and C. Sminchisescu. Generalized boundaries from multiple image interpretations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1312–1324, 7 2014.

[170] Xiaofeng Ren and Liefeng Bo. Discriminatively trained sparse code gradients for contour detection. In *IEEE International Conference on Neural Information Processing Systems*, pages 584–592, 2012.

[171] J. J. Lim, C. L. Zitnick, and P. Dollr. Sketch tokens: A learned mid-level representation for contour and object detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3158–3165, 6 2013.

[172] Piotr Dollar and C. Lawrence Zitnick. Fast edge detection using structured forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(8):1558–1570, 2015.

[173] Jyri Kivinen, Chris Williams, and Nicolas Heess. Visual boundary prediction: A deep neural prediction network and quality dissection. In *International Conference on Artificial Intelligence and Statistics*, volume 33, pages 512–521, 4 2014.

[174] George E. Dahl, Marc'Aurelio Ranzato, Abdel-rahman Mohamed, and Geoffrey Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In *IEEE International Conference on Neural Information Processing Systems*, pages 469–477, 2010.

[175] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2015.

[176] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Zhijiang Zhang. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3982–3991, 2015.

[177] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *International Journal of Computer Vision*, pages 1–16, 2017.

[178] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[179] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *IEEE International Conference on Machine Learning*, 2013.

[180] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[181] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[182] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal on Computer Vision*, 59(2):167–181, 9 2004.

[183] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2011–2018, June 2013.

[184] Sam Hallman and Charless C. Fowlkes. Oriented edge forests for boundary detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1732–1740, 2015.

[185] P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:128–140, 2014.

[186] Qiyang Zhao. Segmenting natural images with the least effort as humans. In *Proceedings of the British Machine Vision Conference*, pages 110.1–110.12, September 2015.

[187] Yaroslav Ganin and Victor Lempitsky. N4-fields: Neural network nearest neighbor fields for image transforms. In *IEEE Asian Conference on Computer Vision*, pages 536–551, 2015.

[188] Jyh-Jing Hwang and Tyng-Luh Liu. Pixel-wise deep learning for contour detection. In *IEEE International Conference on Learning Representations*, 2015.

[189] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 922–928, 9 2015.

[190] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015.

[191] Andrew Brock, Theodore Lim, J.M. Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *Computing Research Repository - arXiv*, 2016.

[192] Yangyan Li, Sren Pirk, Hao Su, Charles Ruizhongtai Qi, and Leonidas J. Guibas. Fpnn: Field probing neural networks for 3d data. In *IEEE International Conference on Neural Information Processing Systems*, pages 307–315, 2016.

[193] Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *Computing Research Repository - arXiv*, 2017.

[194] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics*, 36(4):72:1–72:11, 7 2017.

[195] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[196] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Computing Research Repository - arXiv*, 2017.

[197] Rong Liu, Hao Zhang, Ariel Shamir, and Daniel Cohen-Or. A part-aware surface metric for shape analysis. *Computer Graphics Forum*, 28(2):397–406, 2009.

[198] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 4 2002.

[199] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 5 1999.

[200] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. *ACM Transactions on Graphics*, 29(3), 2010.

[201] Zizhao Wu, Ruyang Shou, Yunhai Wang, and Xinguo Liu. Interactive shape co-segmentation via label propagation. *Computers & Graphics*, 38:248–254, 2014.

[202] Truc Le, Giang Bui, and Ye Duan. A multi-view recurrent neural network for 3d mesh segmentation. *Computers & Graphics*, 66(Supplement C):103–112, 2017.

[203] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 156–164, 2003.

[204] Reza Zadeh Vishakh Hegde. Fusionnet: 3d object classification using multiple data representations. *Computing Research Repository - arXiv*, 2016.

[205] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *Computing Research Repository - arXiv*, 2016.

[206] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[207] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. Lecture 6 - overview of mini-batch gradient descent. Computer Science lecture at University of Toronto.

[208] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 9 1997.

VITA

Truc Duc Le was born in Ho Chi Minh city, Vietnam. He attended the University of Science, Vietnam National University of Ho Chi Minh city where he graduated with a B.S. Degree in Computer Science in 2012. He got the Vietnam Education Foundation fellowship, a funding for Vietnamese young scientists monitored by the United States's government, to study towards his Ph.D. Degree in Computer Science at University of Missouri. In 2012, he began working with Prof. Ye Duan on 3D computer vision research with main focus on using deep learning to recognize and segment 3D data.