Fine-Scaled 3D Geometry Recovery from Single RGB Images

Kumulative Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Jun Li

aus

Hunan, V.R.China

Bonn, 2017

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn.

Erstgutachter: Prof. Dr. Reinhard Klein, Bonn
Zweitgutachter: Jun.-Prof. Dr. Angela Yao, Bonn
Tag der Promotion: 12.12.2017
Erscheinungsjahr: 2018

Abstract

3D geometry recovery from single RGB images is a highly ill-posed and inherently ambiguous problem, which has been a challenging research topic in computer vision for several decades. When fine-scaled 3D geometry is required, the problem become even more difficult. 3D geometry recovery from single images has the objective of recovering geometric information from a single photograph of an object or a scene with multiple objects. The geometric information that is to be retrieved can be of different representations such as surface meshes, voxels, depth maps or 3D primitives, *etc*.

In this thesis, we investigate fine-scaled 3D geometry recovery from single RGB images for three categories: facial wrinkles, indoor scenes and man-made objects. Since each category has its own particular features, styles and also variations in representation, we propose different strategies to handle different 3D geometry estimates respectively.

We present a lightweight non-parametric method to generate wrinkles from monocular Kinect RGB images. The key lightweight feature of the method is that it can generate plausible wrinkles using exemplars from one high quality 3D face model with textures. The local geometric patches from the source could be copied to synthesize different wrinkles on the blendshapes of specific users in an offline stage. During online tracking, facial animations with high quality wrinkle details can be recovered in real-time as a linear combination of these personalized wrinkled blendshapes.

We propose a fast-to-train two-streamed CNN with multi-scales, which predicts both dense depth map and depth gradient for single indoor scene images.The depth and depth gradient are then fused together into a more accurate and detailed depth map. We introduce a novel set loss over multiple related images. By regularizing the estimation between a common set of images, the network is less prone to overfitting and achieves better accuracy than competing methods. Fine-scaled 3D point cloud could be produced by re-projection to 3D using the known camera parameters.

To handle highly structured man-made objects, we introduce a novel neural network architecture for 3D shape recovering from a single image. We develop a convolutional encoder to map a given image to a compact code. Then an associated recursive decoder maps this code back to a full hierarchy, resulting a set of bounding boxes to represent the estimated shape. Finally, we train a second network to predict the fine-scaled geometry in each bounding box at voxel level. The per-box volumes are then embedded into a global one, and from which we reconstruct the final meshed model.

Experiments on a variety of datasets show that our approaches can estimate fine-scaled geometry from single RGB images for each category successfully, and surpass state-of-the-art performance in recovering faithful 3D local details with high resolution mesh surface or point cloud.

Acknowledgments

This dissertation would not have been accomplished without the support and cooperation of many people I want to thank hereby.

First of all, I would like to thank Prof. Dr. Reinhard Klein for your support during the past years of my Ph.D. study and related research, for your patience, enthusiasm, and immense knowledge. You gave me a lot of valuable advices during my Ph.D. study, providing a perfect balance of guidance and freedom. Your guidance helped me in all the time of research and writing of this thesis.

I furthermore want to express my gratitude to Jun.-Prof. Dr. Angela Yao. You regularly arranged individual discussions with me and I have learned a lot from you. You inspired me to new ideas, encouraged me to publish high quality papers and made me strive to become a better researcher. You also helped me a lot in improving my scientific English writing. The door to your office was always open whenever I had a question or ran into troubles .

During my Ph.D. study, I was affiliated with the Computer Graphics group, headed by Prof. Dr. Reinhard Klein. I would like to thank my colleagues in the group: Dr. Alexander Berner, Dr.Roland Ruiters, Dr.Ingrid Irmer, Dr.Max Hermann, Dr. Michael Weinmann, Dr. Raoul Wessel, Dr. Anna Vögele, Canling Shen, Ralf Sarlette, Oliver Burghard, Stefan Hartmann, Heinz Christian Steinhausen, Fabian Aiteanu. All of you gave me many help in my study and life in Germany.

I would like to thank my co-authors and also the anonymous reviewers of our papers. I am particularly indebted to Kai Xu, Zhiquan Cheng for your help and encouragement during my Ph.D. study.

My Ph.D. study at University of Bonn was mainly supported by the China Scholarship Council. I am sincerely grateful to the Chinese government for providing me with such a good opportunity to study in University of Bonn.

Last but not least, a heartful thank-you goes to my family. I would not be here today without the support, encouragement and love from my parents. My mother always encouraged me during my hardest days. It is the optimism, patient and hard work I learned from you which were accompanying me through my study.

Contents

Chapter 1

Introduction

3D geometry recovery from images is a fundamental problem of computer vision, and has been a core research topic for several decades. It has widespread applications such as 3D modeling, scene understanding, depth-aware image editing or re-rendering, robotics, *etc*. Most prior work has focused on stereovision or structure from motion when two or more required images are available. Benefiting from multi-view concepts like point correspondences and photo-consistency, it has been shown that high-quality dense reconstruction can be inferred from a set of photographs [Beeler et al., 2010, 2011]. However for the special case that only a single still image is available, the problem become much more difficult. Inferring 3D geometry from single RGB images still remains extremely challenging for machines in the last few decades, especially for recovering fine-scaled 3D details from single view.

3D geometry recovery from single images has the objective of recovering geometric information from a single photograph of an object or a scene with multiple ones. The geometric information that is to be retrieved can be of different representations such as 3D surface meshes, voxels, depth maps or 3D primitives, *etc*. Unlike human vision which has impressive ability to understand the three dimensional (3D) structures from single two dimensional (2D) images, single view 3D geometry recovery is a highly ill-posed and inherently ambiguous problem for machines where the stereo and multiview concepts can not be applied. A key strategy in early work for solving the ambiguity was to use strong assumptions and prior knowledge. Images cues such as shading, contour edge, silhouette, location, *etc*. are used to infer geometric relations of captured objects or scenes [Prados and Faugeras, 2005, Prasad et al., 2006, Saxena et al., 2008, Super and Bovik, 1995]. Depending on the recovery goal and the input images, smoothness, geometric and semantic relations, volume/area, *etc*. are usually chosen as priors in fixed form, or gained by learning. Some previous work focused on 3D geometry recovery for curved objects or piecewise planer objects. However user interactions are usually required to indicate silhouettes and the inputs are limited to certain classes [Oswald et al., 2009, Töppe et al., 2010, Zhang et al., 2001].

With such assumptions and priors, unfortunately, the inferred 3D geometry can only approximate large structures of an image, fine-scaled geometries are rarely recovered. With the recent low-cost depth sensors like Kinect, RGB-D data can be captured much easier, boosting the development of learning-based approaches. Especially with the development of convolutional neural networks(CNNs), the accuracy of dense depth map estimation has been greatly improved [Chen et al., 2016, Eigen and Fergus, 2015, Liu et al., 2015]. However, the estimated depth maps still suffer from details missing and artifacts at fine scales.

Actually realistic 3D recovery should be with rich local detailings, which are critical in high quality reconstruction, object recognition, shape matching, scene understanding, *etc*. As shown in Fig. 1.1, the left Armadillo and Happy Buddha model in (a) and (b) looks very unrealistic due to the lacking of local details. It becomes greatly vivid when the fine-scaled local features are recovered (the right model). There is huge demand for 3D high quality models in film industry, visual reality and computer games. It would save a lot of time and manual work if we can automatically recover fine-scaled 3D geometry from single RGB images, such as wrinkles for facial animation and high quality 3D contents for indoor scene modeling. Another important advantage of fine-scaled details recovery is that we can extract meaningful local descriptor from the detailed geometry, which could be widely used in object recognition, correspondence and shape registration, robot localization and navigation. Other tasks like image novel view synthesis and re-lighting both require fine-scaled geometry recovery, as the quality of local shading is highly dependent with accurate surface normals.

Fine-scaled geometry recovery from single RGB images has some common issues. First, rather than estimating the main structure or global layout, it is much more difficult to

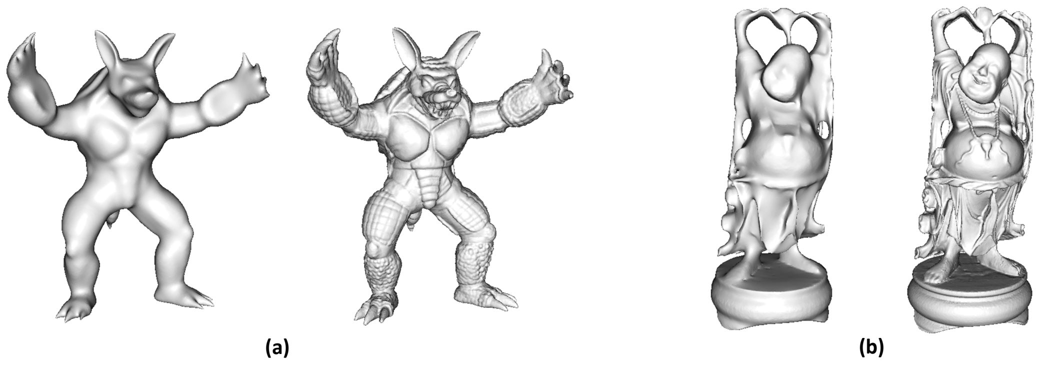


Figure 1.1: *The left model in (a) and (b) is lack of local details; and the right model has rich local geometry features. The Armadillo and Happy Buddha model are taken from the Stanford 3D scanning repository*

recover fine-scaled details from single view. We need to explore the image information from local regions and build a mapping to real 3D geometry at fine scale. Second, it requires high resolution representations, including triangle meshes, depth maps, voxels, *etc*. Such high resolution outputs always bring heavy computation and memory cost on CPU or GPU. Therefore, we need to develop elegant methods to reduce the algorithm complexity, especially for real-time applications like facial animation. Third, for learning approaches image and high quality geometry pairs are required to train a predictor to infer fine-scaled geometry from single images. However, high quality training data with corresponding RGB image is not easy to collect on a large scale. Usually we have to augment the training data by deforming the 3D geometry and applying the same operations on images.

On the other hand, the fine-scaled 3D geometry of different categories usually has different features, styles and requirements , which might need different strategies to solve the problems respectively. For example, the wrinkles on human faces have their own geometry characteristics which can reinforce the perception of the complex emotions in the facial performance. Indoor scene images always contain multiple objects with heavy occlusions and large variations on texture and structure, which require more general algorithms to handle such a challenging problem. While man-made object like chairs, tables, planes, *etc*., are always highly structured (hierarchical decompositions and nested symmetries). Hence the structural information is more critical in 3D geometry recovery of man-made artifacts.

In this thesis, we aim at fine-scaled 3D geometry recovery from single RGB images for three categories: facial wrinkles, indoor scenes and man-made objects. Each category has its own particular features, demands and also challenges, which will be discussed in Section 1.1, "Challenges".

1.1 Challenges

The most difficulty in inferring 3D geometry from single RGB images lies in the fact that this problem is inherently ill-posed. The process of 2D image generation is not invertible and we can not retrieve exact 3D coordinates from a single image. The problem become even more difficult when we target to recover not only the main 3D structures but also the local fine-scale details. For each individual task, we briefly introduce the major challenges and the most related work :

Facial wrinkle recovery. Prior systems for high quality facial modeling usually run in controlled laboratory environments by professional users with high-cost devices, like laser scanning, multi-view stereo[Beeler et al., 2010, 2011]. Some sophisticated deformation methods also have been applied to the fine-scaled wrinkle synthesis [Bradley et al., 2010]. However, they are generally hard to be accessed by common users, which might hinder their widespread applications. When only monocular facial images are available, the problem

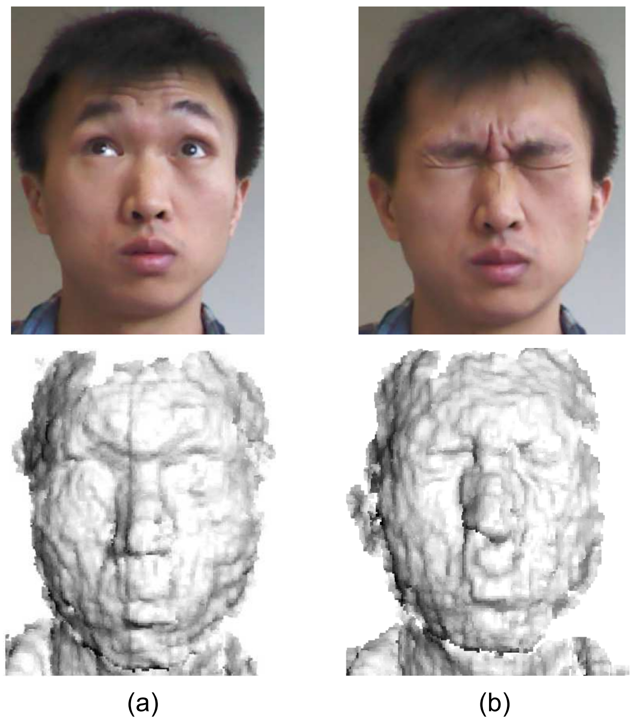


Figure 1.2: *Due to low resolution of Kinect depth image, it is not possible to capture facial details like wrinkles. The only way to recover wrinkles is to explore the wrinkle information in RGB images.*

become even much more difficult. [Blanz and Vetter, 1999a, Cao et al., 2013] could recover 3D faces from single images based on a morphable model, however facial details are greatly missing.

Due to the complex operations in laboratory and data privacy, we can only find very limited online public face data with high quality facial details. Therefore, some deep learning methods which require large scale training data might not work for this problem. On the other hand, as wrinkles are highly dependent with facial expressions, the appearances should be consistent during the facial animation, and be robust to skin color and luminance change. We aim at recover wrinkle details for RGB-Depth images from Kinect. As shown in Fig. 1.2, facial details are totally missing in the point cloud converted from corresponding depth image, we can only infer the fine-scaled wrinkles from image cues.

Depth map estimation for indoor scenes. Inferring the underlying depth from single RGB indoor images is non-trivial problem. As shown in Fig. 1.3, indoor scenes always have large textural and structural variations, heavy object occlusions and rich geometric details, all of which make accurate depth estimation very challenging.

Before RGB-D images were readily available, previous works usually construct simple 3D models from the image labels [Hoiem et al., 2005] or estimate mean depth values from image structure [Torralba and Oliva, 2002]. Once RGB-D data could be collected from laser or depth cameras on a large scale, it became feasible to apply learning-based approaches for dense depth estimation [Karsch et al., 2012, Liu et al., 2014, Saxena et al., 2005, 2008, Zhuo et al., 2015]. Nevertheless, with only hand-crafted features, the inferred depth maps can only approximate the global layout of an image.

In recent years, deep learning methods have demonstrated their strength in a multitude of vision tasks, including monocular depth estimation. Depth accuracy on a per-pixel basis has improved significantly with elegantly designed convolutional neural networks(CNNs) [Bo et al., 2015, Eigen et al., 2014, Liu et al., 2015, Wang et al., 2015] . Rather than yielding over-smoothed approximations which estimate only the coarse depth of large structures such as walls and ceilings, state-of-the-art multi-scale networks [Chen et al., 2016, Eigen and Fergus, 2015] can capture finer-scale items such as furniture and home accessories. Despite the impressive evaluation scores of recent works, however, it still suffer from distortions and artifacts at finer scales, which are especially prominent when projected into 3D.

3D shape recovery for man-made objects. Man-made objects are usually highly structured as shown in Fig. 1.4.There are two main challenges for geometry recovery of man-made objects. One is the large structural variations among classes and even in a class, like chairs. It is usually not easy to recover the detailed geometry of different structures. Another challenging is how to preserve the structural relations between components of man-made objects, such as symmetries. People would never design a table with four legs but different length or a chair with only one armrest, which has symmetry broken.

Recent approaches [Girdhar et al., 2016, Wu et al., 2016] predicted volume data from input single RGB images by learning a volumetric fully convolutional network. However, the resolution of 3D grid are limited due to the GPU memory, usually with the size of 32x32x32

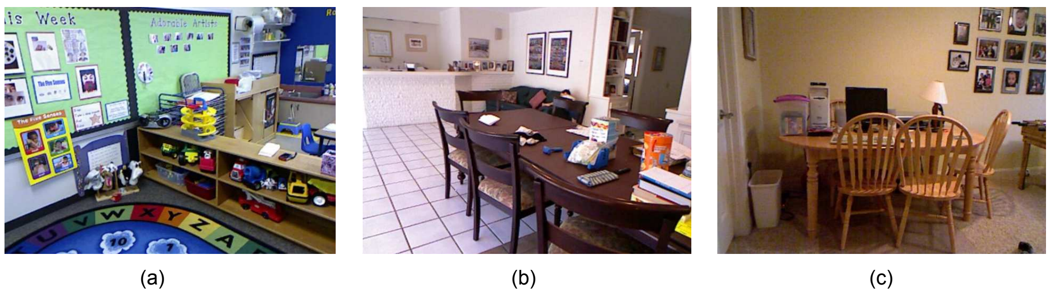


Figure 1.3: *Indoor scenes always have large textural and structural variations, heavy object occlusions and rich geometric details, which bring great challenges for geometry recovery.The three images are taken from NYU dataset [Silberman et al., 2012].*

and 64x64x64. With such a low resolution, detailed geometries are rarely recovered. On the other hand, as the structures like symmetries are not encoded, voxels do no have explicit constraints on each other. Symmetry broken always happens during the volume estimating.

[Xue et al., 2011] and [Liu et al., 2016] explored the symmetry constraints during the depth estimation. [Xue et al., 2011] utilized plane and symmetric lines for depth map recovery, but it is not suitable for non-planar and complex shape parts. [Liu et al., 2016] estimated both symmetry and depth from single images. The symmetry was provided as a constraint to refine the depth map. However, their symmetries was detected in pixel level and very sparse, it is not enough for detailed geometry recovery.

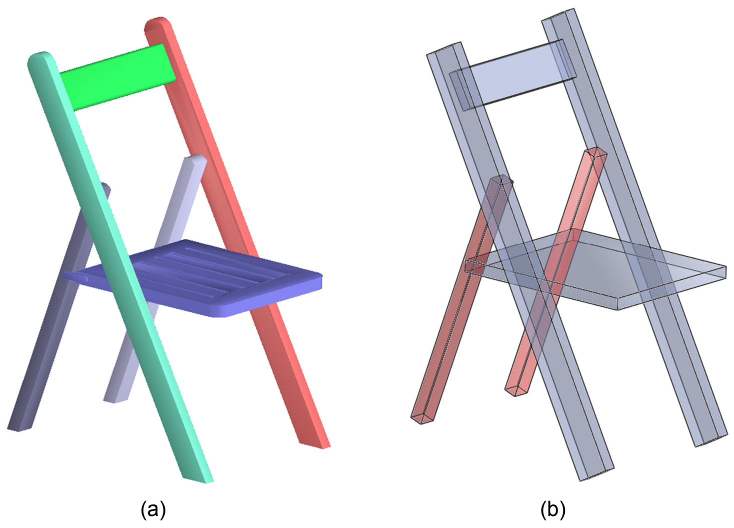


Figure 1.4: *Man-made objects are usually highly structured. (a) is the segmentation of a chair. (b) is the oriented bounding boxes of these components. The two red boxes indicate a reflectional symmetry.*

1.2 Contributions of the Thesis

In this thesis, we investigate the challenging problem of fine-scaled 3D geometry recovery from single RGB images. We aim at three categories: human faces, indoor scenes and man-made objects.In the following, we briefly outline the main contributions of this thesis:

- In chapter 2, “Lightweight Wrinkle Synthesis for 3D Facial Modeling and Animation”, we propose a non-parametric method to synthesize detailed wrinkle geometries and create personalized blendshape models with a single low-cost Microsoft RGB-Depth Kinect camera. As the captured depths by Kinect sensor are usually very noisy and can not provide any information about the facial details like wrinkles, we can only estimate the wrinkles from the corresponding RGB images. To synthesize wrinkles on different 3D personalized blendshapes of various people, we adopt a texture synthesis approach to utilize local geometric exemplars from one high-quality 3D facial model with calibrated texture. The main feature of this method is lightweight, since we only use a very small exemplar database as the source in the texture synthesis and a single RGB-Depth camera. We test our method on a variety of facial performance RGB-Depth videos, the experiments show that our algorithm can recover facial wrinkles which are visually very similar to the input images and provide high-quality clues of the user expressions.

- In chapter 3, "A Two-Streamed Network for Estimating Fine-Scaled Depth Maps from Single RGB Images", we propose a novel set image loss that is defined over multiple related images. This set loss works as a regularizer to minimize the differences among the estimates of the input set images. We find that it makes better use of augmented data and promotes stronger network generalization and less prone to overfitting, further improving the estimation accuracy. Rather than only predicting depth, joint representation of depth and depth gradient estimates are learned with a two-streamed network. We investigate two methods to fuse depth and depth gradient estimates into a more accurate and detailed depth output. One is an end-to-end training via CNNs, the other is a direct optimization exploring the relation between depth and depth gradient. Both methods yield depth maps which, when projected into 3D, have less distortion and are richer with structure and object detailing than competing state-of-the-art.
- In chapter 4 "GRASS: Generative Recursive Autoencoders for Shape Structures", and chapter 5 "3D Shape Recovery from Single RGB Images for Man-made Objects", we investigate how to encode highly structured information for man-made objects and estimate the 3D geometry from single images. Shape structures are represented by layouts of OBBs (oriented bounding boxes for each shape component) and their relations such as symmetry and adjacency. In chapter 4, we propose the first generative neural network model for structured 3D shape representations-GRASS. We introduce a novel RvNN based autoencoder which learns to encode and decode shape structures via discovered symmetry hierarchies, followed by two generative models trained to synthesize box-level symmetry hierarchies and volumetric part geometries, respectively. In chapter 5, we extend this RvNN architecture to estimate 3D shape geometry from single RGB images. Given a single image of man-made object, our method can not only recover the 3D shape structure including symmetry information but also a high resolution surface mesh.

1.3 Structure of the Thesis

In the subsequent chapters, we present our contributions in more detail with experimental results, which show the performance of 3D fine-scaled geometry recovery by our methods with respect to the state-of-the-art.

In chapter 2, we present a lightweight wrinkle synthesis method to enhance the reconstructed 3D face models with detailed wrinkles.

In chapter 3, we propose a fast-to-train two-streamed CNN that predicts both depth and depth gradients, which are then fused together into an accurate and detailed depth map.

In chapter 4, we introduce a novel neural network architecture for encoding and synthesis of 3D shapes, particularly their structures. By developing a recursive neural net (RvNN) based autoencoder, we propose the first generative neural network model for structured 3D shape representations.

The following chapter 5 extends the RvNN architecture to estimate the fine-scaled 3D geometry from single RGB images of man-made objects.

At the end of each chapter, we give a summary and some potential future directions of research on top of the proposed approaches.

Chapter 6 emphasizes the key insights in the thesis, summarizes the experiences we have learned from our investigation and discusses the future work.

Part I

Wrinkles

Chapter 2

Lightweight Wrinkle Synthesis for 3D Facial Modeling and Animation

2.1 Abstract

We present a lightweight non-parametric method to generate wrinkles for 3D facial modeling and animation. The key *lightweight* feature of the method is that it can generate plausible wrinkles using a single low-cost Kinect camera and one high quality 3D face model with details as the example. Our method works in two stages: (1) offline personalized wrinkled blendshape construction. User-specific expressions are recorded using the RGB-Depth camera, and the wrinkles are generated through example-based synthesis of geometric details. (2) online 3D facial performance capturing. These reconstructed expressions are used as blendshapes to capture facial animations in real-time. Experiments on a variety of facial performance videos show that our method can produce plausible results, approximating the wrinkles in an accurate way. Furthermore, our technique is low-cost and convenient for common users.

This work was published in Computer Aided Design, 58(117-122), 2015, [Li et al., 2015].

2.2 Introduction

3D facial modeling is an important research topic in computer graphics, mainly due to its wide applications in game and film industries. Recent progress in 3D facial modeling research advocates the importance of the modeling of fine-scale wrinkles, since it can reinforce the perception of the complex emotions in the facial performance [Bickel et al., 2007, Garrido et al., 2013]. For example, the frown lines are indispensable clues of unpleasant or disapproval emotions.

Laser scanning, multi-view stereo reconstruction and sophisticated deformation methods have been applied to the modeling of highly-detailed 3D facial models with fine-scale wrinkles [Beeler et al., 2010, 2011, Bradley et al., 2010]. However, these systems usually run in controlled laboratory environments by professional users with high-cost devices. They are generally hard to be accessed by common users in their living environments. A recent contribution by Garrido et al. [Garrido et al., 2013] allows users to reconstruct highly-detailed 3D facial models under uncontrolled lighting. While it has significantly simplified the requirements regarding on the capturing conditions, the demand of binocular stereo reconstruction for template face acquisition might still hinder its widespread applications by common users.

In this paper, we propose a non-parametric method to synthesize detailed wrinkle geometries and create personalized blendshape models *with a single low-cost Microsoft RGBD Kinect camera*, which are subsequently used to track RGBD facial performance videos to create 3D facial animations with detailed wrinkles (see Fig. 2.1). We utilize the texture synthesis approach to synthesize wrinkles on the 3D face expression model for various people. The distinctive feature of this method is *lightweight*, since we only use one high-quality 3D facial model with calibrated texture as the source in the texture synthesis and a single RGB-Depth camera. The key observation is that, although the facial wrinkles look different from one person to another, the variation of their local geometry is much less. This implies that it is possible to copy the local geometric patches of one source to synthesize different wrinkles of multiple subjects. Therefore, one can expect to synthesize different kinds of wrinkles in a lightweight fashion.

Our method works in two stages: offline personalized wrinkled blendshape construction and online 3D facial performance capturing. In the offline stage, the user-specific expressions are recorded as blendshapes, and the wrinkles on them are generated through example-based geometric detail synthesis. During the online stage, given an RGB-D video captured by a Kinect camera, the 3D facial animation with detailed wrinkles is reconstructed for each frame as the linear combination of the wrinkled blendshape models.

It should be noted that the synthesized wrinkles can only approximate the real wrinkle geometry. However, experiments show that our lightweight wrinkle synthesis method effectively guarantees the visual plausibility of the reconstructed detailed 3D facial models and animations.

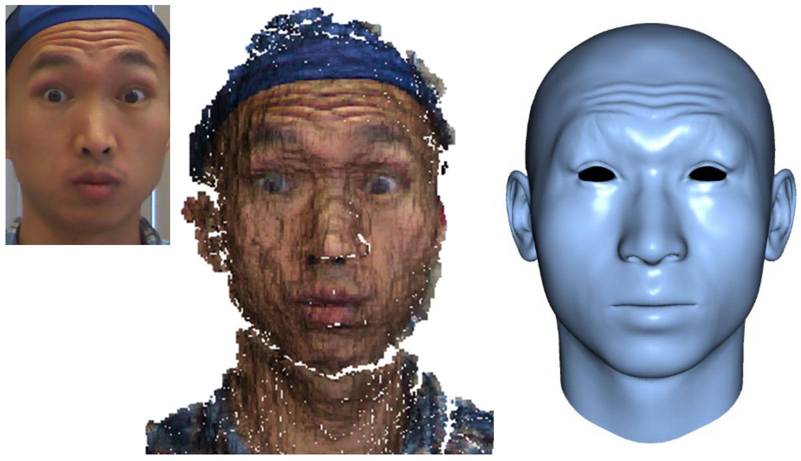


Figure 2.1: *Top-left: Captured RGB image. Middle: The corresponding Kinect raw depth data using the RGB image as its texture. Right: 3D mesh recovered with wrinkle details using our method.*

2.3 Related Work

3D facial modeling: The highly detailed 3D facial model is usually created via state-of-the-art 3D reconstruction techniques, such as laser scanning, structured light, multi-view video based systems.

XYZRGB [XYZRGB, 2012] is a typical laser-based 3D scanning system. It takes over 10 seconds per scan which makes the system more suitable for capturing static expressions. Structured light systems are capable of capturing 3D dynamic faces with salient wrinkles in less than 0.1 second per scan. They are adopted in [Ma et al., 2008, Zhang et al., 2004] to scan facial mesh animation sequences with detailed wrinkles.

In contrast to the mentioned active scanning systems, multi-view video-based facial capture systems are passive. They capture in fast speed, which is important to capture the fast occurring subtleties in the facial expressions. Markers are often used to assist the feature tracking in the facial performance video. Bickel et al. [Bickel et al., 2007] explicitly tracked the wrinkles in the performance video and imposed them on the reconstructed facial model via linear shell deformation to create bulging effects.

Marker-less approaches can ease the burden of putting markers on the human face. Such approaches usually rely on multi-view stereo reconstruction with dense correspondences to recover the detailed wrinkles [Beeler et al., 2010, 2011, Bradley et al., 2010]. A lightweight solution, binocular stereo reconstruction, for marker-less facial animation is proposed in [Garrido et al., 2013] and [Valgaerts et al., 2012]. This technique is robust to illumination change because of the albedo estimation for the captured face.

Real-time facial animation: Facial performance video is a major type of input for real-time facial animation algorithms, since it is easy and fast to capture. The combination of marker tracking and fast deformation algorithms has already been used to produce facial animation in real-time [Vicon, 2012]. Robust face tracking algorithms combined with linear blendshape models are also widely used in single camera based real-time facial animation[Cao et al., 2013, Li et al., 2013, Valgaerts et al., 2012, Weise et al., 2009]. To separately parameterize the different attributes of facial animation, a multi-linear model for face animation transfer is developed in [Vlasic et al., 2005]. However, the detailed wrinkles are usually missing in their reconstructed facial animations.

In [Huang et al., 2011], Huang et al. proposed to scan wrinkled 3D face models to create blendshape models, and successfully reconstruct the detailed wrinkles in the real-time facial animation. Bickel et al. proposed to learn a mapping from the sparse measurement facial skin strain to the wrinkle formulation to synthesize wrinkle geometries [Bickel et al., 2008]. The blendshape models and training examples in these two methods are obtained through laser scanning and structured light, respectively. In contrast, our work allows users to create their own wrinkled blendshape models with a single RGBD camera, which is low cost and easy to access.

Example-based texture synthesis: It aims to synthesize a new texture appearing to be generated by the same process of the given exemplar. The 2D texture synthesis algorithms can be categorized into pixel-level algorithms [Efros and Leung, 1999, Wei and Levoy, 2000], i.e. synthesize textures at each pixel, and patch-level algorithms, which can synthesize textures at a connected group of pixels simultaneously [Efros and Freeman, 2001, Liang et al., 2001]. They are generally faster than pixel-based approaches.

We adapt the texture optimization algorithm in [Kwatra et al., 2005] to synthesize height values used to displace the face mesh vertices to generate wrinkles. While the individual components of our method may resemble previous approaches, but we are not aware of the existing work to synthesize wrinkles with non-parametric texture synthesis.

2.4 Wrinkle Synthesis

To be robust to skin color and luminance change, our wrinkle synthesis runs in the gradient domain. Basically, there are three types of images used for both the input and the example face: the RGB image I, the gradient image G and the height image H, where the gradient image G stores $\partial I/\partial x$ and $\partial I/\partial y$ in its two channels and the height values are stored in H. Given an input image I_f, we first convert it into a gradient image G_f. Afterwards, we synthesize a gradient image G_s using the patches in the example gradient image G_e and compute a height image H_s using the correspondence between the example gradient image G_e and height image H_e. H_s is then used to displace the vertices on the face model to form wrinkles. The algorithm pipeline is shown in Fig. 2.2. In the following, we first describe how to generate the exemplars and then detail the wrinkle synthesis algorithm.

2.4.1 Exemplar Extraction

Our exemplars for texture synthesis are extracted from one high quality face model M_e in the public facial performance sequence ETH man from [Beeler et al., 2011], as shown in (a) of Fig. 2.3. The face model is accompanied with a high quality RGB texture T_e.

While the example gradient image G_e can be computed from T_e easily using the Sobel operator, the computation of height map H_e to encode the wrinkle geometry is not straightforward. To this end, we deform a smooth neutral face, which is also provided in this facial sequence, to match the selected expression model M_e and extract the height values to be stored in the H_e. The algorithm to compute H_e can be separated into two steps:

- 1. The non-rigid shape registration algorithm proposed in [Huang et al., 2008] is used to register a smooth neutral face to the selected expression. To avoid overfitting the detailed features in the deformation, we cluster the vertices according to the vertex normal variation on the neutral mesh. The deformation variables are then reduced to rigid transformations for each cluster, which is similar to Huang et al [Huang et al., 2008]. Since the neutral face and M_e share the same mesh connectivity, we can set up per-vertex correspondence and obtain a smooth registered mesh which successfully captures the overall facial expression but lacks the fine-scale wrinkles, as shown in Fig. 2.3 (b).
- 2. After the registration, the fine-scale wrinkle information of M_e is extracted as its position difference between M_e and the registered smooth face by closest point searching. We compute the height values by projecting the difference vector into the normal direction at the searched closest point on the registered face. With the associated texture T_e, we can store height values into a height image H_e. Note that the correspondence between G_e and H_e can be well established using texture coordinates of M_e.

Since there are many areas in T_e without wrinkle information, we manually crop the wrinkle region on the forehead as the final G_e and H_e. The reduced search area in the exemplar is also beneficial to the performance of the texture synthesis algorithm. However, the wrinkles on the forehead are mostly horizontal. We thus compute new exemplars by rotation to significantly enlarge the wrinkle geometry variations. For example, we could rotate the face by 90 degree and obtain vertical wrinkle exemplars. In our experiments, we rotate the forehead wrinkles by $30°, 60°, 90°, 120° and 150°$. Combined with the original forehead wrinkles, we collect six exemplars in total to be used in the wrinkle synthesis algorithm.

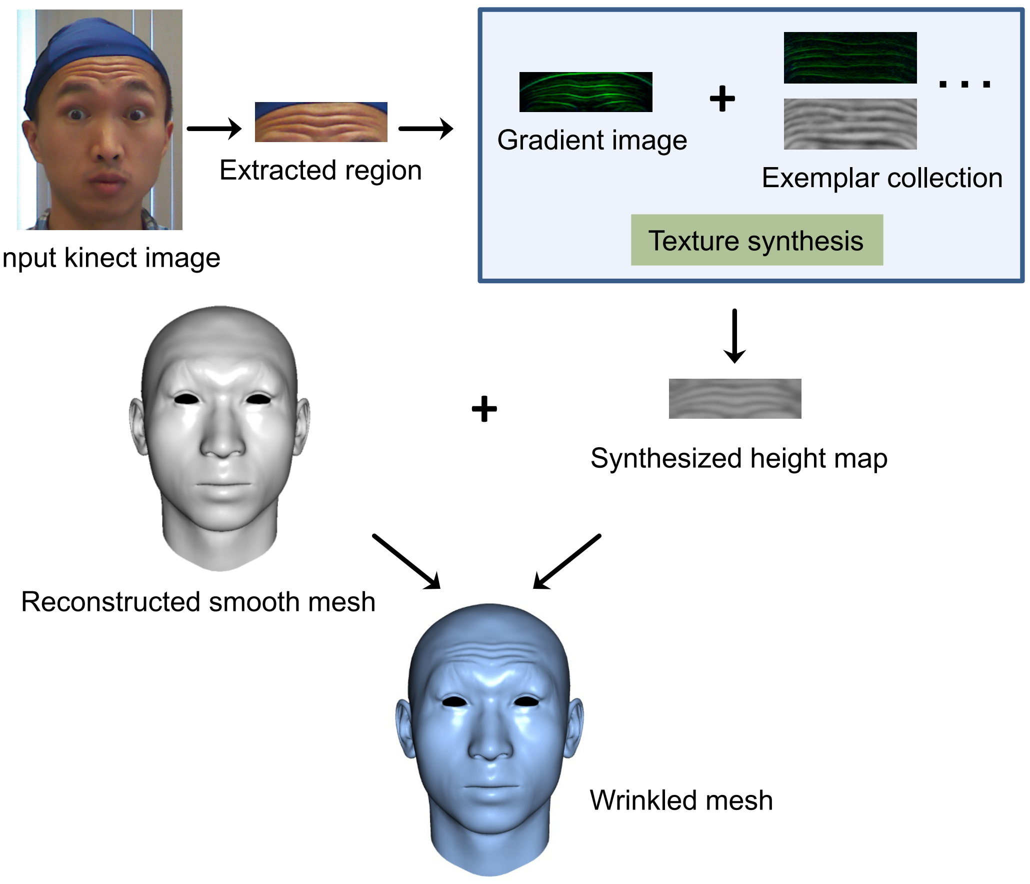


Figure 2.2: *The pipeline of our offline wrinkle synthesis method.*

2.4.2 Salient Wrinkle Region Detection

Before we perform the texture synthesis algorithm, we detect wrinkle edges from the input image. The texture synthesis is thus applied in a significantly smaller region to boost the performance. It also avoids adding unnecessary height displacement on non-wrinkle regions.

The detection is based on the observation that the gradient values have different signs on different sides of a wrinkle boundary. Specifically, in the implementation, we detect edges along both x and y directions. In x direction, for a pixel $p(x, y)$, if the gradient values in $q_1(x-1, y)$ and $q_2(x+1, y)$ have different signs and their difference is larger than a user-specified threshold ϵ, we mark this point as an edge point. We perform this operation along the y direction as well. From our experiments, we found that this strategy is more robust for

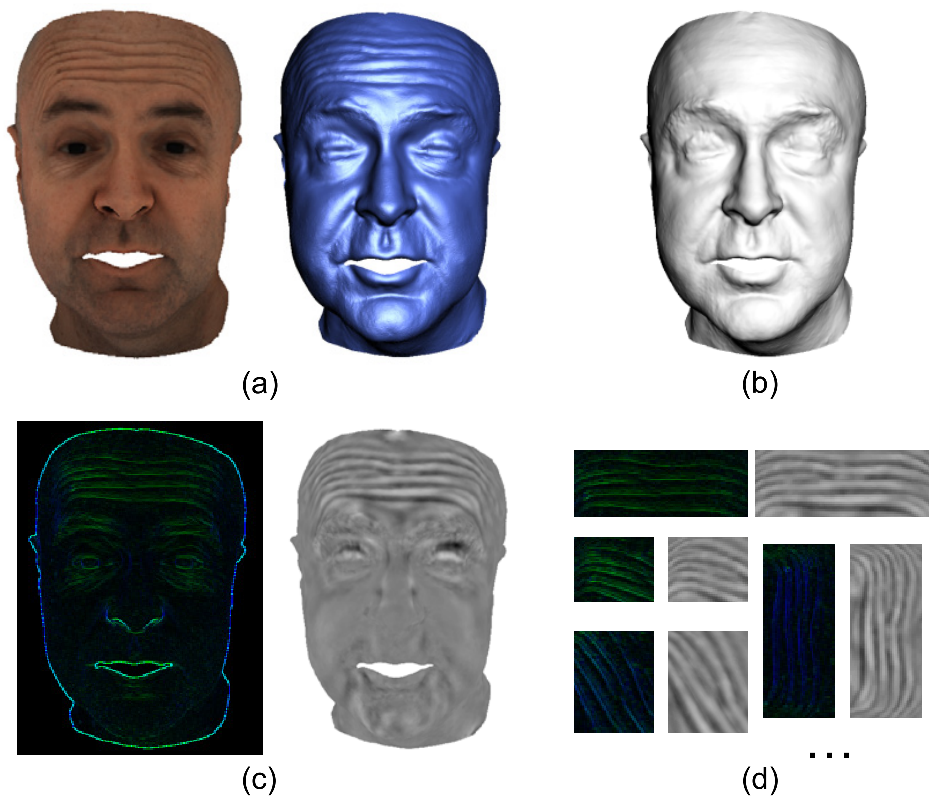


Figure 2.3: *(a) shows the texture T_e and 3D model M_e of the selected expression, (b) is the registered smooth mesh. (c) shows the corresponding gradient image G_e and the height map H_e. (d) shows the extracted exemplars.*

wrinkle detection since it captures the characteristics of the buckling geometry of wrinkles.

After the edge point detection, we group them into lines through tracing them based on adjacency. We remove very short lines and the lines around mouth lips, eye brows and nose using the feature points detection in [Saragih et al., 2011], since these lines are usually formed by color or lighting changes on smooth regions. The remaining lines are further clustered to line sets according to spatial adjacency. The 2D bounding box of each line set represents a wrinkle region. For some cases, automatic detection may not work well, for instance, the small wrinkle regions around eye corners. We thus allow user interaction to extract the wrinkle region more accurately. The user interaction is only necessary for small regions around the eyes. We only ask the user to draw several rectangles to indicate the small regions, and it takes no more than 30 seconds.

2.4.3 Texture Synthesis

For texture synthesis, we require the synthesized gradient image G_s locally similar to the example gradient image G_e and globally almost identical to the input gradient image G_f. That is, we hope to use the local patches from G_e to form G_s which are similar to G_f. The synthesized height image H_s is calculated by copying height values from example height image H_e to H_s with the closest neighborhood information between G_e and G_s. However, if we only consider the similarity of the gradient images, the resulting height map might not be smooth, since the similarity in gradient values does not directly imply the smoothness of the synthesized height values. We thus adapt the objective function in the texture optimization technique in [Kwatra et al., 2005] according to:

$$\mathbf{E} = \sum_{p \in G_s} \left\| \mathbf{g}_p - \mathbf{g}'_p \right\|^2 + \lambda \left\| \mathbf{h}_p - \mathbf{h}_q \right\|^2 \tag{2.1}$$

where the first term measures the similarity between G_s and G_e. $\mathbf{g}_p$ is a vector of stacked gradient values from a small size window centered at p in G_s, and $\mathbf{g}'_p$ denotes a stacked vector of gradient values at the same pixel p from G_e. The second term measures the similarity between the formed height images H_s and H_e locally, where $\mathbf{h}_p$ is the vector of stacked height values at pixel p from H_s, and $\mathbf{h}_q$ represents the vector of stacked height values at pixel q from H_e. q indicates the closest pixel in G_e to the pixel p found in the texture optimization procedure using their neighborhood information.

Texture optimization can be achieved iteratively with an algorithm similar to Expectation-Maximization (EM) [Kwatra et al., 2005]. The key difference of our optimization algorithm is to set the initial gradient image directly as G_f. In the E step, our algorithm calculates the average gradient and the height value at each pixel p according to the neighborhood information of its closest pixel q in the example gradient G_e and the height image H_e.

For the M step, where the closest neighborhood is searched in the example, we fix the gradient values at each input pixel same to the G_f. This prevents the algorithm from altering the synthesized gradient image iteratively, which will lead to a severe drift from the input gradient image in the final result. Specifically, we search the nearest neighborhood in G_e and H_e using the combined gradient and height distance as defined in Eq. 2.1, but using the input gradient values and the synthesized height values. As the scale of the height values in H_e (we use *mm* as unit) is different from the gradient values, we weight the channel H by λ (usually we set λ to 10) to keep a balance between the spatial coherence of the height map and the visual appearance similarity with G_f.

Wrinkle geometry generation: We then displace the vertices on the 3D face expression model along their normals to form wrinkle details, according to the synthesized height image H_e. Note that we only keep the height values in the neighborhood around the detected wrinkle regions (see section 3.2) and set others to be zero. In our experiments, the neighborhood size is set to 5 pixels. After displacing the vertex, we apply Laplacian smooth operations to further improve the mesh quality. It also constructs smooth transitions between the wrinkled and non-wrinked regions.

2.5 Application to Real-time Facial Animation

Our wrinkle synthesis method allows users to use a low-cost Microsoft Kinect to generate personalized blendshape models, which is also a linear model to reinforce the captured facial animation with detailed wrinkles.

In our implementation, we usually capture twenty expressions for each user, such as *neutral pose, smile, brow lower, brow raiser, jaw left, jaw right, jaw forward, mouth left, mouth right, mouth open, cheek blowing, lip funnel, lip pucker, grin and chin raiser*. More complex expressions are required to be captured for the user if higher quality is demanded. While the number of captured blendshape models is less than the number required in FACS system [Ekman and Friesen, 1978], we found the captured user-specific expressions in the blendshape model are sufficient to reconstruct the facial animation with detailed wrinkles.

As the depth data from Kinect camera is noisy, we adopt a method similar to [Weise et al., 2011] to create high-quality expression models. The user needs to rotate her/his head slightly while keeping the expression unchanged in front of the camera. The captured multiple depth images are registered into the first scan by fast ICP, and finally merged into one smooth point cloud [Weise et al., 2011]. The first scan is required to be of front view, and we also captured its corresponding RGB image as the input of our wrinkle synthesis algorithm. We then apply the morphable face model [Blanz and Vetter, 1999a] to the point cloud data captured for the neutral pose, resulting in a smooth 3D neutral face mesh. This neutral mesh is used as template and warped to each captured expressions using the non-rigid registration approach of [Huang et al., 2008]. These reconstructed expression meshes share the same topology.

2.5.1 Online Facial Performance Capture

The reconstructed blendshape models can be directly used in real-time performance-driven facial animation. The overall procedure consists of two steps: rigid motion tracking to compute the global rigid transformation of the head and non-rigid motion tracking to reconstruct the local deformations at each expression.

For rigid motion tracking, we utilize the fast projective variant of ICP [Rusinkiewicz and Levoy, 2001] based on point-to-plane constraints. However, due to the noisy Kinect data, the resulting rigid tracking is not temporally smooth. To filter out high frequency flickering, we first smooth the raw data from Kinect temporally, i.e. a weighted average on the depth map for k (default 5) consecutive frames. We also filter the computed rotations (represented by quaternion) and translations to further improve the tracking quality [Weise et al., 2011].

The purpose of non-rigid tracking is to determine the weights of the blendshape models and reconstruct the user expressions in each frame, which can be formulated into a non-negative least squares problem:

$$\arg\min_{\alpha_i} ||F(\alpha_1 ... \alpha_n)^t - f_i||^2 \tag{2.2}$$

where $F(\alpha_1 ... \alpha_n)^t$ denotes the features on the shape determined by current weights $\{\alpha_i\}$, and f_i are the corresponding captured features. These features could be 3D sample points or 2D sparse features detected from RGB images. Finally, the tracked wrinkled mesh could be recovered by:

$$\mathbf{E} = \mathbf{N} + \sum_i \alpha_i B_i \tag{2.3}$$

where $\mathbf{N}$ represents the 3D neutral model and B_i the i-th blendshape model.

2.6 Results

In this section, we validate our method in the generation of 3D facial models with detailed wrinkles and real-time facial animations of various people using the Microsoft Kinect camera. The current facial performance capture system runs in real-time on a machine with Intel Core 2.66GHz Quad CPU and GeForce GTX 580 graphics card. The average computational time of the wrinkle synthesis for each captured expression is around 10 seconds, and the cost of online facial performance with wrinkles is about 20 ms for one frame in average.

Comparison with multi-view stereo: Fig. 2.4 illustrates a comparison between our results and the passive facial capture method [Bradley et al., 2010]. Although the reconstructed wrinkle details from [Bradley et al., 2010] are smoother than our synthesized wrinkle geometry, our synthesis results are visually very similar and provide high-quality clues of the input user expressions.

Testing on Kinect data: We test our method on various captured RGB-D images and videos from different people to demonstrate the ability of our system to produce plausible 3D facial performances with detailed wrinkles.

In Fig. 2.5 (a) are the wrinkle synthesis results on a few expressions captured the Kinect sensor. The first row is the RGB images, the second row lists the meshes before wrinkle details were added and the last row shows the corresponding wrinkled mesh. Note the neutral pose is shown in the left column. In Fig. 2.5 (b) are more examples of face models with synthesized wrinkles using our method. Fig. 2.6 shows some sample frames from online facial

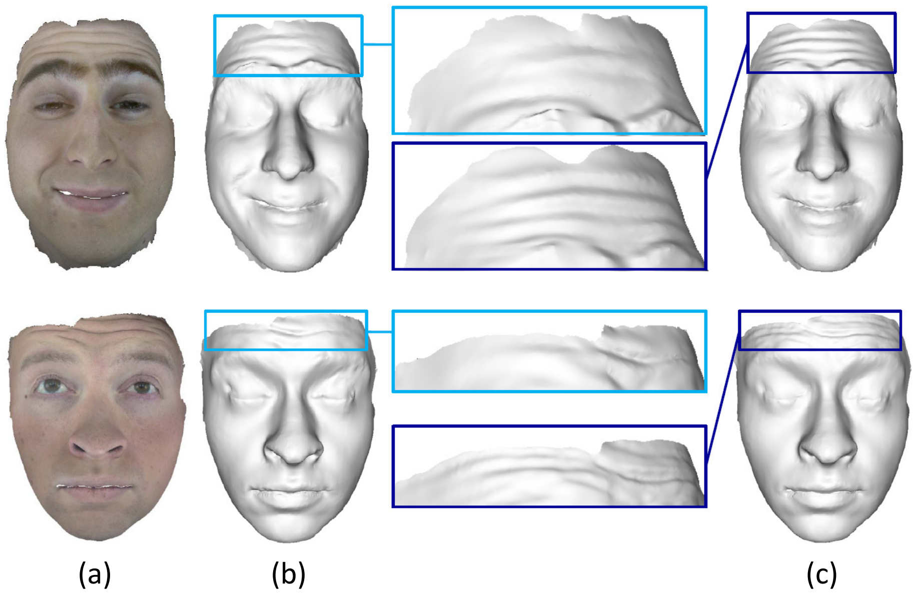


Figure 2.4: *Comparison with Bradley et al [Bradley et al., 2010]. (a) The original captured images. (b) The results from [Bradley et al., 2010]. (c) Wrinkle details generated by our method.*

performance with well reconstructed wrinkle details. For the left part, from top to bottom are RGB images, Kinect point clouds and the reconstructed facial models with detailed wrinkles. The close-up view in the right column shows the quality of synthesized wrinkles from our system. As our wrinkle generation depends on the coefficients of blendshapes, it is robust to light changing, hair occlusion and head rotation during online tracking.

Limitation: Our wrinkle synthesis algorithm now is limited to front view facial images, and it might be influenced by shadows cast by hairs on the face. We plan to explore facial image normalization and shadow removal algorithms to further improve the range of facial images that can be handled by our wrinkle synthesis algorithm.

2.7 Conclusions

We have presented a lightweight wrinkle synthesis method to enhance the reconstructed 3D face models with detailed wrinkles. Our method is lightweight, because we only use a single Microsoft Kinect camera plus one selected, high quality wrinkled 3D face model to generate all the results in this paper.

In future, we plan to enlarge the database so as to incorporate different styles of wrinkle geometry images from people of different ages and races. In wrinkle synthesis, images similar to the capture data will be first retrieved for the wrinkle synthesis. This would lead to higher quality of synthesized results with more enriched expressions captured by our system. We would also like to explore the fast texture synthesis method to reduce the wrinkle generation time.

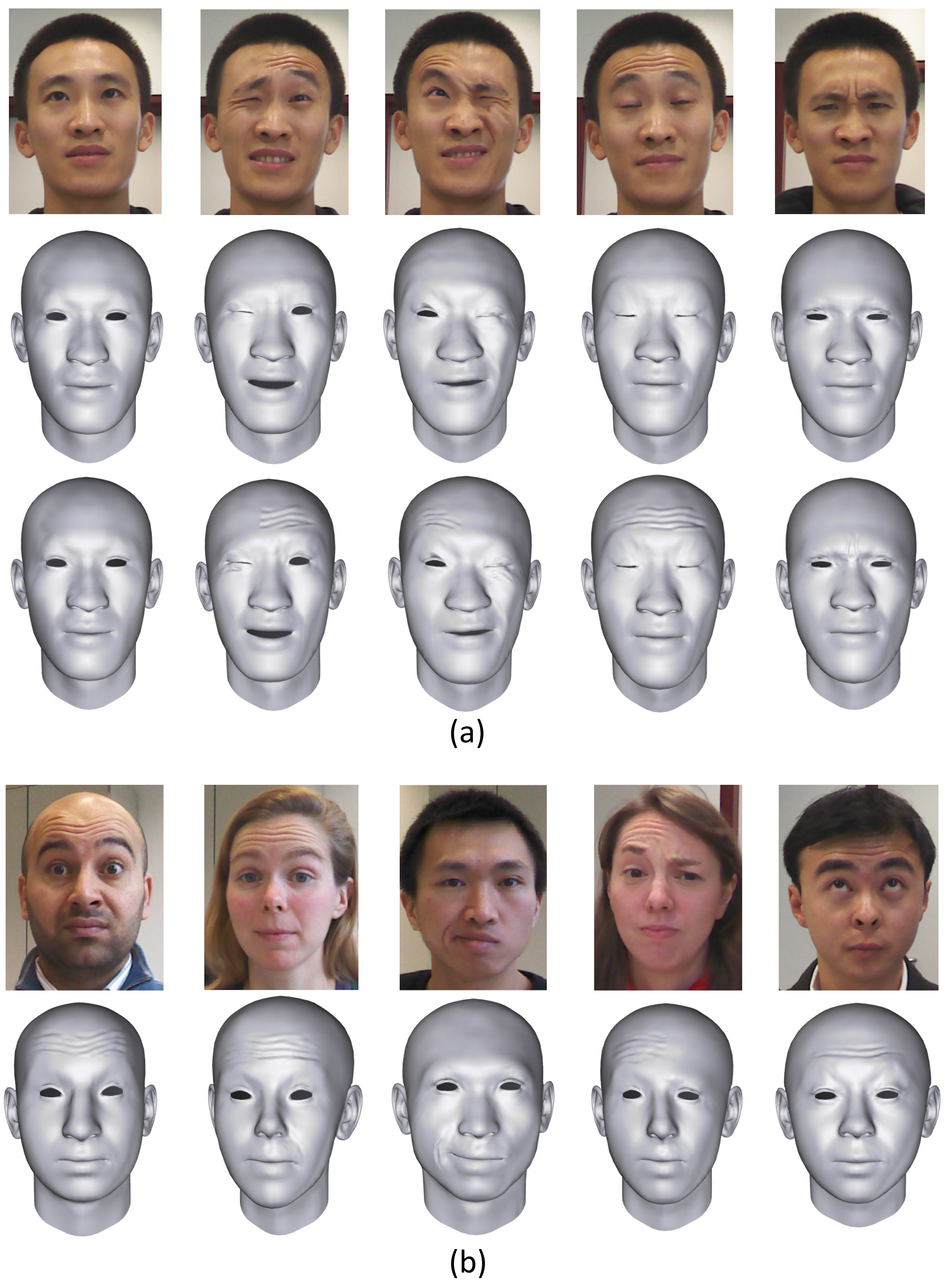


Figure 2.5: *Wrinkle synthesis results using our method.*

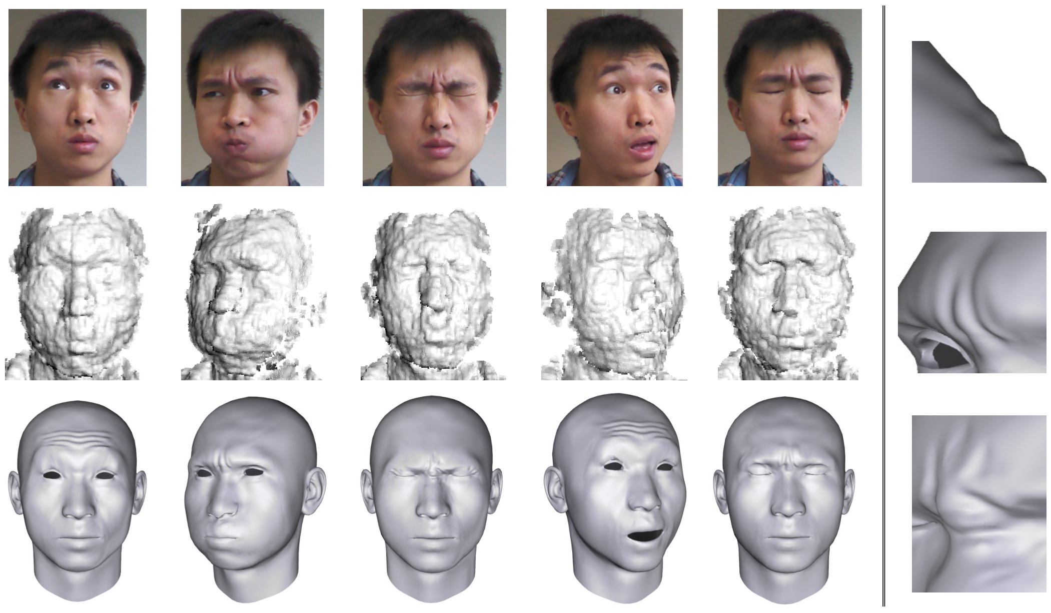


Figure 2.6: *Online facial performance with recovered wrinkle details.*

Part II

Indoor scenes

Chapter 3

A Two-Streamed Network for Estimating Fine-Scaled Depth Maps from Single RGB Images

3.1 Abstract

Estimating depth from a single RGB image is an ill-posed and inherently ambiguous problem. State-of-the-art deep learning methods can now estimate accurate 2D depth maps, but when the maps are projected into 3D, they lack local detail and are often highly distorted. We propose a fast-to-train two-streamed CNN that predicts depth and depth gradients, which are then fused together into an accurate and detailed depth map. We also define a novel set loss over multiple images; by regularizing the estimation between a common set of images, the network is less prone to over-fitting and achieves better accuracy than competing methods. Experiments on the NYU Depth v2 dataset shows that our depth predictions are competitive with state-of-the-art and lead to faithful 3D projections.

This work was published in International Conference on Computer Vision (ICCV), 2017, [Li et al., 2017a]

3.2 Introduction

Estimating depth for common indoor scenes from monocular RGB images has widespread applications in scene understanding, depth-aware image editing or re-rendering, 3D modelling, robotics, *etc*. Given a single RGB image as input, the goal is to predict a dense depth map for each pixel. Inferring the underlying depth is an ill-posed and inherently ambiguous problem. In particular, indoor scenes have large texture and structural variations, heavy object occlusions and rich geometric detailing, all of which contributes to the difficulty of accurate depth estimation.

The use of convolutional neural networks (CNNs) has greatly improved the accuracy of depth estimation techniques [Bo et al., 2015, Eigen and Fergus, 2015, Garg et al., 2016, Kim et al., 2016, Laina et al., 2016, Liu et al., 2015, Roy and Todorovic, 2015, Wang et al., 2015]. Rather than coarsely approximating the depth of large structures such as walls and ceilings, state-of-the-art networks [Eigen and Fergus, 2015, Laina et al., 2016] benefit from using pre-trained CNNs and can capture fine-scaled items such as furniture and home accessories. The pinnacle of success for depth estimation is the ability to generate realistic and accurate 3D scene reconstructions from the estimated depths. Faithful reconstructions should be rich with local structure; detailing becomes especially important in applications derived from the reconstructions such as object recognition and depth-aware image re-rendering and or editing. Despite the impressive evaluation scores of recent works [Eigen and Fergus, 2015, Laina et al., 2016] however, the estimated depth maps still suffer from artifacts at finer scales and have unsatisfactory alignments between surfaces. These distortions are especially prominent when projected into 3D (see Figure 3.1).

Other CNN-based end-to-end applications such as semantic segmentation [Chen et al., 2016, Long et al., 2015] and normal estimation [Bansal et al., 2016, Eigen and Fergus, 2015] face similar challenges of preserving local details. The repeated convolution and pooling operations are critical for capturing the entire image extent, but simultaneously shrink resolution and degrade the detailing. While up-convolution and feature map concatenation strategies [Dosovitskiy et al., 2015, Laina et al., 2016, Long et al., 2015, Ronneberger et al., 2015] have been proposed to improve resolution, output map boundaries often still fail to align with image boundaries. As such, optimization measures like bilateral filtering [Barron and Poole, 2016] or CRFs [Chen et al., 2016] yield further improvements.

It is with the goal of preserving detailing that we motivate our work on depth estimation. We want to benefit from the accuracy of CNNs, but avoid the degradation of resolution and detailing. First, we ensure network accuracy and generalization capability by introducing a novel set image loss. This loss is defined jointly over multiple images, where each image is a transformed version of an original image via standard data augmentation techniques. The set loss considers not only the accuracy of each transformed image's output depth, but also has a regularization term to minimize prediction differences within the set. Adding this regularizer greatly improves the depth accuracy and reduces RMS error by approximately 5%. As similar data augmentation approaches are also used in other end-to-end frameworks, *e.g.* for semantic segmentation and normal estimation, we believe that the benefits of the set loss will carry over to these applications as well.

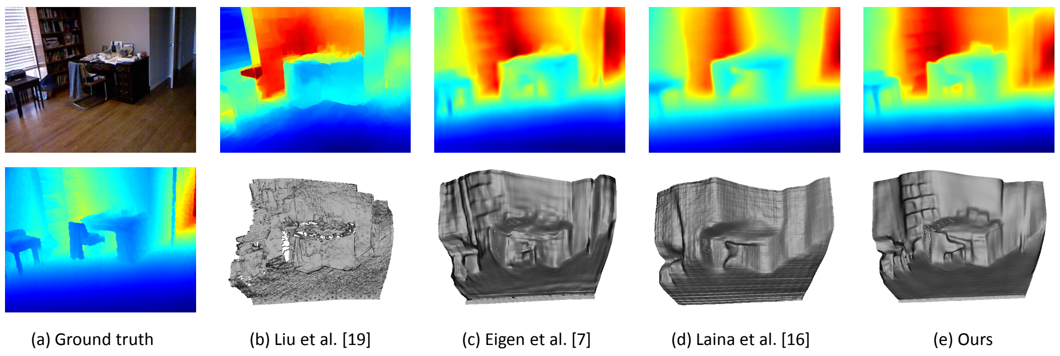


Figure 3.1: *3D projects from estimated depth maps of state-of-the-art methods (b-d), along with ground truth (a). Our estimated depth maps(e) are more accurate than state-of-the-art methods with fine-scaled details. Note that colours values of each depth map are individually scaled.*

We capture scene detailing by considering information contained in depth gradients. We postulate that local structure can be better encoded with first-order derivative terms than the absolute depth values. Perceptually, it is sharp edges and corners which define an object and make it recognizable, rather than (correct) depth values (compare in Figure 3.4). As such, we think it's better to represent a scene with both depth and depth gradients, and propose a fast-to-train two-streamed CNN to regress the depth and depth gradients (see Figure 3.2). In addition, we propose two possibilities for fusing the depth and depth gradients, one via a CNN, to allow for end-to-end training, and one via direct optimization. We summarize our contributions as follows:

- A novel set image loss with a regularizer that minimizes differences in estimated depth of related images; this loss makes better use of augmented data and promotes stronger network generalization, resulting in higher estimation accuracy.
- A joint representation of the 2.5D scene with depth and depth gradients; this representation captures local structures and fine detailing and is learned with a two-streamed network.
- Two methods for fusing depth and depth gradients into a final depth output, one via CNNs for end-to-end training and one via direct optimization; both methods yield depth maps which, when projected into 3D, have less distortion and are richer with structure and object detailing than competing state-of-the-art.

Representing the scene with with both depth and depth gradients is redundant, as one can be derived from the other. We show, however, that this redundancy offers explicit consideration for local detailing that is otherwise lost in the standard Euclidean loss on depth alone and/or with a simple consistency constraint in the loss. Our final depth output is accurate and clean with local detailing, with fewer artifacts than competing methods when projected into 3D.

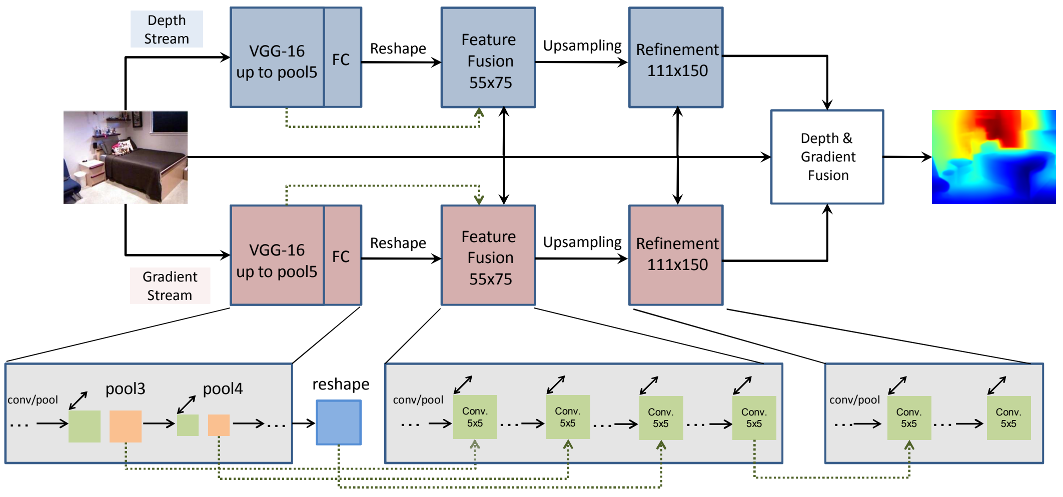


Figure 3.2: *Our two-streamed depth estimation network architecture; the top stream (blue) estimates depth while the bottom (pink) estimates depth gradients. The dotted lines represent features fused from the VGG convolutional layers (see Section 3.4.1). The depth and depth gradients are then combined either via further convolution layers or directly with an optimization enforcing consistency between the depth and depth gradients. Figure is best viewed in colour.*

3.3 Related Work

Depth estimation is a rich field of study and we discuss only the monocular methods. A key strategy in early works for handling depth ambiguity was to use strong assumptions and prior knowledge. For example, Saxena *et al.* [Saxena et al., 2005, 2008] devised a multi-scale MRF, but assumed that all scenes were horizontally aligned with the ground plane. Hoiem *et al.* [Hoiem et al., 2005], instead of predicting depth explicitly, estimated geometric structures for major image regions and composed simple 3D models to represent the scene.

Once RGB-D data could be collected from laser or depth cameras on a large scale, it became feasible to apply data-driven learning-based approaches [Karsch et al., 2012, Liu et al., 2014, Saxena et al., 2005, 2008, Zhuo et al., 2015]. Karsch *et al.* [Karsch et al., 2012] proposed a non-parametric method to transfer depth from aligned exemplars and formulated depth estimation as an optimization problem with smoothness constraints. Liu *et al.* [Liu et al., 2014] modelled image regions as super-pixels and used discrete-continuous optimization for depth estimation and later integrated mid-level region features and global scene layout [Zhuo et al., 2015]. Others tried to improve depth estimations by exploiting semantic labels [Hane et al., 2015, Ladick et al., 2014, Liu et al., 2010]. With hand-crafted features, however, the inferred depth maps are coarse and only approximate the global layout of a scene. Furthermore, they lack the finer details necessary for many applications in computer vision and graphics.

Deep learning has proven to be highly effective for depth estimation [Bo et al., 2015, Eigen and Fergus, 2015, Garg et al., 2016, Kim et al., 2016, Liu et al., 2015, Roy and Todorovic, 2015, Wang et al., 2015]. Liu *et al.* [Liu et al., 2015] combined CNNs and CRFs in a unified framework to learn unary and pairwise potentials with CNNs. They predicted depth at a superpixel level which works well for preserving edges, but when projected to 3D, suffers from distortions and artifacts, as each superpixel region retains the same or very similar depth after an in-painting post-processing.

More recent methods [Chakrabarti et al., 2016, Eigen and Fergus, 2015, Laina et al., 2016] have the harnessed the power of pre-trained CNNs in the form of fully convolutional networks [Long et al., 2015]. The convolutional layers from networks such as VGG [Simonyan and Zisserman, 2015] and ResNet [He et al., 2016] are fine-tuned, while the fully connected layers are re-learned from scratch to encode a spatial feature mapping of the scene. The learned map, however, is at a much lower resolution than the original input. To recover a high-resolution depth image, the feature mapping is then up-sampled [Chakrabarti et al., 2016, Eigen and Fergus, 2015] or passed through up-convolution blocks [Laina et al., 2016]. Our network architecture follows a similar fully convolutional approach, and increases resolution via up-sampling. In addition, we add skip connections between the up-sampling blocks to better leverage intermediate outputs.

3.4 Learning

3.4.1 Network Architecture

Our network architecture, shown in Figure 3.2, follows a two-stream model; one stream regresses depth and the other depth gradients, both from an RGB input image. The two streams follow the same format: an image parsing block, flowed by a feature fusion block and finally a refinement block. The image parsing block consists of the convolutional layers of VGG-16 (up to *pool5*) and two fully connected layers. The output from the second fully connected layer is then reshaped into a $55\times75\times D$ feature map to be passed onto the feature fusion block, where $D=1$ for the depth stream and $D=2$ for the gradient stream. In place of VGG-16, other pre-trained networks can be used as well for the image parsing block, *e.g.* VGG-19 or ResNet.

The feature fusion block consists of one 9×9 convolution and pooling, followed by eight successive 5×5 convolutions without pooling. It takes as input a down-sampled RGB input and then fuses together features from the VGG convolutional layers and the image parsing block output. Specifically, the features maps from VGG *pool3* and *pool4* are fused at the input to the second and fourth convolutional layers respectively, while the output of the image parsing block is fused at the input to the sixth convolutional layer, all with skip layer connections. The skip connections for the VGG features have a 5×5 convolution and a 2x or 4x up-sampling to match the working 55×75 feature map size; the skip connection from the image parsing block is a simple concatenation. As noted by other image-to-image mapping works [Isola et al., 2016, Long et al., 2015, Ronneberger et al., 2015], the skip connections provide a convenient way to share hierarchical information and we find that this also results in much faster network training convergence. The output of the feature fusion block is a coarse 55×75×D depth or depth gradient map.

The refinement block, similar to the feature fusion block, consists of one 9×9 convolution and pooling and five 5×5 convolutions without pooling. It takes as input a down-sampled RGB input and then fuses together a bilinearly up-sampled output of the feature fusion block via a skip connection (concatenation) to the third convolutional layer. The working map size in this block is 111×150, with the output being depth or gradient maps at this higher resolution.

The depth and gradient fusion block brings together the depth and depth gradient estimates from the two separate streams into a single coherent depth estimate. We propose two possibilities, one with convolutional processing in an end-to-end network, and one via a numerical optimization. The two methods are explained in detail in Section 3.4.3. We refer the reader to the supplementary material for specifics on the layers, filter sizes, and learning rates.

3.4.2 Set Image Loss

For many machine learning problems, it has become standard practice to augment the training set with transformed versions of the original training samples. By learning with the augmented set, the resulting classifier or regressor should be more robust to these variations. The loss is applied over some batch of the augmented set, where transformed samples are treated as standard training samples. However, there are strong relations between original and transformed samples that can be further leveraged during training. For example, a sample image that is re-coloured to approximate different lighting conditions should have exactly the same depth estimate as the original. A flipped sample will also have the same depth estimate as the original after un-flipping the output and so on.

Based on this observation, we formulate the set loss as follows. We start by defining the pixel-wise l_2 difference between two depth maps D_1 and D_2:

$$l_2(D_1, D_2) = \frac{1}{n}\sum_p (D_1^p - D_2^p)^2, \tag{3.1}$$

where p is a pixel index, to be summed over n valid depth pixels[1]. A simple error measure comparing an estimated depth map D_i and its corresponding ground truth D_{gt_i} can then be defined as $l_2(D_i, D_{\mathrm{gt}_i})$.

Now consider an image set $\{I, f_1(I), f_2(I)...f_{N-1}(I)\}$, of size N, where I is the original image and f are the data augmentation transformations such as colour adjustment, flipping, rotation, skew, *etc*. For a set of images, the set loss L_{set} is given by

$$L_{\mathrm{set}} = L_{\mathrm{single}} + \lambda \cdot \Omega_{\mathrm{set}}, \tag{3.2}$$

which considers the estimation error of each image in the set as independent samples in L_{single}, along with a regularization term Ω_{set} based on the entire set of transformed images, with λ as a weighting parameter between the two.

More specifically, L_{single} is simply the the estimation error of each (augmented) image considered individually, *i.e.*

$$L_{\mathrm{single}} = \frac{1}{N} \sum_{i=1}^{N} l_2(D_i, D_{\mathrm{gt}_i}). \tag{3.3}$$

The regularization term Ω_{set} is defined as the l_2 difference between estimates within an image set:

$$\Omega_{\mathrm{set}} = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j>i}^{N} l_2(D_i, g_{ij}(D_j)). \tag{3.4}$$

Here, D_i and D_j are the depth estimates of any two images I_i and I_j in the image set. g is a mapping between their transformations, *i.e.* $g_{ij}(\cdot) = f_i(f_j^{-1}(\cdot))$. For all f which are spatial transformations, g should realign the two estimated depth maps into a common reference frame for comparison. For colour transformations, f and f^{-1} are simply identity functions. Ω_{set} acts as a consistency measure which encourages all the depth maps of an image set to be the same after accounting for the transformations. This regularization term has loose connections to tangent propagation [Simard and Denker, 1992], which also has the aim of encouraging invariance to input transformations. Instead of explicit computing the tangent vectors, we transform augmented samples back into a common reference before doing standard back-propagation.

[1]Invalid pixels are parts of the image with missing ground truth depth; such holes are typical of Kinect images and not considered in the loss.

3.4.3 Depth and Depth Gradient Estimation

To learn the network in the depth stream, we use our proposed set loss as defined in Equations 3.2 to 3.4. For learning the network in the depth gradient stream, we use the same formulation, but modify the pixel-wise difference for two gradient maps G_1 and G_2, substituting l_{2g} for l_2:

$$l_{2g}(G_1, G_2) = \frac{1}{n}\sum_p (G_{1x}^p - G_{2x}^p)^2 + (G_{1y}^p - G_{2y}^p)^2, \tag{3.5}$$

where n is the number of valid depth pixels and G_x^p and G_y^p the X and Y gradients at pixel p respectively.

Fusion in an End-to-end Network We propose two possibilities for fusing the outputs of the depth and gradient streams into a final depth output. The first is via a combination block, with the same architecture as the refinement block. It takes as input the RGB image and fuses together the depth estimates and gradient estimates via skip connections (concatenations) as inputs to the third convolutional layer. We use the following combined loss L_{comb} that maintains depth accuracy and gradient consistency:

$$L_{\text{comb}} = L_{\text{set}} + \frac{1}{N}\sum_{i=1} l_{2g}(\nabla D_i, G_{\text{est}_i}), \tag{3.6}$$

where ∇D_i indicates application of the gradient operator on depth map D_i. In this combined loss, the first term L_{set} is based only on depth, while the second term enforces consistency between the gradient of the final depth and estimated gradients with the same l_{2g} pixel-wise difference from Equation 3.5.

Fusion via Optimization Alternatively, as optimization measures have also shown to be highly effective in improving output map detailing [Barron and Poole, 2016, Chen et al., 2016], we directly estimate an optimal depth D^* based on the following minimization:

$$\begin{aligned} D^* = &\operatorname*{argmin}_D \sum_{p=1}^{n} \phi(D^p - D_{\text{est}}^p) + \\ &\omega \sum_p^n [\phi(\nabla_x D^p - G_x^p) + \phi(\nabla_y D^p - G_y^p)], \end{aligned} \tag{3.7}$$

where D_{est} is the estimated depth and G_x and G_y are the estimated gradients in x and y. $\phi(x)$ acts as a robust L_1 measure, *i.e.* $\phi(x) = \sqrt{x^2 + \epsilon}$, $\epsilon = 10^{-4}$; ∇_x, ∇_y are x, y gradient operators on depth D (we use filters $[-1, 0, 1]$, $[-1, 0, 1]^\top$) and p is the pixel index summed over the n valid pixels. We solve for D with iteratively re-weighted least squares using the implementation provided by Karsch [Karsch et al., 2012].

3.4.4 Training Strategy

We apply the same implementation for the networks in both the depth and the gradient streams. With the exception of the VGG convolutional layers, the fully connected layers and all layers in the feature fusion, refinement and combination block are initialized randomly.

The two streams are initially trained individually, each with a two-step procedure. First, the image parsing and feature fusion blocks are trained with a loss on the depth and depth gradients. These blocks are then fixed, while in the refinement blocks are trained in a second step. For each step, the same set loss with the appropriate pixel differences (see Equations 3.2, 3.1, 3.5) is used, albeit with different map resolutions (55×75 after feature fusion, 111×75 after refinement). Training ends here if the optimization-based fusion is applied. Otherwise, for the end-to-end network fusion, the image parsing, feature fusion and refinement blocks are fixed while the fusion block is trained using the combined loss (Equation 3.6). A final fine-tuning is applied to all the blocks of both streams jointly based on this combined loss.

The network is fast to train; only 2 to 3 epochs are required for convergence (see Figure 3.3). During the first 20K iterations, we stabilize training with gradient clipping. For fast convergence, we use a batch size of 1; note that because our loss is defined over an image set, a batch size of 1 is effectively a mini-batch of size N, depending on the number of image transformations used.

The regularization constants λ and ω are set to 1 and 10 respectively. Preliminary experiments showed that different values of λ which controls the extent of the set image regularizer does not affect the resulting accuracy, although a larger λ does slow down network convergence. ω, controlling the extent of the gradients in the gradient optimization, was set by a validation set; a larger ω over-emphasizes artifacts in the gradient estimates and leads to less accurate depth maps.

Method	Base Network	rel	log10	rms	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Karsch *et al.*	-	0.35	0.131	1.2	-	-	-
Liu *et al.*	-	0.335	0.127	1.06	-	-	-
Ladicky *et al.*	-	-	-	-	0.542	0.829	0.941
Li *et al.*	-	0.232	0.094	0.821	0.621	0.886	0.968
Liu *et al.*	-	0.213	0.087	0.759	0.650	0.906	0.976
Eigen *et al.*	VGG-16	0.158	-	0.641	0.769	0.950	0.988
Laina *et al.*	VGG-16	0.194	0.083	0.790	0.629	0.889	0.971
Chakrabarti *et al.*	VGG-19	0.149	-	0.620	0.806	0.958	0.987
Laina *et al.*	ResNet-50	0.127	0.055	0.573	0.811	0.953	0.988
L_{single}, depth only	VGG-16	0.161	0.068	0.640	0.765	0.950	0.988
L_{set}, depth only	VGG-16	0.153	0.065	0.617	0.786	0.954	0.988
L_{set}, depth + bilateral filtering	VGG-16	0.152	0.065	0.621	0.785	0.954	0.988
L_{set}, depth + gradients, end-to-end	VGG-16	0.153	0.064	0.615	0.788	0.954	0.988
L_{set}, depth + gradients, optimization	VGG-16	0.152	0.064	0.611	0.789	0.955	0.988
L_{set}, depth + gradients, optimization	VGG-19	0.146	0.063	0.617	0.795	0.958	0.991
L_{set}, depth + gradients, optimization	ResNet-50	0.143	0.063	0.635	0.788	0.958	0.991

Table 3.1: *Accuracy of depth estimates on the NYU Depth v2 dataset, as compared with state of the art. Smaller values on rel,* $\log_{10}$ *and rms error are better; higher values on* $\delta <$ *threshold are better.*

3.5 Experimentation

3.5.1 Dataset & Evaluation

We use the NYU Depth v2 dataset [Silberman et al., 2012] with the standard scene splits; from the 249 training scenes, we extract ~220k training images. RGB images are down-sampled by half and then cropped to 232×310 to remove blank boundaries after alignment with depth images. Depths are converted to log-scale while gradients are kept in a linear scale.

We evaluate our proposed network and the two fusion methods on the 654 NYU Depth v2 [Silberman et al., 2012] test images. Since our depth output is 111 × 150 and a lower resolution than the original NYUDepth images, we bilinearly up-sample our depth map (4x) and fill in missing borders with a cross-bilateral filter, similar to previous methods [Bo et al., 2015, Eigen and Fergus, 2015, Liu et al., 2015, 2014]. We evaluate our predictions in the valid Kinect depth projection area, using the same measures as previous work:

- mean relative error (rel): $\frac{1}{T}\sum_i^T \frac{|d_i^{gt}-d_i|}{d_i^{gt}}$,
- mean $\log_{10}$ error ($\log_{10}$): $\frac{1}{T}\sum_i^T |\log_{10} d_i^{gt} - \log_{10} d_i|$,
- root mean squared error (rms): $\sqrt{\frac{1}{T}\sum_i^T \left(d_i^{gt} - d_i\right)^2}$,
- thresholded accuracy: percentage of d_i such that $\max\left(d_i^{gt}/d_i, d_i/d_i^{gt}\right) = \delta <$ threshold.

In each measure, d_i^{gt} is the ground-truth depth, d_i the estimated depth, and T the total pixels in all evaluated images. Smaller values on rel, $\log_{10}$ and rms error are better and higher values on percentage(%) δ < threshold are better.

We make a qualitative comparison by projecting the estimated 2D depth maps into a 3D point cloud. The 3D projections are computed using the Kinect camera projection matrix and the resulting point cloud is rendered with lighting. Representative samples are shown in Figure 3.4; the reader is referred to the Supplementary Material for more results over the test set.

3.5.2 Depth Estimation Baselines

The accuracy of our depth estimates is compared with other methods in Table 3.1. We consider as our baseline the accuracy of only the depth stream with a VGG-16 base network, without adding gradients (L_{single}, depth only). This baseline already outperforms [Laina et al., 2016] (VGG-16) and is comparable to [Eigen and Fergus, 2015] (VGG-16). With the set loss, however (L_{set}, depth only), we surpass [Eigen and Fergus, 2015] with more accurate depth estimates, especially in terms of rms-error and thresholded accuracy with $\delta < 1.25$.

Current state-of-the-art results are achieved with fully convolutional approaches [Chakrabarti et al., 2016, Eigen and Fergus, 2015, Laina et al., 2016]. The general trend is that deeper base networks (VGG-19, ResNet-50 vs. VGG-16) leads to higher depth accuracy. We observe a similar trend in our results, though improvements are not always consistent. We achieve some gains with VGG-19 over VGG-16. However, unlike [Laina et al., 2016], we found little gains with ResNet-50.

3.5.3 Fused Depth and Depth Gradients

Fusing depth estimates together with depth gradients achieves similar results as the optimization, both quantitatively and qualitatively. When the depth maps are projected into 3D (see Figure 3.4), there is little difference between the two fusion methods. When comparing with [Laina et al., 2016]'s ResNet-50 results, which are significantly more accurate, one sees that [Laina et al., 2016]'s 3D projections are more distorted. In fact, many structures, such as the shelves in Figure 3.4(a), the sofa in (b) or the pillows in (b,d) are unidentifiable. Furthermore, the entire projected 3D surface seems to suffer from grid-like artifacts, possibly due to their up-projection methodology. On the other hand, the projections of [Chakrabarti et al., 2016] are much cleaner and detailed, even though their method also reports lower accuracy than [Laina et al., 2016]. As such, it is our conclusion that the current numerical evaluation measures are poor indicators of detail preservation. This is expected, since the gains from detailing have little impact on the numerical accuracy measures. Instead, differences are more salient qualitatively, especially in the 3D projection.

Compared to [Eigen and Fergus, 2015], our 3D projections are cleaner and smoother in the appropriate regions, *i.e.* walls and flat surfaces. Corners and edges are better preserved and the resulting scene is richer in detailing with finer local structures. For example, in the cluttered scenes of Figure 3.4(b,c), [Eigen and Fergus, 2015] has heavy artifacts in highly textured areas, *e.g.* the picture on the wall of (b), the windows in (c) and in regions with strong reflections such as the cabinet in (c). Our results are robust to these difficulties and give a more faithful 3D projections of the underlying objects in the scene.

At a first glance, one may think that jointly representing the scene with both depth and depth gradients simply has a smoothing effect. While the fused results are definitely smoother, no smoothing operations, 2D or 3D, can recover non-existent detail. When applying a 2D bilateral filter with 0.1m range and 10×10 spatial Gaussian kernel (L_{set} depth + bilateral filtering), we find little difference in the numerical measures and still some losses in detailing (see Fig.3.4)

3.5.4 Set Loss & Data Augmentation

Our proposed set image loss has a large impact in improving the estimated depth accuracy. For the reported results in Table 3.1, we used three images in the image set: I, flip(I) and colour(I). The flip is around the vertical axis, while the colour operation includes randomly increasing or decreasing brightness, contrast and multiplication with a random RGB value $\gamma \in [0.8, 1.2]^3$. Preliminary experiments showed that adding more operations like rotation and translation did not further improve the performance so we omit them from our training. We speculate that the flip and colouring operations bring the most global variations but leave pinpointing the exact cause for future work.

3.5.5 Training Convergence and Timing

We show the convergence behaviour of our network for the joint training of the image parsing and feature fusion blocks Figure 3.3. Errors decrease faster with a batch size of 1 in comparison to a batch size of 16, and requires only 0.6M gradient steps or 2-3 epochs to converge. For the convergence experiment, we compare the single image loss (batch size 1, 16) with the set image loss as described in Section 3.5.4 and observe that errors are lower with the set loss but convergence still occurs quickly. Note that the fast convergence of our network is not due to the small batch size, but rather the improved architecture with the skip connection. In comparison, the network architecture of [Eigen and Fergus, 2015] requires a total of 2.5M gradient steps to converge (more than 100 epochs) with batchsize 16, but even when trained with a batch size of 1, does not converge so quickly. Training for depth gradient estimates is even faster and converges within one epoch. Overall, our training time

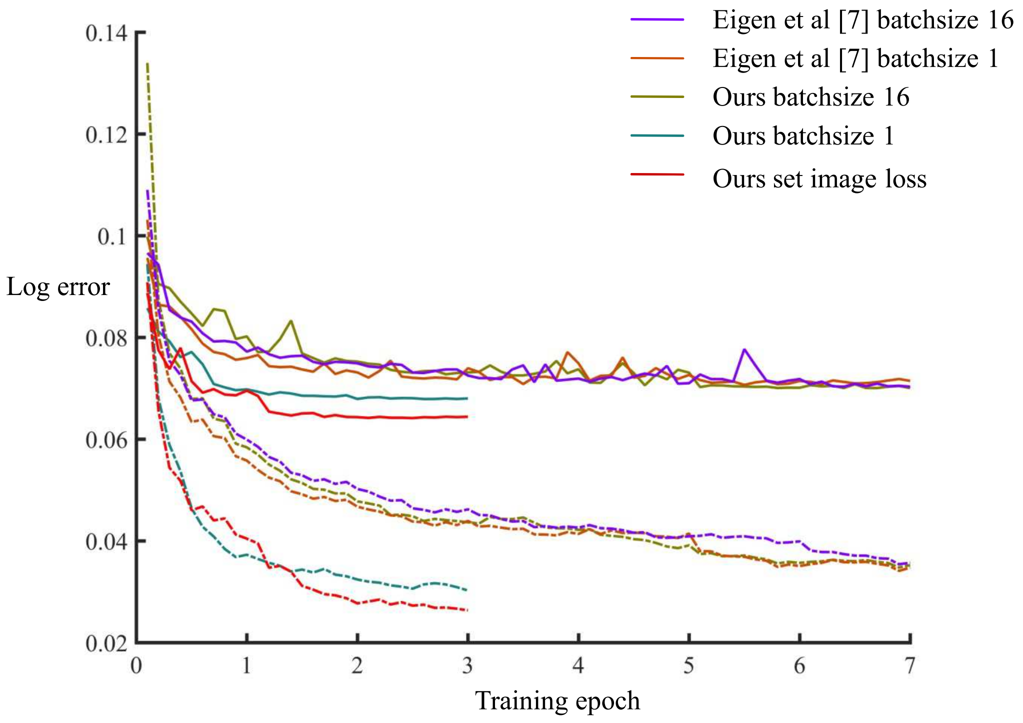


Figure 3.3: *A comparison of the log_{10} training and test errors between our proposed network and [Eigen and Fergus, 2015]. For clarity, we plot the log error at every 0.1 epochs, and show only the first 7 epochs, even though the method of [Eigen and Fergus, 2015] has not yet converged. The dashed lines denote training error, and solid lines denote testing error. For our batch 1 and 16 results, we compare the errors for L_{single} and L_{single}.*

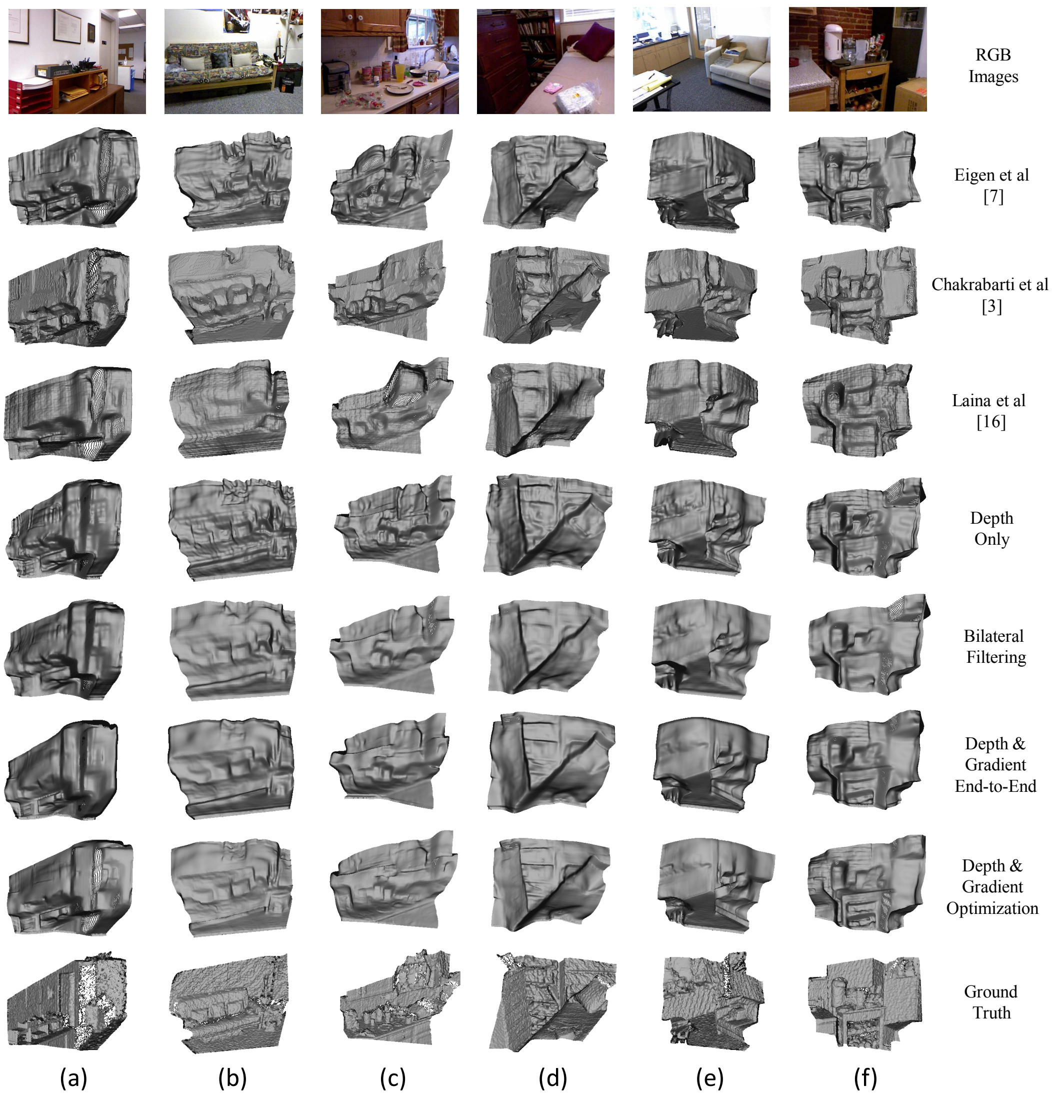
is about 70 hours (50 hours for depth learning and 20 hours for gradient learning) on a single GPU TITAN X.

3.6 Discussion and Conclusion

We have proposed a fast-to-train multi-streamed CNN architecture for accurate depth estimation. To predict accurate and detailed depth maps, we introduced three novel contributions. First, we define a set loss jointly over multiple images. By regularizing the estimation between images in a common set, we achieve better accuracy than previous work. Second, we represent a scene with a joint depth and depth gradient representation, for which we learn with a two-streamed network, to preserve the fine detailing in a scene. Finally, we propose two methods, one CNN-based and one optimization-based, for fusing the depth and gradient estimates into a final depth output. Experiments on the NYU Depth v2 dataset shows that our depth predictions are not only competitive with state-of-the-art but also lead to 3D projections that are more accurate and richer with details.

Looking at our experimental results as well as the results of state-of-the-art methods [Chakrabarti et al., 2016, Eigen and Fergus, 2015, Laina et al., 2016], it becomes clear that the current numerical metrics used for evaluating estimated depth are not always consistent. Such inconsistencies become even more prominent when the depth maps are projected into 3D. Unfortunately, the richness of a scene is often qualified by clean structural detailing which are difficult to capture numerically and which makes it in turn difficult to design appropriate loss or objective functions. Alternatively, one may look to applications using image-estimated depths as input, such as 3D model retrieval or scene-based relighting, though such an indirect evaluation may also introduce other confounding factors.

Our method generates accurate and rich 3D projections, but the outputs are still only 111×150, whereas the original input is 427×561. Like many end-to-end applications, we work at lower-than-original resolutions to trade off the number of network parameters versus the amount of training data. While depth estimation requires no labels, the main bottleneck is the variation in scenes. The training images of NYU Depth v2 are derived from videos of only 249 scenes. The small training dataset size may explain why our set loss with its regularization term has such a strong impact. As larger datasets are introduced [Dai et al., 2017], it may be become feasible to work at higher resolutions. Finally, in the current work, we have addressed only the estimation of depth and depth gradients from an RGB source. It is likely that by combining the task with other estimates such as surface normals and semantic labels, one can further improve the depth estimates.



RMS error	a	b	c	d	e	f
Eigen *et al*. [Eigen and Fergus, 2015]	0.795	0.154	0.377	0.204	0.452	0.253
Charkrabarti *et al*. [Chakrabarti et al., 2016]	0.558	0.236	0.223	0.208	0.400	0.304
Laina *et al*. [Laina et al., 2016]	0.824	0.237	0.306	0.132	0.344	0.159
Depth Only	0.715	0.259	0.246	0.127	0.335	0.202
Bilateral Filtering	0.715	0.259	0.245	0.126	0.334	0.201
Depth & Gradient (End-to-End)	0.719	0.261	0.249	0.130	0.336	0.192
Depth & Gradient (Optimization)	0.715	0.265	0.248	0.124	0.338	0.185

Figure 3.4: *Comparison of example 3D projections with corresponding RMS errors using VGG-16 as the base network. Please see supplementary materials for more results. The four variations of our proposed method all report similar RMS values, though differences are more noticeable in the 3D projection. From examples (b) and (f), where [Laina et al., 2016] reports the lower RMS, one can see that a numerical measure such as RMS is not always indicative of estimated depth quality.*

Part III

Man-made Objects

Chapter 4

GRASS: Generative Recursive Autoencoders for Shape Structures

4.1 Abstract

We introduce a novel neural network architecture for *encoding* and *synthesis* of 3D shapes, particularly their *structures*. Our key insight is that 3D shapes are effectively characterized by their *hierarchical* organization of parts, which reflects fundamental intra-shape relationships such as adjacency and symmetry. We develop a *recursive* neural net (RvNN) based autoencoder to map a flat, unlabeled, arbitrary part layout to a compact code. The code effectively captures hierarchical structures of man-made 3D objects of varying structural complexities despite being fixed-dimensional: an associated decoder maps a code back to a full hierarchy. The learned bidirectional mapping is further tuned using an adversarial setup to yield a generative model of plausible structures, from which novel structures can be sampled. Finally, our structure synthesis framework is augmented by a second trained module that produces fine-grained part geometry, conditioned on global and local structural context, leading to a full generative pipeline for 3D shapes. We demonstrate that without supervision, our network learns meaningful structural hierarchies adhering to perceptual grouping principles, produces compact codes which enable applications such as shape classification and partial matching, and supports shape synthesis and interpolation with significant variations in topology and geometry.

This work was published in ACM Transactions on Graphics (Special Issue of SIGGRAPH), Vol. 36, No. 4, Article 52, 2017. [Li et al., 2017b]

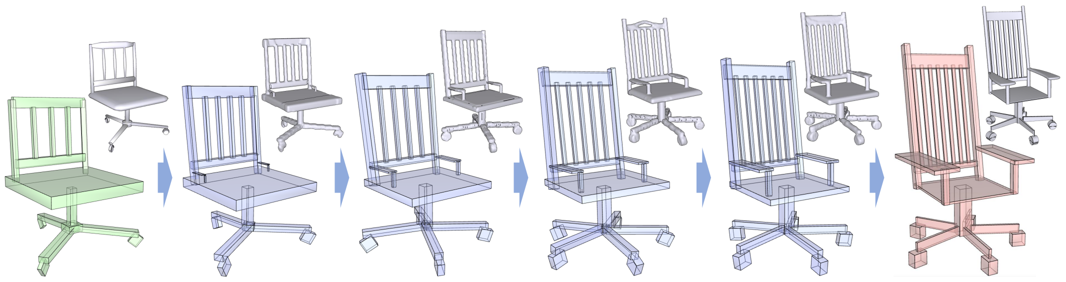


Figure 4.1: *We develop GRASS, a Generative Recursive Autoencoder for Shape Structures, which enables* structural *blending between two 3D shapes. Note the discrete blending of translational symmetries (slats on the chair backs) and rotational symmetries (the swivel legs). GRASS encodes and synthesizes box structures (bottom) and part geometries (top) separately. The blending is performed on fixed-length codes learned by the unsupervised autoencoder, without any form of part correspondences, given or computed.*

4.2 Introduction

Recent progress on training neural networks for image [van den Oord et al., 2016b] and speech [van den Oord et al., 2016a] synthesis has led many to ask whether a similar success is achievable in learning generative models for 3D shapes. While an image is most naturally viewed as a 2D signal of pixel values and a piece of speech as a sampled 1D audio wave, the question of what is *the* canonical representation for 3D shapes (voxels, surfaces meshes, or multi-view images) may not always yield a consensus answer. Unlike images or sound, a 3D shape does not have a natural parameterization over a regular low-dimensional grid. Further, many 3D shapes, especially of man-made artifacts, are highly structured (e.g. with hierarchical decompositions and nested symmetries), while exhibiting rich structural variations even within the same object class (e.g. consider the variety of chairs). Hence, the stationarity and compositionality assumptions [Henaff et al., 2015] behind the success of most neural nets for *natural* images or speech are no longer applicable.

In this paper, we are interested in learning *generative neural nets* for *structured* shape

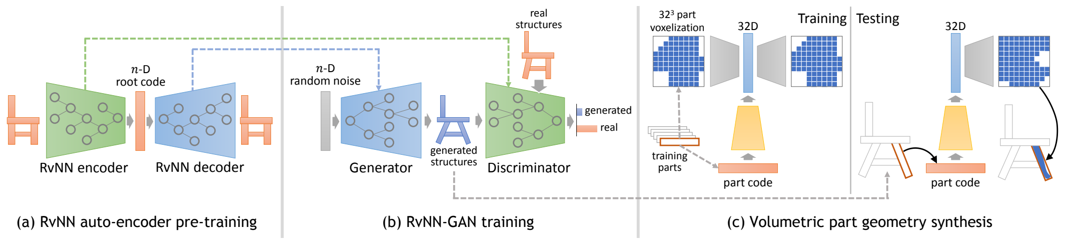


Figure 4.2: *An overview of our pipeline, including the three key stages: (a) pre-training the RvNN autoencoder to obtain root codes for shapes, (b) using a GAN module to learn the actual shape manifold within the code space, and (c) using a second network to convert synthesized OBBs to detailed geometry.*

representations of man-made 3D objects. In general, shape structures are defined by the arrangement of, and relations between, shape parts [Mitra et al., 2013]. Developing neural nets for structured shape representations requires a significant departure from existing works on convolutional neural networks (CNNs) for volumetric [Girdhar et al., 2016, Wu et al., 2016, 2015, Yumer and Mitra, 2016] or view-based [Qi et al., 2016, Sinha et al., 2016, Su et al., 2015] shape representations. These works primarily adapt classical CNN architectures for image analysis. They do not explicitly encode or synthesize part arrangements or relations such as symmetries.

Our goal is to learn a generative neural net for shape structures characterizing an object class, e.g. chairs or candelabras. The main challenges we face are two-fold. The first is how to properly "mix", or *jointly* encode and synthesize (discrete) structure and (continuous) geometry. The second is due to intra-class structural variations. If we treat shape structures as graphs, the foremost question is how to enable a generic neural network to work with graphs of *different* combinatorial structures and sizes. Both challenges are unique to our problem setting and neither has been addressed by networks which take inputs in the form of unstructured, fixed-size, low-dimensional grid data, e.g. images or volumes.

Our key insight is that most shape structures are naturally *hierarchical* and hierarchies can jointly encode structure and geometry. Most importantly, regardless of the variations across shape structures, a coding scheme that recursively contracts hierarchy or tree nodes into their parents attains *unification* at the top — any finite set of structures eventually collapses to root node codes with a possibly large but *fixed length*. We learn a neural network which can recursively encode hierarchies into root codes and invert the process via decoding. Then, by further learning a distribution over the root codes for a class of shapes, new root codes can be generated and decoded to synthesize new structures and shapes in that class.

Specifically, we represent a 3D shape using a *symmetry hierarchy* [Wang et al., 2011b], which defines how parts in the shape are recursively grouped by symmetry and assembled by connectivity. Our neural net architecture, which learns to infer such a hierarchy for a shape in an unsupervised fashion, is inspired by the *recursive neural nets* (RvNN)[1] of Socher et al. [Socher et al., 2012, 2011] developed for text and image understanding. By treating text as a set of words and an image as a set of superpixels, an RvNN learns a parse tree which recursively merges text/image segments. There are two key differences and challenges that come with our work:

- First, the RvNNs of Socher et al. [Socher et al., 2011] always merge two adjacent elements and this is modeled using the same network at every tree node. However, in a symmetry hierarchy, grouping by symmetry and assembly by connectivity are characteristically different merging operations. As well, the network structures at a tree

[1]Note that we are adding the letter 'v' to the acronym RNN, since by now, the term RNN most frequently refers to *recurrent* neural networks.

node must accommodate assembly, reflectional symmetry, and rotational/translational symmetries of varying orders.
- Second, our main goal is to learn a *generative* RvNN, for part-based shape structures that are explicitly represented as discrete structural combinations of geometric entities.

To accomplish these goals, we focus on learning an abstraction of symmetry hierarchies, which are composed of spatial arrangements of oriented bounding boxes (OBBs). Each OBB is defined by a fixed-length code to represent its geometry and these codes sit at the leaves of the hierarchies. Internal nodes of the hierarchies, also characterized by fixed-length codes, encode both the geometry of its child OBBs and their detailed grouping mechanism: whether by connectivity or symmetry.

We pre-train an *unsupervised* RvNN using OBB arrangements endowed with box connectivity and various types of symmetry. Our neural network is an *autoencoder*-based RvNN which recursively assembles or (symmetrically) groups a set of OBBs into a fixed-length root code and then decodes the root to reconstruct the input; see Figure 4.2(a). The network comprises two types of nodes: one to handle assembly of connected parts, and one to handle symmetry grouping. Each merging operation takes two or more OBBs as input. Our RvNN learns how to best organize a shape structure into a symmetry hierarchy to arrive at a compact and minimal-loss code accounting for both geometry and structure.

To synthesize new 3D shapes, we extend the pre-trained autoencoder RvNN into a generative model. We learn a distribution over root codes constructed from shape structures for 3D objects of the same class, e.g. chairs. This step utilizes a generative adversarial network (GAN), similar to a VAE-GAN [Larsen et al., 2015], to learn a low-dimensional manifold of root codes; see Figure 4.2(b). We sample and then project a root code onto the manifold to synthesize an OBB arrangement. In the final stage, the boxes are filled with part geometries by another generative model which learns a mapping between box features and voxel grids; see Figure 4.2(c).

We refer to our overall generative neural network as a *generative recursive autoencoder* for shape structures, or GRASS. Figure 4.2 provides an overview of the complete architecture.

The main contributions of our work can be summarized as follows.

- The first generative neural network model for structured 3D shape representations — GRASS. This is realized by an autoencoder RvNN which learns to encode and decode shape structures via discovered symmetry hierarchies, followed by two generative models trained to synthesize box-level symmetry hierarchies and volumetric part geometries, respectively.
- A novel RvNN architecture which extends the original RvNN of Socher et al. [Socher et al., 2011] by making it generative and capable of encoding a variety of merging operations (i.e. assembly by connectivity and symmetry groupings of different types).

- An *unsupervised* autoencoder RvNN which jointly learns and encodes the structure and geometry of box layouts of varying sizes into *fixed-length* vectors.

We demonstrate that our network learns meaningful structural hierarchies adhering to perceptual grouping principles, produces compact codes which enable applications such as shape classification and partial matching, and supports generative models which lead to shape synthesis and interpolation with significant variations in topology and geometry.

4.3 Related Work

Our work is related to prior works on statistical models of 3D shape structures, including recent works on applying deep neural networks to shape representation. These models can be discriminative or generative, and capture continuous or discrete variations. We review the most relevant works below. Since our focus is shape synthesis, we emphasize generative models in our discussion.

Statistical shape representations. Early works on capturing statistical variations of the human body explored smooth deformations of a fixed template [Allen et al., 2003, Anguelov et al., 2005, Blanz and Vetter, 1999b]. Later papers addressed discrete variations at the part level, employing stochastic shape grammars coded by hand [Müller et al., 2006], learned from a single training example [Bokeloh et al., 2010], or learned from multiple training examples [Talton et al., 2012]. Parallel works explored the use of part-based Bayesian networks [Chaudhuri et al., 2011, Kalogerakis et al., 2012] and modular templates [Fish et al., 2014, Kim et al., 2013] to represent both continuous and discrete variations. However, these methods are severely limited in the variety and complexity of part layouts they can generate, and typically only work well for shape families with a few consistently appearing parts and a restricted number of possible layouts. In a different approach, Talton et al. [Talton et al., 2009] learn a probability distribution over a shape space generated by a procedure operating on a fixed set of parameters. We are also inspired by some non-statistical shape representations such as the work of Wang et al. [Wang et al., 2011b] and van Kaick et al. [van Kaick et al., 2013] on extracting hierarchical structure from a shape: our goal in this paper is to learn consistent, probabilistic, hierarchical representations automatically from unlabeled datasets. Mitra et al. [Mitra et al., 2013] provide an overview of a range of further works on statistical and structure-aware shape representations.

Deep models of 3D shapes. Recently, the success of deep neural networks in computer vision, speech recognition, and natural language processing has inspired researchers to apply such models to 3D shape analysis. While these are of course statistical shape representations, their immediate relevance to this paper merits a separate section from the above. Most of these works have focused on extending computer vision techniques developed for images – 2D grids of pixels – to 3D grids of voxels. Wu et al. [Wu et al., 2015] propose a generative model based on a deep belief network trained on a large, unannotated database of voxelized 3D shapes. They show applications of the model to shape synthesis and probabilistic shape completion for next-best view prediction. Girdhar et al. [Girdhar et al., 2016] jointly train a deep convolutional encoder for 2D images and a deep convolutional decoder for voxelized 3D shapes, chained together so that the vector output of the encoder serves as the input code for the decoder, allowing 3D reconstruction from a 2D image. Yan et al. [Yan et al., 2016] propose a different encoder-decoder network for a similar application. Yumer and Mitra [Yumer and Mitra, 2016] present a 3D convolutional network that maps a voxelized shape plus a semantic modification intent to the deformation field required to realize that intent.

In a departure from voxel grids, Su et al. [Su et al., 2015] build a powerful shape classifier based on multiple projected views of the object, by fine-tuning standard image-based CNNs trained on huge 2D datasets and applying a novel pooling mechanism. Masci et al. [Masci et al., 2015] build a convolutional network directly on non-Euclidean shape surfaces. Qi et al. [Qi et al., 2016] discuss ways to improve the performance of both volumetric and multi-view CNNs for shape classification. In a recent work, Tulsiani et al. [Tulsiani et al., 2017] develop a discriminative, CNN-based approach to consistently parse shapes into a bounded number of volumetric primitives.

We are inspired by the work of Huang et al. [Huang et al., 2015], who develop a deep Boltzmann machine-based model of 3D shape surfaces. This approach can be considered a spiritual successor of Kalogerakis et al. [Kalogerakis et al., 2012] and Kim et al. [Kim et al., 2013], learning modular templates that incorporate fine-grained part-level deformation models. In addition to being fully generative – the model can be sampled for a point set representing an entirely new shape – the method automatically refines shape correspondences and part boundaries during training. However, like the prior works, this approach is limited in the variety of layouts it can represent.

Wu et al. [Wu et al., 2016] exploit the success of generative adversarial nets (GAN) [Goodfellow et al., 2014] to improve upon the model of Wu et al. [Wu et al., 2015]. At its core, their model is a generative decoder that takes as input a 200-D shape code and produces a voxel grid as output. The decoder is trained adversarially, and may be chained with a prior encoder that maps, say, a 2D image to the corresponding shape code. The method supports simple arithmetic and interpolation on the codes, enabling, for instance, topology-varying morphs between different shapes. Our work is complementary to this method: we seek to develop a powerful model of part layout variations that can accurately synthesize complex hierarchical structures beyond the representational power of low-resolution grids, can be trained on relatively fewer shapes, and is independent of voxel resolution.

Neural models of graph structure. The layout of parts of a shape inherently induces a non-Euclidean, graph-based topology defined by adjacency and relative placement. Several works, not concerning geometric analysis, have explored neural networks operating on graph domains. The most common such domains are of course linear chains defining text and speech signals. For these domains, recurrent neural networks (RNNs), as well as convolutional neural networks (CNNs) over sliding temporal windows, have proved very successful. Such linear models have even been adapted to generate non-linear output such as images, as in the work of van den Oord et al. [van den Oord et al., 2016b,c], producing the image row by row, pixel by pixel. These models are, however, limited in such adaptations since it is difficult to learn and enforce high-level graph-based organizational structure. Henaff et al. [Henaff et al., 2015], Duvenaud et al. [Duvenaud et al., 2015] and Niepert et al. [Niepert et al., 2016] propose convolutional networks that operate directly on arbitrary graphs by defining convolution as an operation on the radial neighborhood of a vertex. However, none of these works enable generative models. A different approach to this problem, which directly inspires our work, is the *recursive* neural network (RvNN) proposed by Socher et al. [Socher et al., 2012, 2011], which sequentially collapses edges of a graph to yield a hierarchy. We build upon the autoencoder version of this network, adapting it to learn the particular organizational principles that characterize 3D shape structure, and to extend it from a deterministic model to a probabilistic generative one.

4.4 Overview

Our method for learning GRASS, a hierarchical, symmetry-aware, generative model for 3D shapes, has three stages, shown in Figure 4.2. In this section, we summarize the stages and highlight important components and properties of the neural networks we use.

Geometry and structure encoding. We define an abstraction of symmetry hierarchies, which are composed of spatial arrangements of oriented bounding boxes (OBBs). Each OBB is defined by a fixed-length code to represent its geometry. The fixed length code encodes both the geometry of its child OBBs and their detailed grouping mechanism: whether by connectivity or symmetry.

Stage 1: Recursive autoencoder. In the first stage, we train an autoencoder for layouts of OBBs. The autoencoder maps a box layout with an arbitrary number and arrangement of components to a fixed-length root code that implicitly captures its salient features. The encoding is accomplished via a recursive neural network (RvNN) that repeatedly, in a bottom-up fashion, collapses a pair of boxes represented as codes into a merged code. The process also yields a hierarchical organizational structure for the boxes. The final code representing the entire layout is decoded to recover the boxes (plus the entire hierarchy) by an inverse process, and the training loss is measured in terms of a reconstruction error and back-propagated to update the network weights.

Stage 2: Learning manifold of plausible structures. We extend the autoencoder to a generative model of structures by learning a distribution over root codes that describes the shape manifold, or shape space, occupied by codes corresponding to meaningful shapes within the full code space. We train a generative adversarial model (GAN) for a low-dimensional manifold of root codes that can be decoded to structures indistinguishable, to an adversarial classifier, from the training set. Given a randomly selected root code, we project it to the GAN manifold to synthesize a plausible new structure.

Stage 3: Part geometry synthesis. In the final stage, the synthesized boxes are converted to actual shape parts. Given a box in a synthesized layout, we compute structure-aware recursive features that represent it in context. Then, we simultaneously learn a compact, invertible encoding of voxel grids representing part geometries as well as a mapping from contextual part features to the encoded voxelized geometry. This yields a procedure that can synthesize detailed geometry for a box in a shape structure.

By chaining together hierarchical structure generation and part geometry synthesis, we obtain the full GRASS pipeline for recursive synthesis of shape structures.

4.5 Recursive Model of Shape Structure

In this section, we describe a method to encode shape structures into a short, fixed-dimensional code. The learned encoding is fully invertible, allowing the structure to be reconstructed from the code. In Section 4.6, we present our method to adversarially tune this structure decoder to map random codes to structures likely to come from real shapes. By combining this generator for sampling plausible shape structures with a method for synthesizing the geometry of individual parts (Section 4.7), we obtain our probabilistic generative model for 3D shapes.

Our key observation is that shape components are commonly arranged, or perceived to be arranged, hierarchically. This is a natural organizational principle in well-accepted theories of human cognition and design, which has been extensively leveraged computationally [Serre, 2013]. Perceptual and functional hierarchies follow patterns of component proximity and symmetry. Hence, the primary goal of our structural code is to successfully encode the hierarchical organization of the shape in terms of symmetries and adjacencies. An important metric of success is that the hierarchies are *consistent* across different shapes of the same category. We achieve this via a compact model of recursive component aggregation that tries to consistently identify similar substructures.

Our model is based on Recursive Autoencoders (RAE) for unlabeled binary trees, developed by Socher et al. [Socher, 2014]. The RAE framework proposed by Socher et al. consists of an *encoder* neural network that takes two n-dimensional inputs and produces a single n-dimensional output, and a *decoder* network that recovers two n-D vectors from a single n-D vector. In our experiments, $n = 80$.

Given a binary tree with n-D descriptors for the leaves, the RAE is used to recursively compute descriptors for the internal nodes, ending with a root code. The root code can be inverted to recover the original tree using the decoder, and a training loss formulated in terms of a reconstruction error for the leaves.

RAEs were originally intended for parsing natural language sentences in a discriminative setting, trained on unlabeled parse trees. We adapt this framework for the task of learning and synthesizing hierarchical shape structures. This requires several important technical contributions, including extending the framework to accommodate multiple encoder and decoder types, handling non-binary symmetric groups of parts, and probabilistically generating shapes (as described in the subsequent sections).

4.5.1 Criteria for Recursive Merging.

Our model of hierarchical organization of shape parts follows two common perceptual/cognitive criteria for recursive merging: a mergeable subset of parts is either an adjacent pair (the

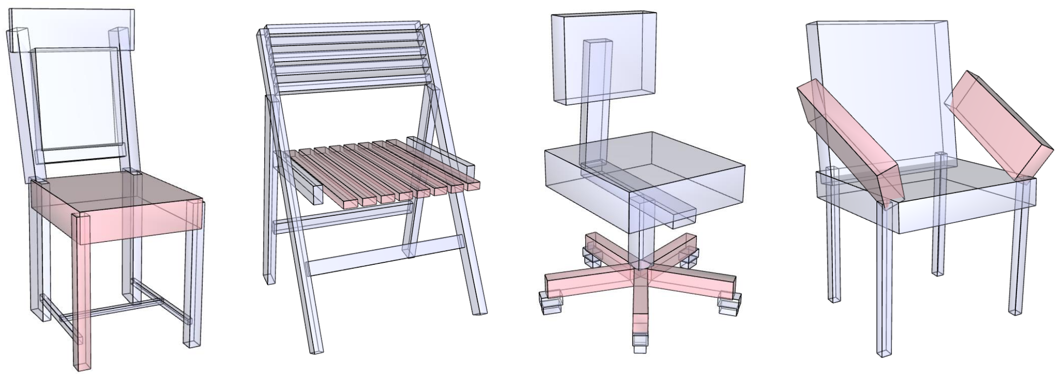


Figure 4.3: *Merging criteria used by our model demonstrated with 3D shapes represented by part bounding boxes (relevant parts highlighted in red). From left: (a) two adjacent parts, (b) translational symmetry, (c) rotational symmetry, and (d) reflective symmetry.*

adjacency criterion) or part of a symmetry group (the *symmetry* criterion)[2]. An adjacent pair is represented by the bounding boxes of constituent parts. In this stage, we are interested only in representing the gross layout of parts, so we discard fine-grained geometric information and store only oriented part bounding boxes, following earlier work on shape layouts [Ovsjanikov et al., 2011] — fine-grained geometry synthesis is described in Section 4.7. We recognize three different types of symmetries, each represented by the bounding box of a generator part plus further parameters: (1) pairwise reflectional symmetry, parametrized by the plane of reflection; (2) k-fold rotational symmetry, parametrized by the number of parts k and the axis of rotation; and (3) k-fold translational symmetry, parametrized by k and the translation offset between parts. The different scenarios are illustrated in Figure 4.3. We generate training hierarchies that respect these criteria, and our autoencoder learns to synthesize hierarchies that follow them.

4.5.2 Synthesizing Training Data.

To train our recursive autoencoder, we synthesize a large number of training hierarchies from a dataset of shapes. These shapes are assumed to be pre-segmented into constituent (unlabeled) parts, but do not have ground truth hierarchies. We adopt an iterative, randomized strategy to generate plausible hierarchies for a shape that satisfy the merging criteria described above. In each iteration, two or more parts are merged into a single one. A mergeable subset of parts is either adjacent or symmetric. We randomly sample a pair that satisfies one of the two criteria until no further merges are possible. In our experiments, we generated 20 training hierarchies for each shape in this fashion. Note that none of these hierarchies is intended to represent "ground truth". Rather, they sample the space of plausible part groupings in a relatively unbiased fashion for training purposes.

4.5.3 Autoencoder Model.

To handle both adjacency and symmetry relations, our recursive autoencoder comprises two distinct types of encoder/decoder pairs. These types are:

Adjacency. The encoder for the adjacency module is a neural network AdjEnc which merges codes for two adjacent parts into the code for a single part. It has two n-D inputs and one n-D output. Its parameters are a weight matrix $W_{ae} \in \mathbb{R}^{n \times 2n}$ and a bias vector $b_{ae} \in \mathbb{R}^n$, which are used to obtain the code of parent (merged) node y from children x_1 and x_2 using the formula

$$y = \tanh(W_{ae} \cdot [x_1\ x_2] + b_{ae})$$

[2]Currently, we make the reasonable single-object assumption that all parts are connected by either adjacency or symmetry. For disconnected, asymmetric shapes, we would need further merging criteria.

The corresponding decoder AdjDec splits a parent code y back to child codes x'_1 and x'_2, using the reverse mapping

$$[x'_1\ x'_2] = \tanh(W_{ad} \cdot y + b_{ad})$$

where $W_{ad} \in \mathbb{R}^{2n \times n}$ and $b_{ad} \in \mathbb{R}^{2n}$.

Symmetry. The encoder for the symmetry module is a neural network SymEnc which merges the n-D code for a generator part of a symmetry group, as well as the m-D parameters of the symmetry itself into a single n-D output. The code for a group with generator x and parameters p is computed as

$$y = \tanh(W_{se} \cdot [x\ p] + b_{se})$$

and the corresponding decoder SymDec recovers the generator and symmetry parameters as

$$[x' p'] = \tanh(W_{sd} \cdot y + b_{sd})$$

where $W_{se} \in \mathbb{R}^{n \times (n+m)}$, $W_{sd} \in \mathbb{R}^{(n+m) \times n}$, $b_{se} \in \mathbb{R}^n$, and $b_{sd} \in \mathbb{R}^{m+n}$. In our implementation, we use $m = 8$ to encode symmetry parameters comprising symmetry type (1D); number of repetitions for rotational and translational symmetries (1D); and the mirror plane for reflective symmetry, rotation axis for rotational symmetry, or position and displacement for translational symmetry (6D).

In practice, the encoders/decoders for both adjacency and symmetry are implemented as two-layer networks, where the dimensions of the hidden and output layers are 100D and 80D, respectively.

The input to the recursive merging process is a collection of part bounding boxes. These need to be mapped to n-D vectors before they can be processed by the autoencoder. To this end, we employ additional single-layer neural networks BoxEnc, which maps the 12D parameters of a box (concatenating box center, dimensions and two axes) to an n-D code, and BoxDec, which recovers the 12D parameters from the n-D code. These networks are non-recursive, used simply to translate the input to the internal code representation at the beginning, and back again at the end.

Lastly, we jointly train an auxiliary classifier NodeClsfr to decide which module to apply at each recursive decoding step. This classifier is a neural network with one hidden layer that takes as input the code of a node in the hierarchy, and outputs whether the node represents an adjacent pair of parts, a symmetry group, or a leaf node. Depending on the output of the classifier, either AdjDec, SymDec or BoxDec is invoked.

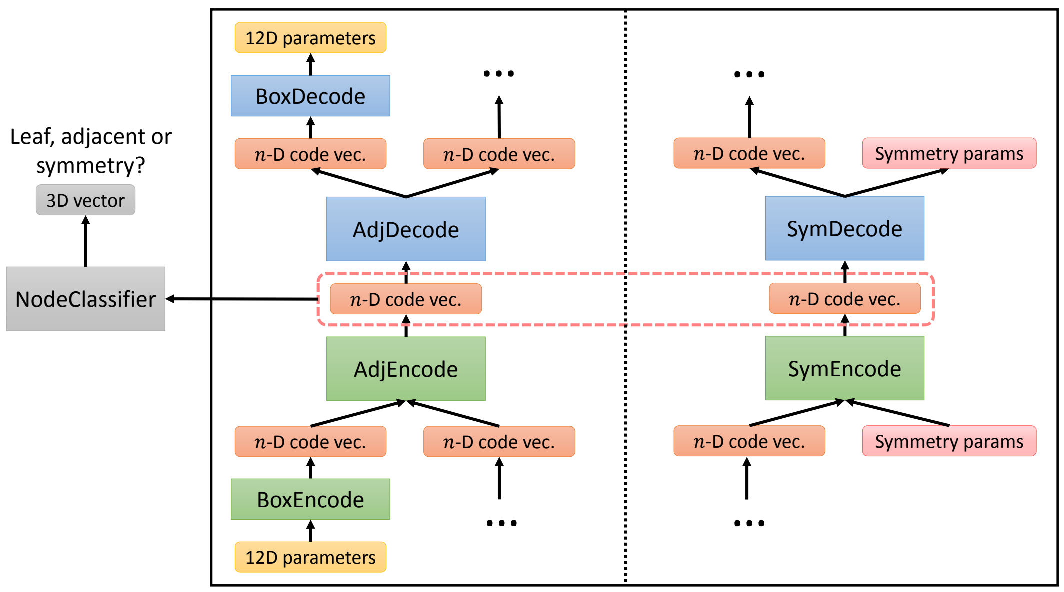


Figure 4.4: *Autoencoder training setup. Ellipsis dots indicate that the code could be either the output of* BoxEnc*,* AdjEnc *or* SymEnc*, or the input to* BoxDec*,* AdjDec *or* SymDec*.*

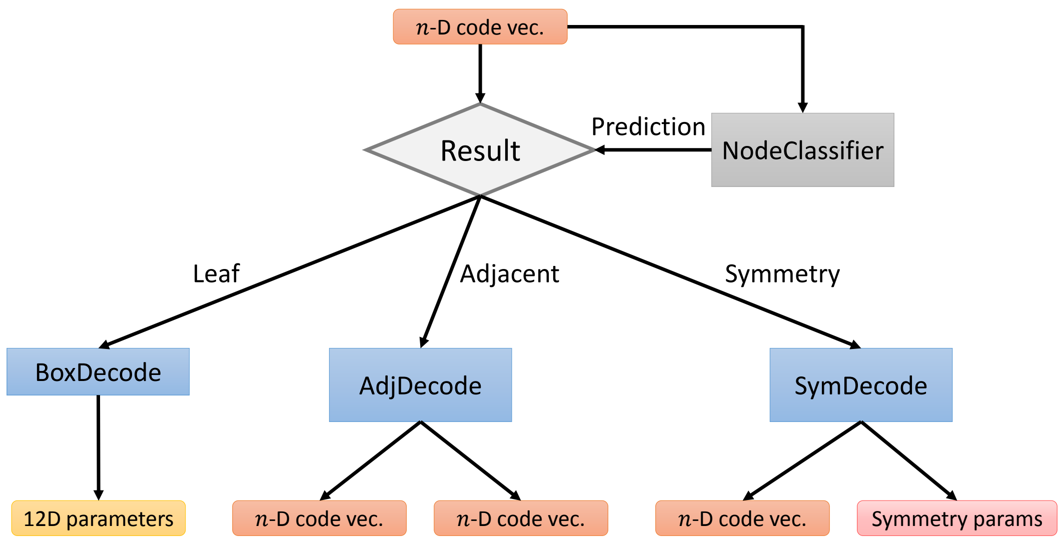


Figure 4.5: *Autoencoder test decoding setup.*

4.5.4 Training.

To train our recursive autoencoder, we use BFGS with back-propagation, starting with a random initialization of weights sampled from a Gaussian distribution. The loss is formulated as a reconstruction error. Given a training hierarchy, we first encode each leaf-level part bounding box using BoxEnc. Next, we recursively apply the corresponding encoder (AdjEnc or SymEnc) at each internal node until we obtain the code for the root. Finally, we invert the process, starting from the root code, to recover the leaf parameters by recursively applying the decoders AdjDec and SymDec, followed by a final application of BoxDec. The loss is formulated as the sum of squared differences between the input and output parameters for each leaf box.

Note that during training (but not during testing), we use the input hierarchy for decoding, and hence always know which decoder to apply at which unfolded node, and the mapping between input and output boxes. We simultaneously train NodeClsfr, with a three-class softmax classification with cross entropy loss, to recover the tree topology during testing. The training setup is illustrated in Figure 4.4.

4.5.5 Testing.

During testing, we must address two distinct challenges. The first is to infer a plausible encoding hierarchy for a novel segmented shape without hierarchical organization. The second is to decode a given root code to recover the constituent bounding boxes of the shape.

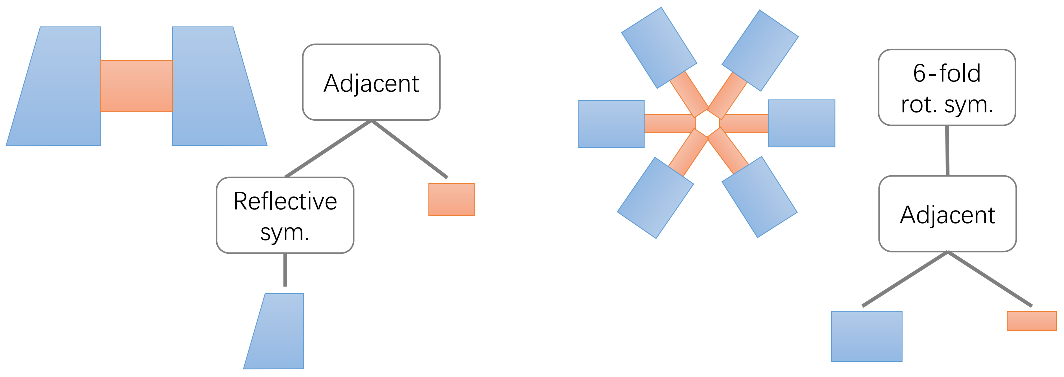


Figure 4.6: *Different two-step encoding orders for two examples, found by minimizing reconstruction errors during testing. Left: Symmetry (reflective) before adjacency. Right: Adjacency before symmetry (6-fold rotational).*

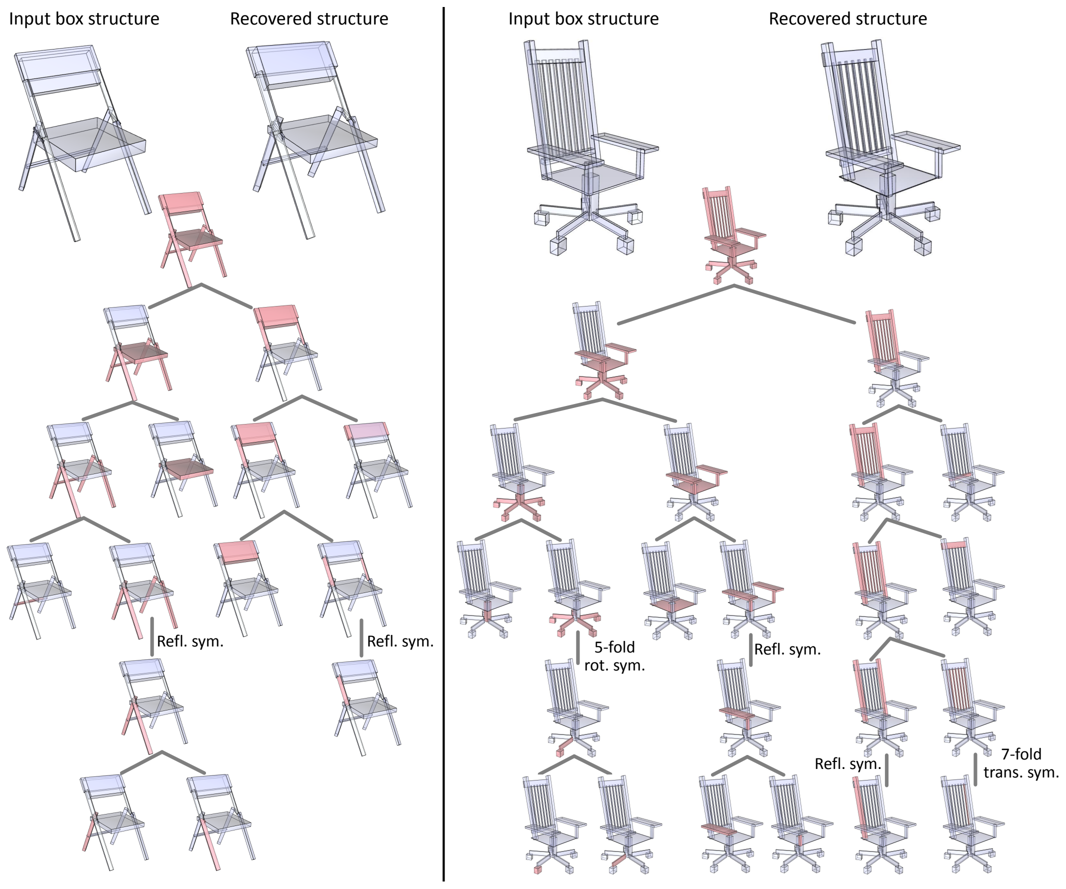


Figure 4.7: *Examples of reconstructing test shapes, without known hierarchies, by successively encoding them to root codes, and decoding them back. The encoding hierarchies inferred by our RvNN encoder are shown at the bottom.*

To infer a plausible hierarchy using the trained encoding modules, we resort to *greedy local search*. Specifically, we look at all subsets that are mergeable to a single part, perform *two* levels of recursive encoding and decoding, and measure the reconstruction error. The merge sequence with the lowest reconstruction error is added to the encoding hierarchy. The process repeats until no further merges are possible.

Particular cases of interest are *adjacency before symmetry*, and *symmetry before adjacency*, as illustrated in Figure 4.6. For each such case, we decode the final code back to the input box parameters (using, as for training, the known merging hierarchy) and measure the reconstruction error. This two-step lookahead is employed only for inferring hierarchies in test mode. During training, we minimize reconstruction loss over the hierarchy for the entire shape, as well as over all subtrees. Thus, the encoder/decoder units are tuned for both locally

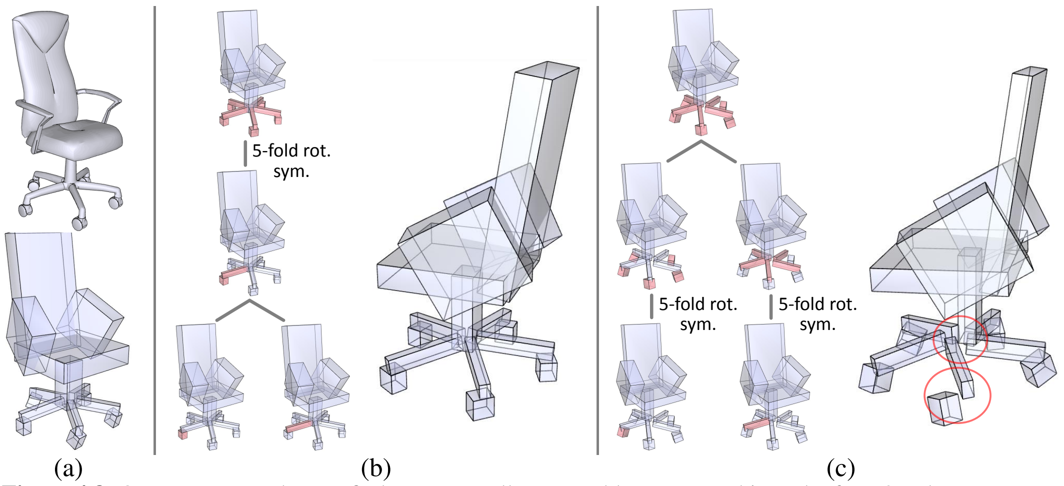


Figure 4.8: *Our RvNN encoder can find a perceptually reasonable symmetry hierarchy for a 3D shape structure, through minimizing reconstruction error. Given an input structure (a), the reconstruction error is much smaller if parts are grouped by adjacency before symmetry (b), instead of symmetry before adjacency (c).*

and globally good reconstructions, and at test time a relatively short lookahead suffices.

To decode a root code (e.g. one obtained from an encoding hierarchy inferred in the above fashion), we recursively invoke NodeClsfr to decide whether which decoder should expand the node. The corresponding decoder (AdjDec, SymDec or BoxDec) is used to recover the codes of child nodes until the full hierarchy has been expanded to leaves with corresponding box parameters. The test decoding setup is illustrated in Figure 4.5.

Several examples of test reconstructions are shown in Figure 4.7. The above procedures are used to encode a novel shape to a root code, and to reconstruct the shape given just this root code. In Figure 4.8, we show how our RvNN is able to find a perceptually reasonable symmetry hierarchy for a 3D shape structure, by minimizing the reconstruction error. Given the structure of a swivel chair, the error is much smaller when a wheel and spike are merged before the 5-fold rotational symmetry is applied, than if two separate rotational symmetries (for wheels and spikes respectively) are applied first.

4.6 Learning manifold of plausible structures

Our recursive autoencoder computes a compact, fixed-dimensional code that represents the inferred hierarchical layout of shape parts, and can recover the layout given just this code associated with the root of the hierarchy. However, the autoencoder developed so far is not a generative model. It can reconstruct a layout from *any* root code, but an arbitrary, random

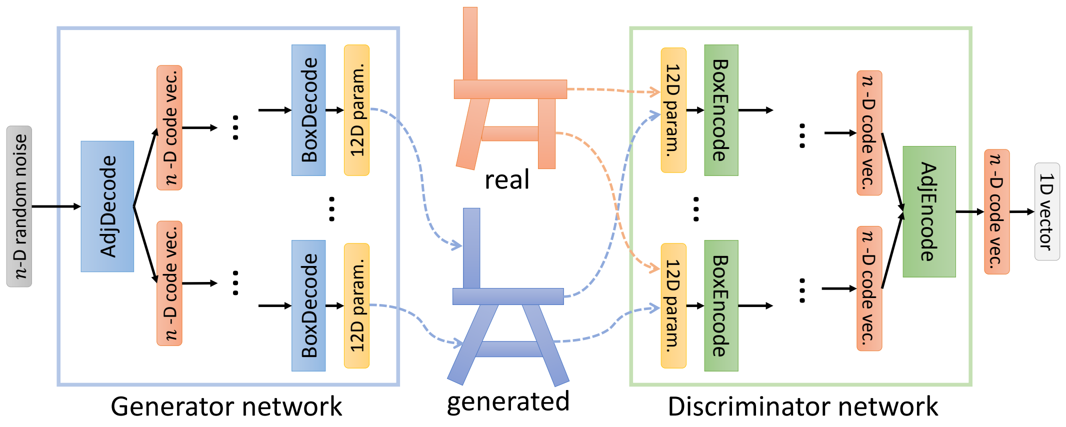


Figure 4.9: *Architecture of our generative adversarial network, showing reuse of autoencoder modules.*

code is unlikely to produce a plausible layout. A generative model must jointly capture the distribution of statistically plausible shape structures.

In this section, we describe our method for converting the auto-encoder-based model to a fully generative one. We fine-tune the autoencoder to learn a (relatively) low-dimensional manifold containing high-probability shape structures. Prior approaches for learning feasible manifolds of parametrized 3D shapes from landmark exemplars include kernel density estimation [Fish et al., 2014, Talton et al., 2009], multidimensional scaling [Averkiou et al., 2014], and piecewise primitive fitting [Schulz et al., 2016]. However, these methods essentially reduce to simple interpolation from the landmarks, and hence may assign high probabilities to parameter vectors that correspond to implausible shapes [Goodfellow et al., 2014].

Recently, *generative adversarial networks* (GANs) [Goodfellow et al., 2014] have been introduced to overcome precisely this limitation. Instead of directly interpolating from training exemplars, a GAN trains a synthesis procedure to map arbitrary parameter vectors only to vectors which a classifier deems plausible. The classifier, which can be made arbitrarily sophisticated, is jointly trained to identify objects similar to the exemplars as plausible, and others as fake. This leads to a refined mapping of the latent space since implausible objects are eliminated by construction. Given a completely random set of parameters, the trained GAN "snaps" it to the plausible manifold to generate a meaningful sample.

In addition to enabling the synthesis of novel but statistically plausible shape structures, the learned manifold also supports interpolation between shape codes. The application of this feature to shape morphing is shown in Section 4.8.

4.6.1 GAN Architecture.

The architecture of our generative adversarial network comprises a generator (G) network, which transforms a random code to a hierarchical shape structure lying on the estimated manifold, and a discriminator (D) network, which checks whether a generated structure is similar to those of the training shapes or not. Our key observation is that we can *directly reuse and fine-tune the autoencoder modules* learned in the previous section, instead of introducing new components. The decoder component (comprising AdjDec, SymDec, BoxDec and NodeClsfr) is exactly what we need to estimate a structure from a given code: it constitutes the G network. The encoder component (comprising AdjEnc, SymEnc and BoxEnc) is exactly what we need to estimate a code for the generated structure. The final code can be compared to the codes of training structures using an additional fully connected layer and a binary softmax layer producing the probability of the structure being "real". This constitutes the D network. Hence, we initialize the GAN with the trained autoencoder modules and further fine-tune them to minimize the GAN loss. The architecture is illustrated in Figure 4.9, and the training procedure described below.

Training. The GAN is trained by stochastic gradient descent using different loss functions for the discriminator D and the generator G. In each iteration, we sample two mini-batches: training box structures x with their associated hierarchies, and random codes $z \in \mathbb{R}^n$. The x samples, with known hierarchies, are passed only through the discriminator, yielding $D(x)$, whereas the z samples are passed through both networks in sequence, yielding $D(G(z))$. The loss function for the discriminator is

$$J_D \;=\; -\frac{1}{2}E_x\left[\log D(x)\right] - \frac{1}{2}E_z\left[\log(1 - D(G(z)))\right],$$

while the loss function for the generator is

$$J_G \;=\; -\frac{1}{2}E_z\left[\log D(G(z))\right].$$

By minimizing the first loss function w.r.t. the weights of the network D, we encourage the discriminator to output 1 for each training sample, and 0 for each random sample. By minimizing the second loss function w.r.t. the weights of the network G, we encourage the generator to fool D into thinking a random sample is actually a real one observed during training. This is a standard adversarial training setup; see Goodfellow et al. [Goodfellow et al., 2014] for more details.

With this straightforward training, however, it is still hard to converge to a suitable balance between the G and D networks, despite the good initialization provided by our autoencoder. This is due to the following reasons. *First*, when mapping a random code z to the manifold,

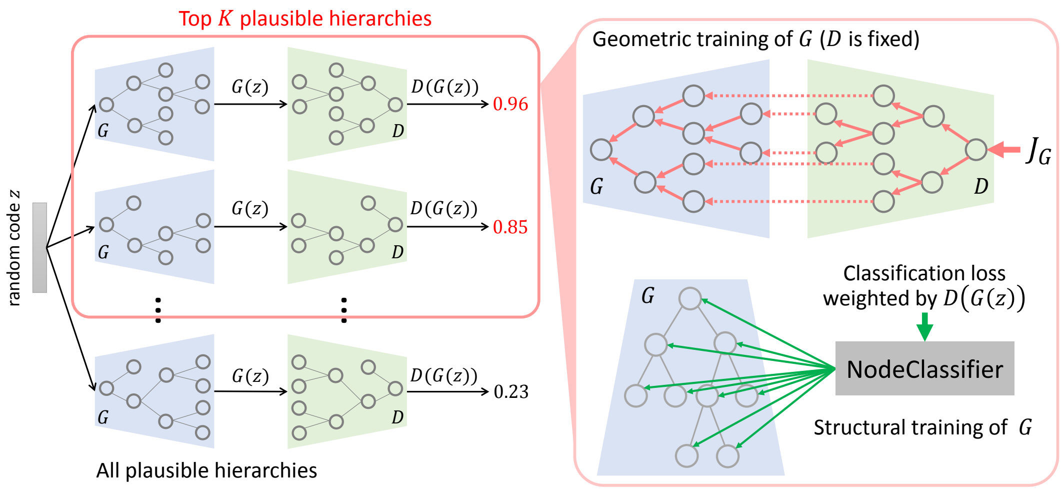

Top K plausible hierarchies
random code z
G
$G(z)$
D
$D(G(z))$
0.96
0.85
0.23
All plausible hierarchies
Geometric training of G (D is fixed)
J_G
Classification loss weighted by $D(G(z))$
NodeClassifier
Structural training of G

Figure 4.10: *The training of our GAN model. Left: Given a random code, we select the top K "plausible" hierarchies from which G can decode a box structure to best fool D. Right: For each selected hierarchy, the training of G is split into geometric (top) and structural (bottom) tuning, based on different loss functions.*

the G network (which is just the recursive decoder) may infer a grossly incorrect hierarchy. The D network finds it easy to reject these implausible hierarchies, and hence does not generate a useful training signal for G. *Second*, the implausible hierarchies generated from random codes may not provide reasonable pathways to back-propagate the loss from D so that G can be tuned properly. *Third*, since the decoding networks in G are split into geometric (e.g. AdjDec) and topological (NodeClsfr) types, they should be tuned separately with different losses deduced from D. To these ends, we devise the following training strategies and priors, to better constrain the training process:

- *Structure prior for G.* In an initial stage, we need to prevent G from mapping a random code z to a severely implausible hierarchy. This is achieved by introducing a strong structure prior to G. We constrain the hierarchies inferred by G to lie in a plausible set. This set includes all hierarchies used to train the autoencoder in Section 4.5. It also includes all hierarchies inferred by the autoencoder, in test mode, for the training shapes. For each z, we search the plausible hierarchies for the top $K = 10$ ones that best fool the discriminator, minimizing J_G (Figure 4.10, left). These hierarchies are then used to back-propagate the loss J_G.
- *Separate geometric and structural training.* Given a selected hierarchy, we first tune the geometric decoders of G via back-propagating the corresponding loss J_G through the hierarchy. This tuning is expected to further fool the discriminator, leading to a higher estimate $D(G(z))$ that $G(z)$ is real (Figure 4.10, top-right). For each selected hierarchy, with its the newly updated $D(G(z))$, we then tune the structural component, NodeClsfr, of G. This is done by minimizing the classification loss of NodeClsfr at

each node in the given hierarchy, using the node type as ground-truth (Figure 4.10, bottom-right). To favor those hierarchies that better fool D, we weight the loss by $D(G(z))$.

- *Constrained random code sampling.* Given the priors and constraints above, it is still difficult to train G to reconstruct a plausible hierarchy from arbitrarily random codes. Therefore, instead of *directly* feeding it random codes from a normal or uniform distribution, we feed it codes drawn from Gaussians around the training samples x, whose *mixture* approximates the standard normal distribution. Further, we train a secondary network f_l to project these "latent" codes to a space of potential root codes which are easier for G to process.
  Specifically, G takes samples from a multivariate Gaussian distribution: $z_s(x) \sim N(\mu, \sigma)$ with $\mu = f_\mu(Enc(x))$ and $\sigma = f_\sigma(Enc(x))$. Here, Enc is the recursive encoder originally trained with the autoencoder (before adversarial tuning), running in test mode. f_μ and f_σ can be approximated by two neural networks.
  We train to minimize the reconstruction loss on x, in addition to the generator loss in the GAN. In fact, the networks Enc and G constitute a variational autoencoder (VAE) if we also tune Enc when learning the parameters of the Gaussian distribution. This leads to an architecture similar to the VAE-GAN proposed by Larsen et al. [Larsen et al., 2015]; see Figure 4.11.
  Consequently, we also impose the loss function for VAE that pushes this variational distribution $p(z_s(x))$, over all training samples x, towards the prior of the standard normal distribution $p(z)$. In summary, we minimize the following loss function:

$$L = L_{\text{GAN}}(z_p) + \alpha_1 L_{\text{recon}} + \alpha_2 L_{\text{KL}}$$

  The GAN loss is $L_{\text{GAN}} = \log D(x) + \log(1 - D(G(f_l(z_p))))$, with $z_p \sim p(z)$. This loss is minimized/maximized by G/D, respectively. The reconstruction loss is defined as $L_{\text{recon}} = \|G(f_l(z_s(x))) - x\|_2$. The Kullback-Leibler divergence loss, $L_{\text{KL}} = D_{\text{KL}}\left(p(z_s(x)) \,\|\, p(z)\right)$, forces the mixture of local Gaussians to approximate the standard normal distribution. We set $\alpha_1 = 10^{-2}$ and $\alpha_2 = 10$ in our experiments.

The results of the GAN training process are fine-tuned RvNN decoder modules and the f_l network. The f_l network projects a random n-D vector drawn from the standard normal distribution to the space of potential root codes, and the tuned decoders map the projected vector to a structure lying on the plausible manifold. Together with a module to generate fine-grained part geometry, described in the next section, this constitutes our recursive, generative model of 3D shapes.

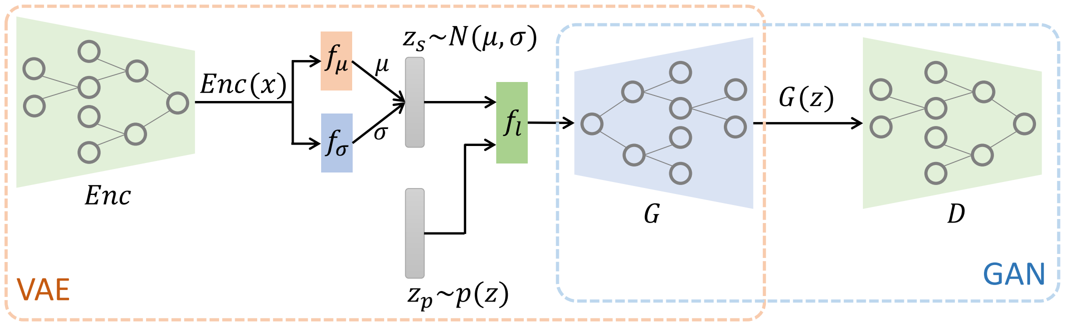


Figure 4.11: *Confining random codes by sampling from a learned Gaussian distributions based on learned root codes $Enc(x)$. Jointly learning the distribution and training the GAN leads to a VAE-GAN network.*

4.7 Part geometry synthesis

In the previous sections, we described our generative model of part layouts in shapes. The final component of our framework is a generative model for fine-grained part geometry, conditioned on the part bounding box and layout. Our solution has two components. First, we develop a fixed-dimensional part feature vector that captures both the part's gross dimensions and its context within the layout. Second, we learn a low-dimensional manifold of plausible part geometries while simultaneously also learning a mapping from part feature vectors to the manifold. This mapping is used to obtain the synthesized geometry for a given part in a generated layout. Below, we describe these steps in detail.

4.7.1 Structure-Aware Recursive Feature (SARF)

The recursive generator network produces a hierarchy of shape parts, with each internal node in the hierarchy represented by an n-D code. We exploit this structure to define a feature vector for a single part. A natural contextual feature would be to concatenate the RvNN codes of all nodes on the path from the part's leaf node to the root. However, since paths lengths are variable, this would not yield a fixed-dimensional vector. Instead, we approximate the context by concatenating just the code of the leaf node, that of its immediate parent, and that of the root into a $3n$-D feature vector (Figure 4.12). The first code captures the dimensions of the part's bounding box, and the latter two codes capture local and global contexts, respectively.

4.7.2 SARF to Part Geometry.

In the second stage, we would like to map a SARF feature vector to the synthesized geometry for the part, represented in our prototype as a $32 \times 32 \times 32$ voxel grid. Such a mapping function is difficult to train directly, since the output is very high (8000) dimensional yet the set of *plausible* parts spans only a low-dimensional manifold within the space of all outputs. Instead, we adapt a strategy inspired by Girdhar et al. [Girdhar et al., 2016]. We set up a deep, convolutional autoencoder, consisting of an encoder GeoEnc to map the voxel grid to a compact, 32D part code, and a decoder GeoDec to map it back to a reconstructed grid. The learned codes efficiently map out the low-dimensional manifold of plausible part geometries. We use the architecture of Girdhar et al., and measure the reconstruction error as a sigmoid cross-entropy loss. Simultaneously, we use a second deep network GeoMap to map an input SARF code to the 32D part code, with both networks accessing the same code neurons. The mapping network employs a Euclidean loss function. We train both networks jointly, using both losses, with stochastic gradient descent and backpropagation. At test time, we chain together the mapping network GeoMap and the decoder GeoDec to obtain a function mapping SARF codes to synthesized part geometry. The training and test setups are illustrated in Figure 4.13. The synthesis of the overall shape geometry is done by predicting part-wise 3D volumes, which are then embedded into a global volume, from which we reconstruct the final meshed model. See Figure 4.14 for an example.

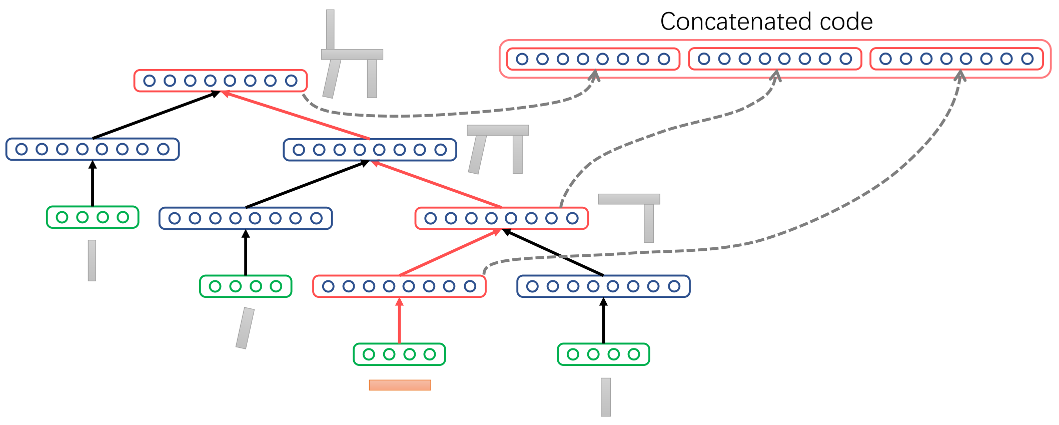


Figure 4.12: *Construction of structure-aware recursive feature (SARF) for a part in a hierarchy. We concatenate the RvNN codes of the part, its immediate parent, and the root into a fixed-dimensional vector.*

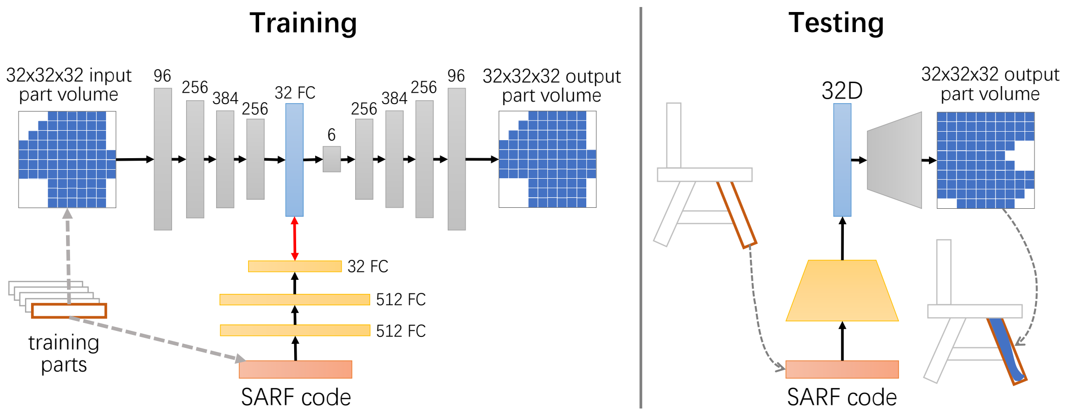


Figure 4.13: *Training and testing setup for part geometry synthesis.*

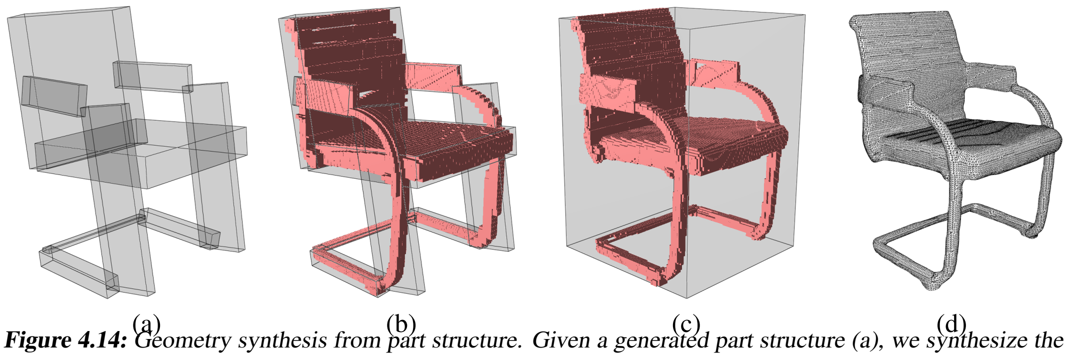


Figure 4.14: *Geometry synthesis from part structure. Given a generated part structure (a), we synthesize the geometry inside each part box in volumetric representation (b). The per-box volumes are then embedded into a global volume (c) from which we reconstruct the final meshed model (d).*

4.8 Results and evaluation

We evaluate our generative recursive model of shape structures through several experiments. First, we focus on validating that our autoencoder-based RvNN learns the "correct" symmetry hierarchies, where correctness could be qualified in different ways, and the resulting codes are useful in applications such as classification and partial matching. Then we test the generative capability of our VAE-GAN network built on top of the RvNN.

4.8.1 Dataset

We collected a dataset containing 1000 3D models from five shape categories: chairs (500), bikes (200), aeroplanes (100), excavators (100), and candelabra (100). These models are collected from the ShapeNet and the Princeton ModelNet. Each model is pre-segmented

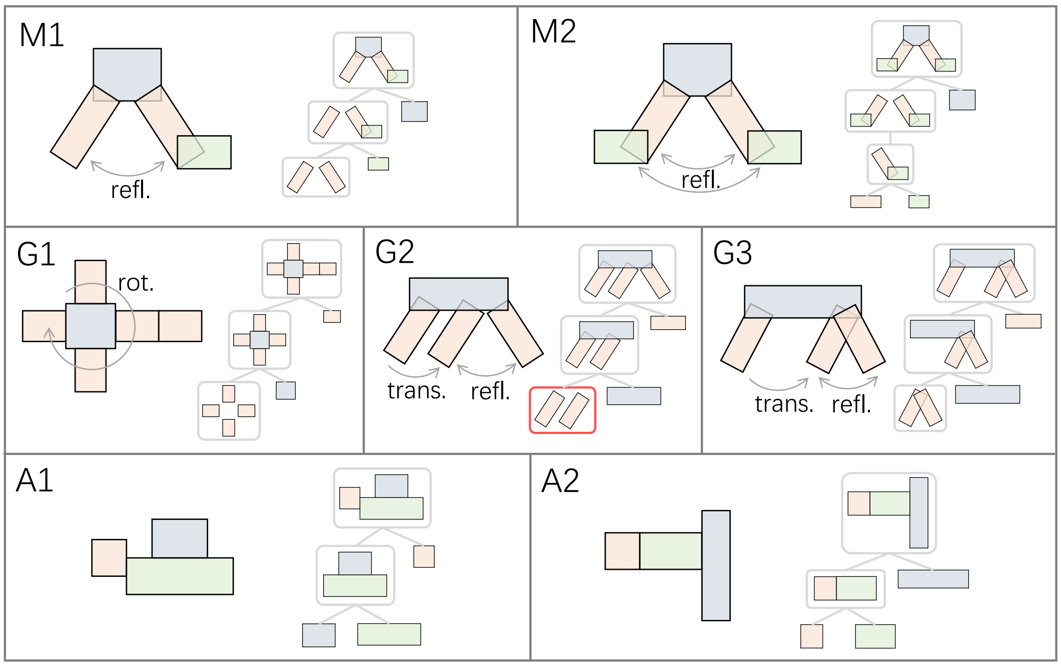


Figure 4.15: *Our RvNN encoder correctly parses six out of seven 2D box arrangements designed to test handcrafted, perceptually-based grouping rules from [Wang et al., 2011b]. The G2 rule is violated in our example, with 2-fold translational symmetry (highlighted in the red box) taking precedence over the reflectional one.*

according to their mesh components or based on the symmetry-aware segmentation utilized in [Wang et al., 2011b]. The average number of segments per shape is 12 for chairs, 10 for bikes, 7 for aeroplanes, 6 for excavators, and 8 for candelabra. Symmetric parts are counted as distinct. We do not utilize any segment labels.

Our RvNN autoencoder is trained with all shapes in the dataset. The generative VAE-GAN is trained per category, since its training involves structure learning which works best within the same shape category. Part geometry synthesis is trained on all parts from all categories.

4.8.2 Learning Recursive Grouping Rules.

In the original work on symmetry hierarchies by Wang et al. [Wang et al., 2011b], a total of seven precedence rules (labeled M1, M2, G1, G2, G3, A1, and A2; see the Appendix for a reproduction of these rules) were handcrafted to determine orders between and among assembly and symmetry grouping operations. For example, rule A1 stipulates that symmetry-preserving assembly should take precedence over symmetry-breaking assembly and rule M2 states that assembly should be before grouping (by symmetry) if and only if the assembled elements belong to symmetry groups which possess equivalent grouping symmetries. These rules were inspired by Gestalt laws of perceptual grouping [Köhler, 1929] and Occam's Razor which seeks the simplest explanation. One may say that they are perceptual and represent a certain level of human cognition.

The intriguing question is whether our RvNN, which is unsupervised, could "replicate" such cognitive capability. To test the rules, we designed seven box arrangements in 2D, one per rule; these patterns are quite similar to those illustrated in Wang et al. [Wang et al., 2011b]. For rule A2, which involves a connectivity strength measure, we simply used geometric proximity. In Figure 4.15, we show the seven box arrangements and the grouping learned by our RvNN. As can be observed, our encoder correctly parses all expected patterns except in the case of G2, where 2-fold translational symmetry takes precedence over the reflectional one in our example.

4.8.3 Consistency of Inferred Hierarchies.

Our RvNN framework infers hierarchies consistently across different shapes. To demonstrate this, we augment two categories of our segmented dataset – *chair* and *candelabra* – with semantic labels (e.g., for chairs: "seat", "back", "leg", and "arm"). Note that these labels occur at relatively higher levels of the hierarchies, since legs, backs, etc., may be subdivided into smaller parts. If the hierarchies are consistent across shapes, these high-level labels should follow a consistent merging order. For example, the seat and legs should be merged before the seat and back are merged. Let ℓ_p denote the label of part p. Given another label ℓ, let $h(p, \ell)$ denote the shortest distance from p to an ancestor that it shares with a part with label ℓ. Note that $h(p, \ell_p) = 0$ by definition. Let S_ℓ be the set of parts with label ℓ. For labels ℓ_1, ℓ_2, we measure the probability $P_\ell(\ell_1 \prec \ell_2)$ that ℓ_1 is more regularly grouped with ℓ than ℓ_2 as $\sum_{p \in S_\ell} \mathbb{I}(h(p, \ell_1) < h(p, \ell_2))/|S_\ell|$, where $\mathbb{I}$ is the indicator function and the sum is additionally restricted over shapes in which all three labels appear. The overall consistency is estimated as one minus the average entropy over all label triplets:

$$C = 1 + \binom{|L|}{3}^{-1} \sum_{\ell, \ell_1, \ell_2 \in L,\ \ell \neq \ell_1 \neq \ell_2} P_\ell(\ell_1 \prec \ell_2) \log_2 P_\ell(\ell_1 \prec \ell_2)$$

The average consistency over the two categories of training shapes was measured as 0.81, and over the two categories of test shapes as 0.72. The high values show that our RvNN infers hierarchies consistently across different shapes. Figure 4.16 shows several pairs of shapes with consistent inferred hierarchies.

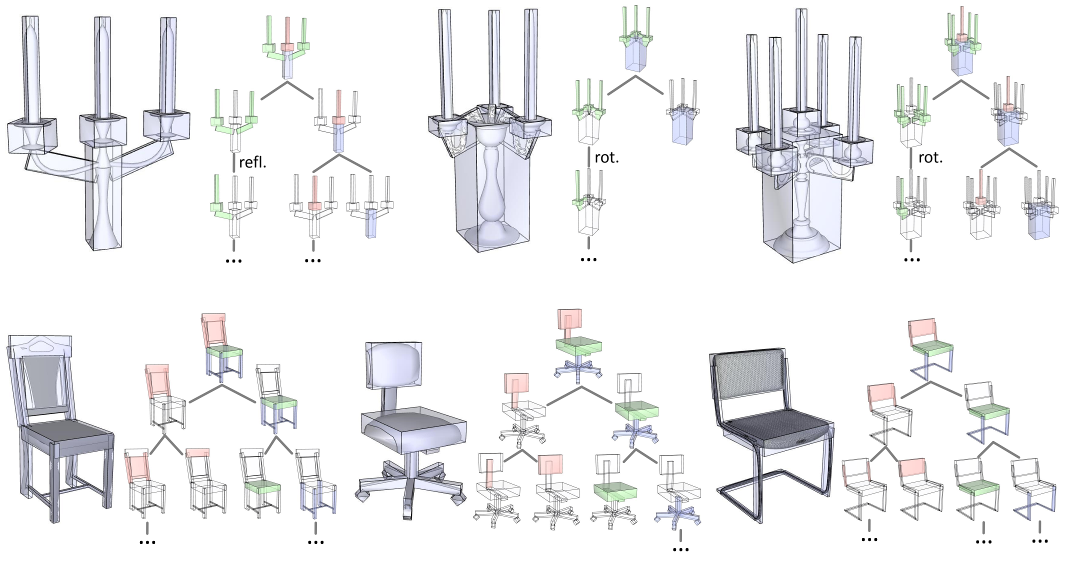


Figure 4.16: *Inferred hierarchies are consistent across sets of shapes, shown for two shape classes (candelabra and chairs).*

4.8.4 Classification of Shape Structures.

Our autoencoder generates compact encodings for shapes segmented into arbitrary numbers of parts, via a recursively inferred hierarchy. To test whether these codes effectively characterize shapes and shape similarities, we conducted a *fine-grained* shape classification experiment for each of four classes: airplane, chair, bike, and candle. The sub-classes were: airplane – 5 classes including jet, straight-wing, fighter, delta-wing, swept-wing; chair – 5 classes including armchair, folding, swivel, four-leg, sofa; bike – 4 classes including motorcycle, casual bicycle, tricycle, mountain bike; and candelabra – 3 classes including with arms, w/o arm, with two-level arms. To represent each shape, we used the average of all codes in the shape's hierarchy, which, as in Socher et al. [Socher et al., 2011], we found to work better than just the root.

Following the standard protocol for each category of shapes, we hold out one shape in turn, and sort the remaining shapes by increasing the L_2 distance between average codes, terminating the results by a variable upper limit on the distance. The number of results from the class of the query shape are considered as true positives.

We show precision-recall plots for four classes of interest in Figure 4.17. The average accuracy of (subclass) classification over all four classes is 96.1%.

As baselines, we show the performance of two state-of-the-art descriptors on this task [Su et al., 2015, Wu et al., 2015]. This is not an entirely equal-grounded comparison: our method

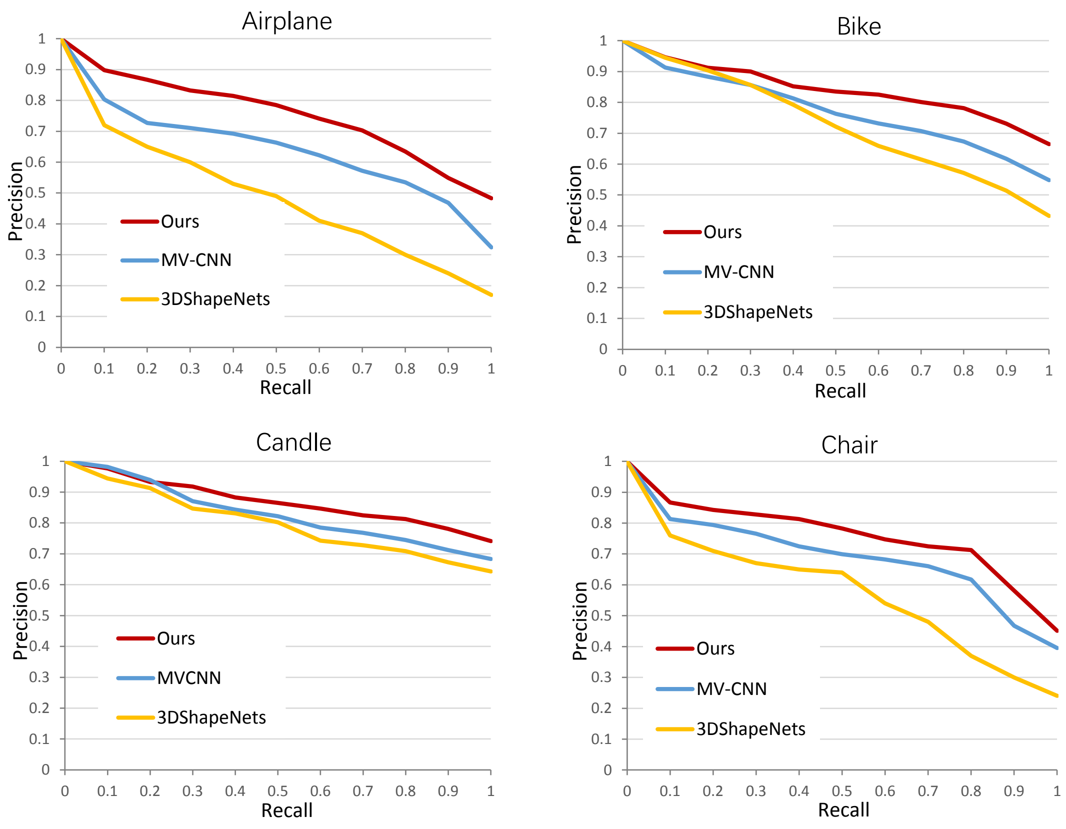


Figure 4.17: *Precision-recall plots for classification tasks.*

leverages a prior segmentation of each shape into (unlabeled) parts, whereas the baseline methods do not. However, our method does not consider any fine-grained part geometry, only oriented bounding box parameters. The considerable improvement of our method over the baselines demonstrates that gross structure can be significantly more important for shape recognition than fine-grained geometry, and accurate and consistent identification of part layouts can be the foundation of powerful retrieval and classification methods.

4.8.5 Partial Structure Matching.

While the previous experiment tested full shape retrieval, it is also interesting to explore whether subtree codes are sufficiently descriptive for part-in-whole matching. As before, we use the average of codes in a subtree as the feature for the subtree. Figure 4.18 contains some partial retrieval results, showing that our method correctly retrieves subparts matching the query.

4.8.6 Shape Synthesis and Interpolation.

Our framework is generative, and can be used to synthesize shapes from the learned manifold in a two-step process. First, the VAE-GAN network is sampled using a random seed for a hierarchical bounding box layout. Second, the leaf nodes of the hierarchy are mapped to fine-grained voxelized geometry, which is subsequently meshed. Several examples of synthesized shapes are shown in Figure 4.19.

Our model can also be used to interpolate between two topologically and geometrically different shapes. For this task, we compute the root codes of two shapes via inferred hierarchies. Then, we linearly interpolate between the codes, reconstructing the shape at each intermediate position using the synthesis procedure above. Although intermediate codes may not themselves correspond to root codes of plausible shapes, the synthesis procedure projects them onto the valid manifold by virtue of the VAE-GAN training. We demonstrate

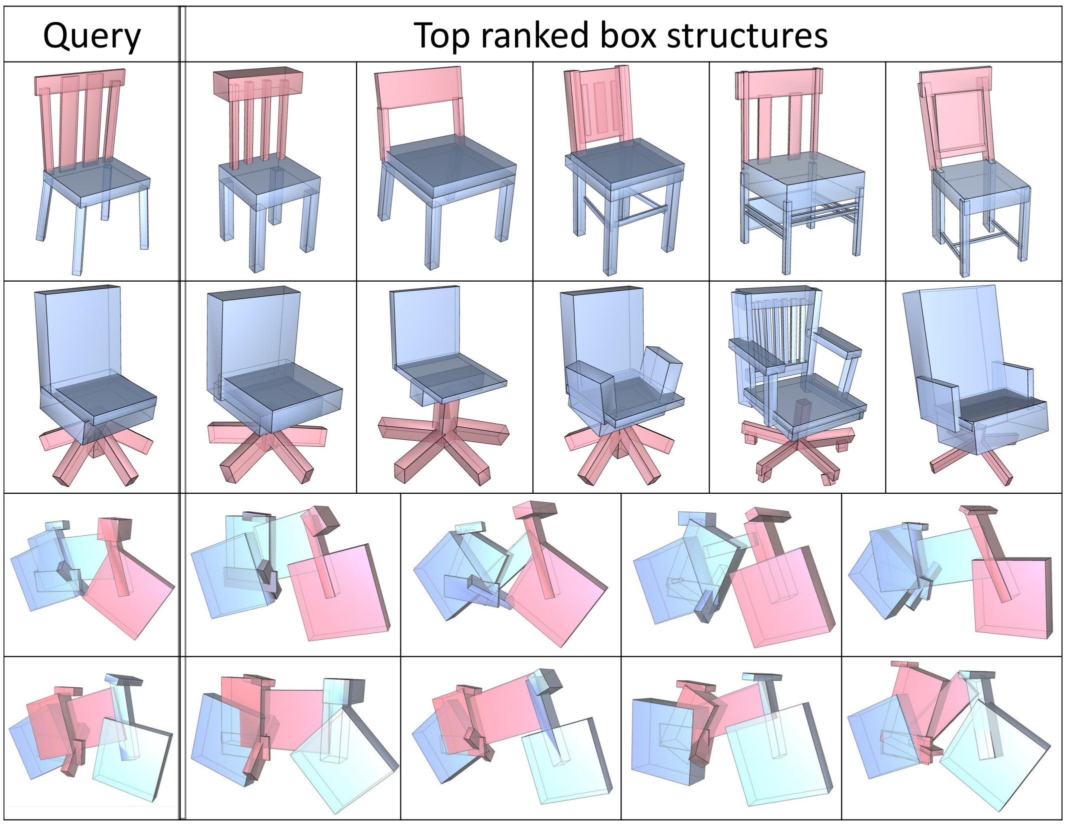


Figure 4.18: *Partial structure retrieval results for two shape classes (chair and bicycle). The query and matching parts are highlighted in red.*

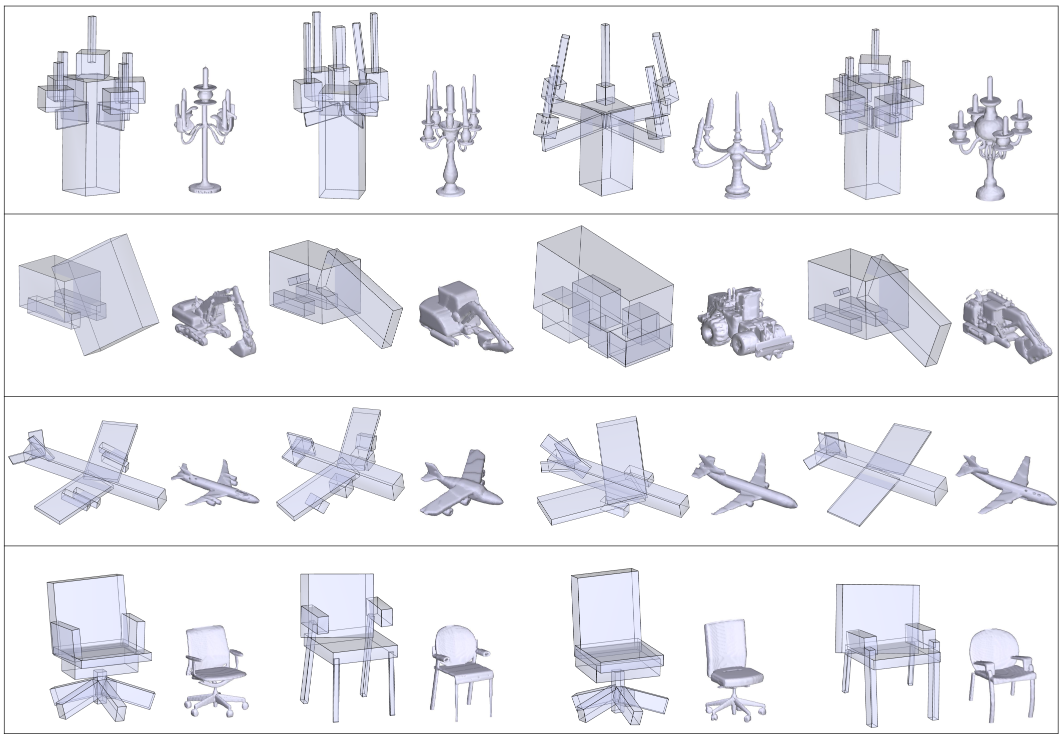


Figure 4.19: *Examples of shapes synthesized from different classes.*

example interpolations in Figure 4.20. Note that our model successfully handles topological changes both in the part layout and within parts, while maintaining symmetry constraints. Unlike Jain et al. [Jain et al., 2012], we do not require prior knowledge of part hierarchies. Unlike both Jain et al. and Alhashim et al. [Alhashim et al., 2014], we do not require part correspondences either, and we can handle smooth topological changes in individual parts.

4.8.7 Implementation and Timing.

Our RvNN and VAE-GAN are implemented in MATLAB. The geometry synthesis model is implemented using the MatConvNet neural network library. Pre-training the autoencoder (Section 4.5) took 14 hours. Adversarial fine-tuning (Section 4.6) took about 20 hours for each shape class. Training the part geometry synthesis network (Section 4.7) took 25 hours. Mapping a random code vector to the manifold of plausible structures to synthesize a hierarchy takes 0.5 seconds, and augmenting it with synthesized fine-grained part geometry takes an additional 0.2 seconds per part.

4.9 Discussion, limitation, and Future Work

With the work presented, we have only made a first step towards developing a structure-aware, generative neural network for 3D shapes. What separates our method apart from previous attempts at using neural nets for 3D shape synthesis is its ability to learn, without supervision, and synthesize *shape structures*. It is satisfying to see that the generated 3D shapes possess cleaner part structures, such as symmetries, and more regularized part geometries, when compared to voxel fields generated by previous works [Girdhar et al., 2016, Wu et al., 2016]. What is unsatisfying however is that we decoupled the syntheses of structure and fine geometry. This hints at an obvious next step to integrate the two syntheses.

The codes learned by our RvNN do combine structural and geometric information into a single vector. Through experiments, we have demonstrated that the hierachical grouping learned by the RvNN appears to conform to perceptual principles as reflected by the precedence rules handcrafted by Wang et al. [Wang et al., 2011b]. The codes also enable applications such as fine-grained classification and partial shape retrieval, producing reasonable results. However, the internal mechanisms of the code and precisely how it is mixing the structural and geometric information is unclear. The fact that it appears to be able to encode hiearachies of arbitrary depth with a fixed-length vector is even somewhat mysterious. An interesting future work would be to "visualize" the code to gain an insight on all of these questions. Only with that insight would we be able to steer the code towards a better separation between the parts reflecting the structure and the parts reflecting low-level geometry.

Our current network still has a long way to go in fully mapping the *generative structure manifold*. We cannot extrapolate arbitrarily – we are limited to a VAE-GAN setup which samples codes similar to, or in between, the exemplars. Hence, our synthesis and interpolation are confined to a local patch of that elusive "manifold". In fact, it is not completely clear whether the generative structure space for a 3D shape collection with sufficiently rich structural variations is a low-dimensional manifold. Along similar lines, we have not discovered flexible mechanisms to generate valid codes, e.g., by applying algebraic or crossover operations, from available codes. All of these questions and directions await future investigations. It would be interesting to thoroughly investigate the effect of code length on structure encoding. Finally, it is worth exploring recent developments in GANs, e.g. Wasserstein GAN [Arjovsky et al., 2017], in our problem setting. It would also be interesting to compare with plain VAE and other generative adaptations.

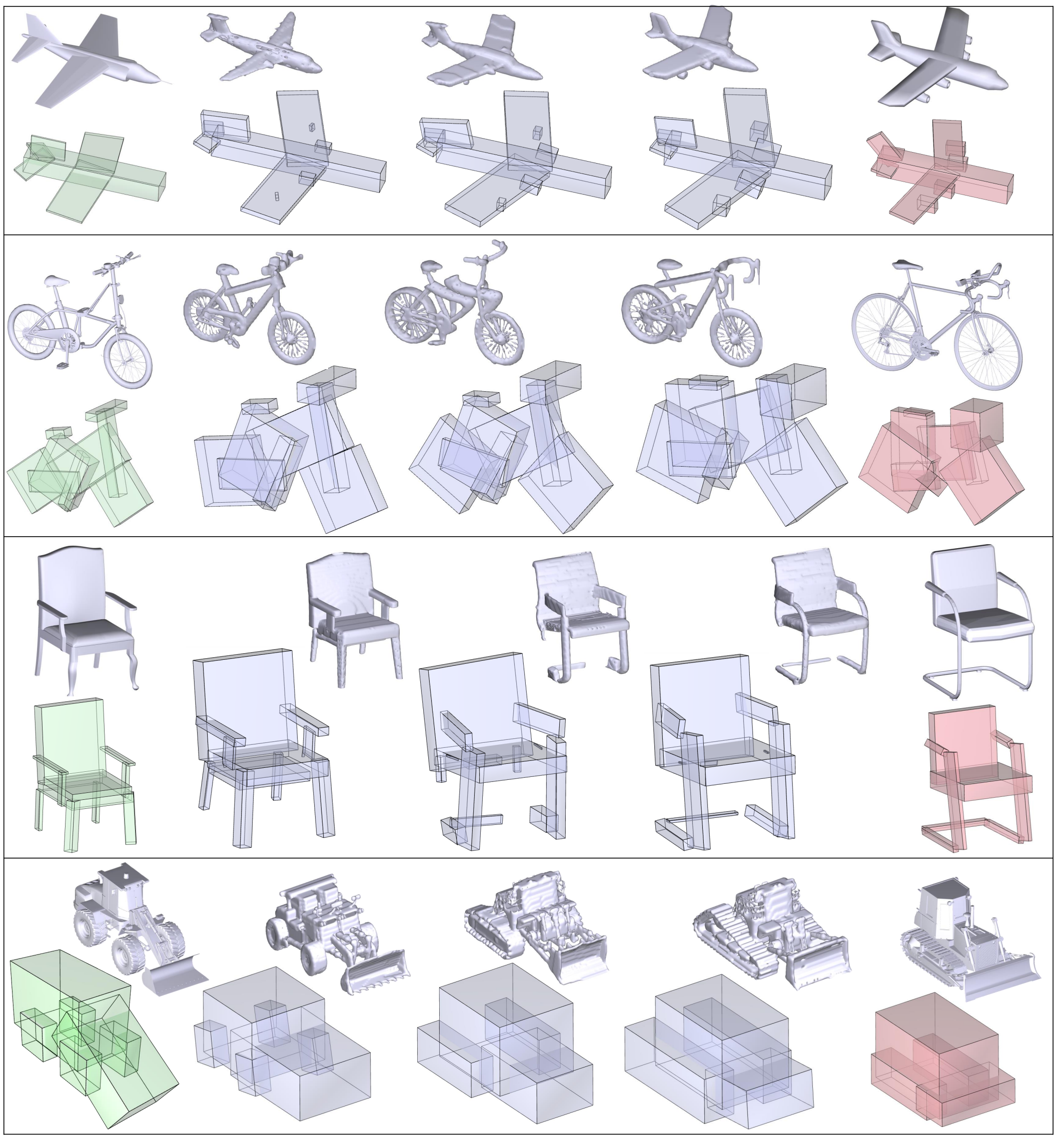


Figure 4.20: *Linear interpolation between root codes, and subsequent synthesis, can result in plausible morphs between shapes with significantly different topologies.*

Chapter 5

3D Shape Recovery from Single RGB Images for Man-made Objects

5.1 Abstract

We propose a novel autoencoder architecture to recover 3D shape from single RGB images for man-made objects. A deep convolutional neural network (VGG-16) is used as an image encoder, which takes a 2D image as input and outputs the latent image features; A recursive neural net is used as a decoder of the latent feature code and maps it back to a set of oriented bounding boxes (OBBs) recursively, organizing by a symmetry hierarchy. These OBBs represents the main structure of the 3D shape being recovered, and the detailed geometry of OBBs is synthesized by a second trained network based on the input image and structural context. Experiments on a variety of image sets show that our method can produce high resolution surface meshes for man-made objects from single view.

This work has not been published yet.

5.2 Introduction

3D shape recovery from 2D images of an object is an important and challenging problem in computer vision, which has widespread applications such as 3D modeling, image editing, object recognition and virtual reality, *etc*. Although many previous methods have achieved good results from multi-view stereo [Hartley and Zisserman, 2003, Seitz et al., 2006], the problem become much more difficult when only monocular images are available. On the other hand, since the majority of images on the internet and in personal photo albums are single views, it has large demand for common users to automatically recover the 3D shape from single RGB images.

3D shape recovery from single view is a highly ill-posed and inherently ambiguous problem, which always need strong assumptions or user interactions in previous methods[Oswald et al., 2009, Prasad et al., 2006]. A later work of [Xue et al., 2011] proposed a depth map recovery method using symmetric and piecewise planar constraint. With the development of large database of 3D models like ShapeNet [Chang et al., 2015], deep learning methods have been investigated for shape recovery. [Liu et al., 2016] developed a convolutional neural network to estimate single-view depth with symmetry enhancement. However, when projecting the depth map to 3D, it greatly suffers from distortions and artifacts. Using volumetric representation, [Girdhar et al., 2016, Wu et al., 2016] predicted 3D grid data from input single RGB images, which captured the coarse volumes of objects. However due to the limited resolution of the voxle grid, which is always only 32x32x32 or 64x64x64, fine-scaled geometry features are rarely recovered by above approaches.

In this paper, we are interested in 3D shape recovery for man-made 3D objects from single views, especially we aim at reconstructing a high quality surface mesh. Actually, man-made objects are always highly structured, such as hierarchical decompositions and nested symmetries. Inspired by the work in Li et al. [2017b], we propose a coarse-to-fine method to recover the 3D shape geometry in two stages. In the first stage, we develop a recursive neural net (RvNN) based autoencoder, mapping an image to a set of oriented bounding boxes (OBBs), organizing by a symmetry hierarchy. The encoder is a deep convolutional neural network (VGG-16) which encodes the input image to a latent feature code. A recursive neural net works as the decoder to convert the latent code to a set of OBBs recursively. In the second stage, to synthesize part geometry in each OBB, we train another convolutional neural network to generate the detailed geometry of OBBs, using the image information and structural context.

Summarizing, our contributions are: *(i)* a new VGG-RvNN based autoencoder for predicting shape structures (arbitrary numbers of OBBs and their relations) from single images; *(ii)* an improved fine-grained geometry synthesis network to generate higher resolution volumes in each OBB, with respect to both the image visual appearance and structural context; *(iii)* experiments show the advantages of our methods in symmetry preserving and detailed geometry.

5.3 Related Work

Our work is related to prior works on single view recovery methods. Early work of shape recovery from single images always used strong assumptions, priors and user interactions.

[Prados and Faugeras, 2005, Wang et al., 2011a] developed the classical technique of shape-from-shading, specifically [Prados and Faugeras, 2005] had a progress on deriving conditions for unique solutions, and [Wang et al., 2011a] focused on how do handle the specular case. Shadows was used to narrow down possible reconstruction in [Yu and Chang, 2002]. Considering texture as an inherent property of an object and , it is possible to infer the geometry from how the texture is deformed in image projection [Hassner and Basri, 2006, Super and Bovik, 1995], assuming that the objects have a regular and known texture. [Delage et al., 2005, Han and Zhu, 2003, Hoiem et al., 2005] explored basic geometric relations to help dissolve ambiguities in the process of shape recovery. Unfortunately, the results of the above methods are usually far away from high quality and fine detailed geometry.

Silhouette and contour edges are always used as salient structures to infer geometry from images [Karpenko et al., 2002, Karpenko and Hughes, 2006]. Curved objects has been investigated in [Oswald et al., 2009, Prasad et al., 2006, Zhang et al., 2001]. However, user interactions are usually required and the methods are limited to certain classes. [Xue et al., 2011] estimated the depth based on symmetric and piecewise planar constraint, and [Liu et al., 2016] proposed a deep learning method to estimate single-view depth with respect to symmetry predictions. The main problem of their approaches is that when projecting depth map to 3D, it suffers from distortions and artifacts and are lack of detailed geometry.

Recent approaches [Girdhar et al., 2016, Wu et al., 2016] predicted 3D volume data from input single RGB images based on volumetric fully convolutional network. However, the resolution of the estimated volume are usually limited to 64x64x64 due to the memory of GPU, therefore fine-grained geometry can not be recovered well at such a low resolution. Unlike predicting the whole volume of a shape, we synthesize geometry in part level. Then the per-box volumes are embedded into a global volume at a higher resolution, from which we reconstruct a high quality surface mesh.

5.4 Structure Recovery

To train the VGG-RvNN based autoencoder, we use image-hierarchy pairs as our training data. Given a input image of an object, we want to firstly recover the main structure of the underlying shape. Our encoder is a deep convolutional neural network (VGG-16), which captures the full information of the input image and outputs a n-D vector code,$n = 80$. A recursive neural net is developed as our decoder, mapping the code to a symmetry hierarchy. The leaf nodes of this hierarchy are a set of oriented bounding boxes (OBBs) and their corresponding symmetry parameters (if they have), from which we can recover a complete shape structure. In this section, we describe how to learn this VGG-RvNN based autoencoder. The training data collection is addressed in detail in Section 5.6.

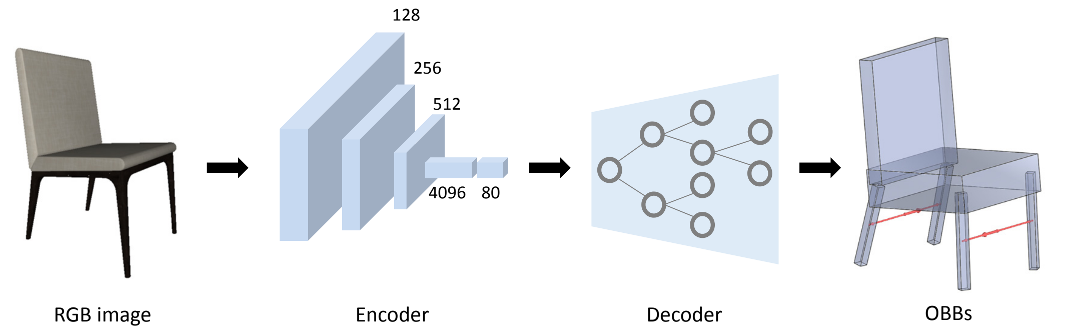


Figure 5.1: *Our network architecture of structure recovery. The encoder is a VGG-16 based neural network and the decoder is a recursive neural net. The OBBs are recovered from the leaf nodes of decoded hierarchy with symmetry cues. The red arrows represent two reflectional symmetries.*

5.4.1 Network Architecture

Our input images have size 256x256, we use VGG-16 as our image encoder and modify the kernel size of the first and second fully connected layer. The original fully connected layer is removed. The output of our image encoder is a 80D code. We use a recursive neural net as our decoder like Li et al. [2017b], recursively decoding the image code to form a hierarchy tree. There are three types of nodes in our hierarchy: leaf node, adjacency node and symmetry node. Each node is decoded by its corresponding decoder respectively :

Adjacency decoder. Decoder AdjDec splits a parent code p back to its child codes c'_1 and c'_2, using the mapping function

$$[c'_1\ c'_2] = \tanh(W_{ad} \cdot p + b_{ad})$$

where $W_{ad} \in \mathbb{R}^{2n \times n}$ and $b_{ad} \in \mathbb{R}^{2n}$.
Symmetry decoder. Decoder SymDec recovers the generator and symmetry parameters as

$$[c'\ s'] = \tanh(W_{sd} \cdot p + b_{sd})$$

where $W_{sd} \in \mathbb{R}^{(n+m)\times n}$, and $b_{sd} \in \mathbb{R}^{m+n}$. We use $m = 8$ for symmetry parameters consisting of symmetry type (1D); number of repetitions for rotational and translational symmetries (1D); and the reflectional plane for reflective symmetry, rotation axis for rotational symmetry, or position and displacement for translational symmetry (6D).
Leaf decoder. Decoder BoxDec converts the code of a leaf node to a 12D box parameters from which an oriented bounding box can be reconstructed.

$$[x'] = \tanh(W_{ld} \cdot p + b_{ld})$$

where $W_{ld} \in \mathbb{R}^{12 \times n}$, and $b_{ld} \in \mathbb{R}$.

During training, the type of each node is known. We also train a node classifier together with the three decoders, which will help to predict node type in testing stage. In our implementation, the classifier and decoders for both adjacency and symmetry are two-layer networks, where the size of the hidden layer and output layer are 200D and 80D respectively.

5.4.2 Learning Loss

Our loss is formulated as the sum of a reconstruction error and a cross entropy loss of node classification. Given an input image, we first encode it using our VGG-16 model and get an image feature code. Starting from the root code, the leaf parameters (box parameters and symmetry parameters) are recovered by recursively applying the decoders AdjDec and SymDec, followed by a final decoding with BoxDec. The reconstruction error is formulated as the sum of squared differences between the input and output parameters for each leaf box and symmetry entry. We scale our models into a unit bounding box to make the reconstruction error between box parameters and symmetry parameter more comparable.

5.5 Detailed Geometry Synthesis

In the previous section, we described how to infer the part layouts of shapes from single view images. However, there is no geometry details in each bounding box so far. To synthesize the detailed part geometry of each leaf bounding box, our solution has three components.

First, we use a fixed-dimensional feature vector capturing both part's box geometry and its context in the recovered symmetry hierarchy. Instead of concatenating the codes of all nodes on the path from leaf node to the root, we get a structure-aware recursive feature (SARF) vector with fixed-dimension by concatenating the codes of the leaf node, its immediate parent node and the root node, same as Li et al. [2017b].

Second, we develop a CNN with five convolutional layers and one fully connected layer, which outputs a code capturing the global understanding of the input image. We call this code as visual code. Unlike Li et al. [2017b], we aim at recover 3D shape from single RGB images, that means the shape geometry should be with respect to the image in a visual manner. SARF code only captures the bounding box feature and the context information, and the visual code from this CNN provide additional information of the object appearance.

Third, we adapt a strategy inspired by [Girdhar et al., 2016] as shown in Fig. 5.2. A deep convolutional autoencoder GeoEnc maps the voxel grid to a compact 32D code, and a

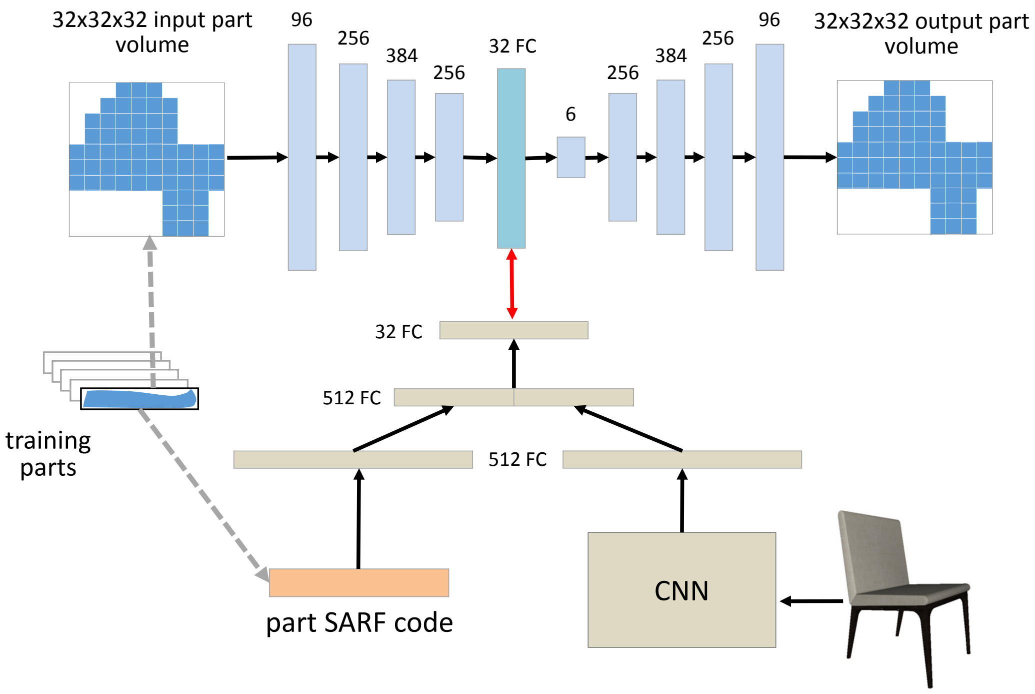


Figure 5.2: *We train a network to synthesize the detailed part geometry base one the SARF code and the visual code.*

decoder GeoDec maps it back to a reconstructed grid. The input of this autoencoder is the part geometry in volume representation. We use a second network GeoMap, consisting of the above CNN and another three fully convolutional layers, to map an SARF code and its corresponding visual code also to a 32D code. We define a Euclidean loss for these two 32D codes and train the whole network jointly. During testing, we use the mapping GeoMap and the decoder GeoDec to synthesize part detailed geometry in each bounding box. As we have 32x32x32 resolution for each part geometry, the overall resolution for the entire shape is much higher. The per-box volumes are then embedded into a global volume, and from the global volumetric data we finally reconstruct a high quality surface mesh.

5.6 Training

To train the VGG-RvNN autoencoder, we calculate the derivative by using backpropagation through structure for the decoder. We use stochastic gradient descent to train both VGG-RvNN autoencoder and our part geometry synthesis network. The collection of our training data is described in the following section.

5.6.1 Training Data

We collected a sub set of models from ShapeNet database, containing three shape categories: chairs (300), bikes (100), aeroplanes (100). Each model is presegmented according to their mesh components or based on the symmetry-aware segmentation utilized in Wang et al. [2011b]. We further augment our training data by component-aware deformation. For each model, we automatically generate 100 variations using the deformation method proposed in Xu et al. [2012]. Texture information is preserved during deformation and will be used to render RGB images for these variation models.

To train the VGG-RvNN based autoencoder, we need generate image-hierarchy pairs as our training data. We use the method in Li et al. [2017b] to infer consistent hierarchy trees for 3D models. More specifically, we train a unsupervised autoencoder to encode the shape structure and then use this autoencoder to do greedy local search to find the "best "hierarchy. From all subsets that are mergeable, we perform two levels of recursive encoding and decoding, and measure the reconstruction error. We add the merge operation with the lowest reconstruction error to the encoding hierarchy. The process repeats until no further merges are possible.

We render 3D shapes in 72 views (from 24 angles and 3 elevations), corresponding with one 3D model and its hierarchy. We use these image-hierarchy pairs to train our VGG-RvNN based autoencoder to infer shape structures from single RGB images.

5.7 Results

We evaluate our method on a variety of image sets, including chairs, planes, bikes. In Fig. 5.3, we show the 3D shape structure recovery results. From the results we can see that our method can approximate the main structure from the input image in a more accurate way. Especially we can also recover the symmetries of the shape from single view, such the reflectional symmetry of the chair legs, the plane wings and the rotational symmetry of the swivel chairs.

The detailed symmetry recovery results are shown in Fig. 5.4. The recovered surface meshes are visually close to the input image, although there are still some local differences. The most advantage of our method is that the symmetries of the shapes are preserved well even for the local detailed geometry. As we synthesize the fine-grained geometry in part level, so the overall resolution is much higher than previous work, resulting in a high quality surface mesh.

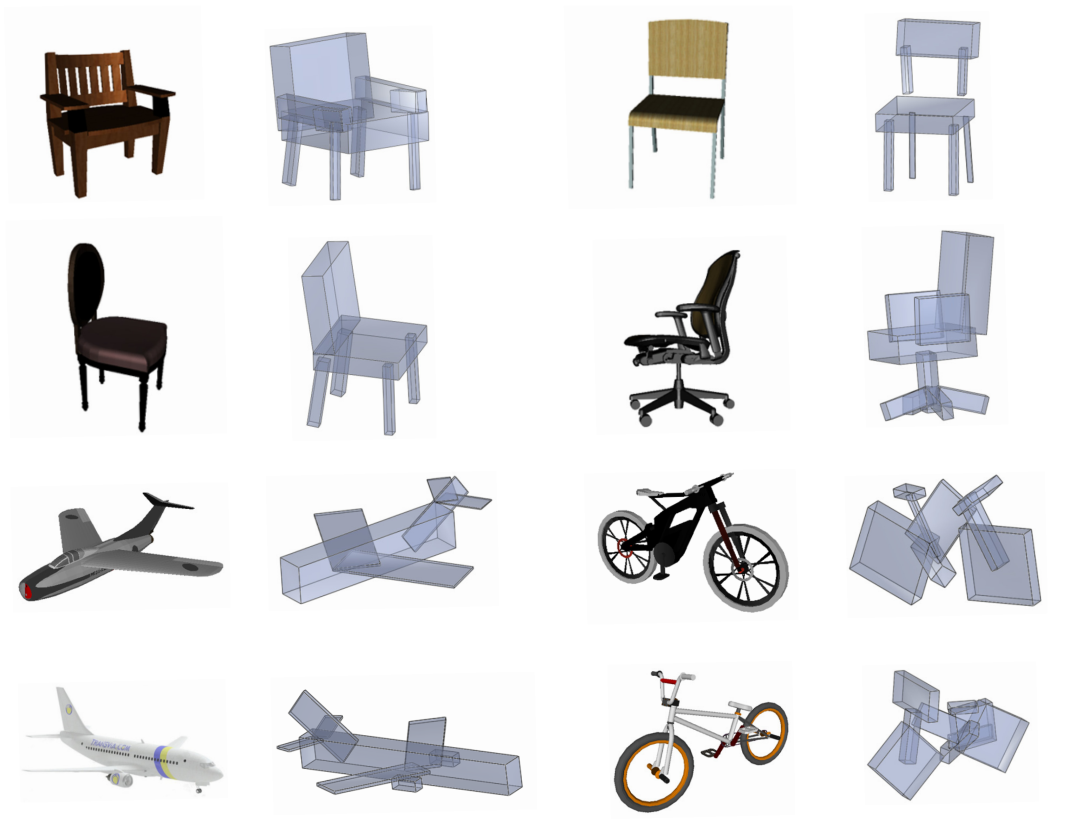


__Figure 5.3:__ The left is the input RGB images, and the right is the inferred 3D shape structures by our VGG-RvNN autoencoder.

5.7.1 Limitations

Our method also has some limitations. For some complex structures or new structures which are not seen in the training data, our method might fail to recover accurately. To overcome this limitation, a larger database is required to train our network. Another limitation is that as we synthesize detailed geometry in part level, sometimes the volumes might not connect to each other very well. Therefore, to avoid user interaction, we may have to retrieve a template from the training data which is most similar with the recovered shape and perform connectivity repairing with the guide of the template.

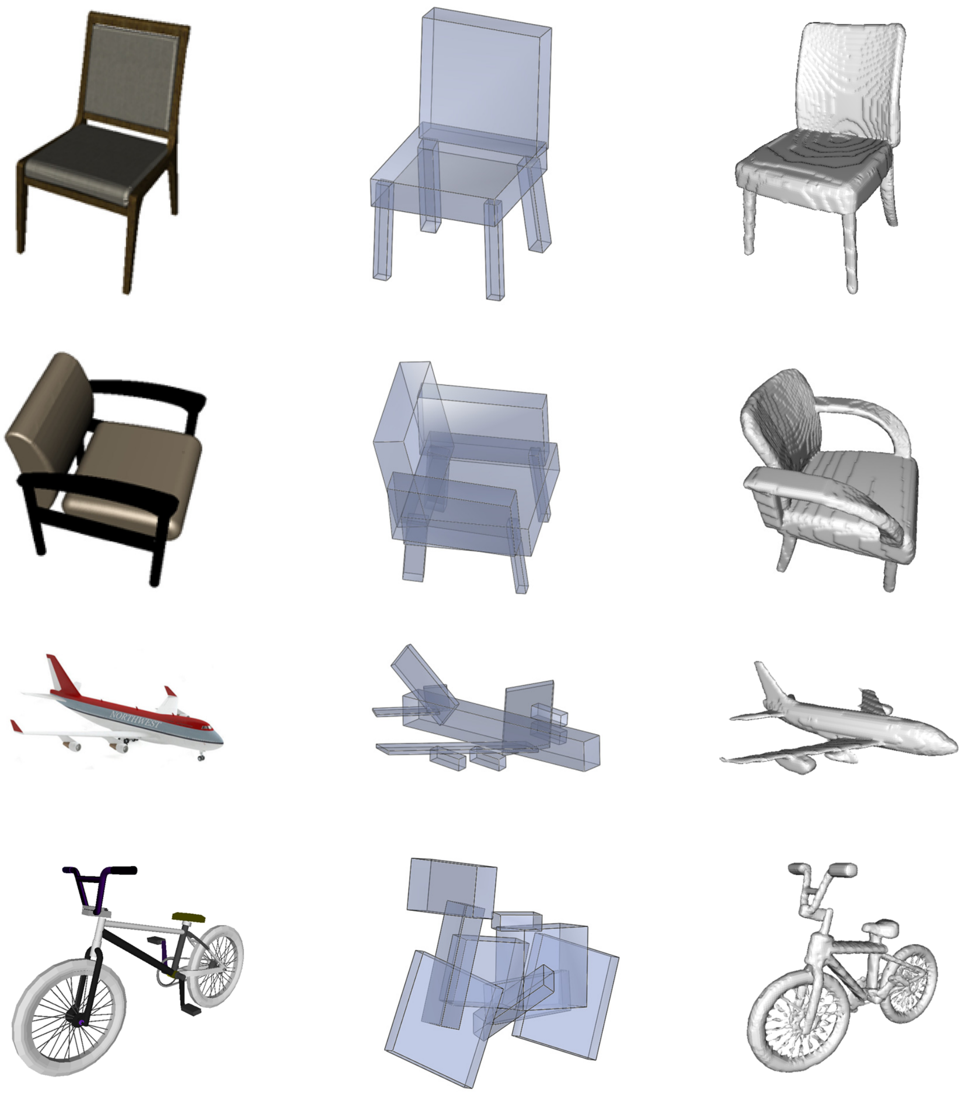


Figure 5.4: The left is the input RGB images, the middle is the inferred 3D shape structures, and the right is the final recovered surface mesh.

5.8 Conclusion

We developed a novel autoencoder architecture for 3D shape geometry recovery from single RGB images of man-made objects. The most difference with previous work is that we can infer a hierarchy from the input image. The leaf nodes in the hierarchy are box parameters and symmetry parameters, from which we can recover a set of oriented bounding boxes representing the main shape structure. The detailed geometry in each bounding box is synthesized by a second trained network. The per-box volumes are then embedded into a global volume, and finally we reconstruct a high quality surface mesh from the global volume data.

We test our method on a variety of image sets, we can infer the shape structure in an accurate way for most cases. In the feature, we would like to investigate strategies about the how to find more consistent hierarchy with semantic information, such as in a hierarchy, chair legs will be grouped before merged with other parts. We believe that with more consistent hierarchies, we can further improve our performance.

Part IV

Conclusion

Chapter 6

Conclusions

3D geometry recovery from single view images is an inherently ill-posed problem as the stereo and multiview technique are impossible to be applied. The problem become even more difficult when fine-scaled 3D geometry has to be inferred accurately. Unlike 2D images, there is no natural parameterization over a regular low-dimensional grid for 3D shapes. People do not have a consensus answer to the question of what is the canonical representation for 3D shapes (voxels, surfaces meshes, or multi-view images).

This dissertation has covered three different approaches for fine-scaled 3D geometry recovery from single RGB images, targeting at facial wrinkles, indoor scenes and man-made objects. Each category has its own particular features, styles and also variations in representation. Therefore, we developed three different strategies to solve the problems respectively.

First, we proposed a non-parametric lightweight approach for 3D facial wrinkle recovery from single RGB images captured by Kinect sensor. We had observed that although the facial wrinkles are different with expressions and people, the variation of their local geometry is much less. Based on this observation, we performed texture synthesis in gradient domain, to copy local geometric patches from the source to synthesize different wrinkles for users. Our method only used one high-quality 3D facial model with calibrated texture as the source in the texture synthesis. User-specific expressions were recorded using the RGB-Depth camera, and then the wrinkles were generated through example-based synthesis of geometric details to construct personalized wrinkled blendshape. Given an RGB-D video captured by a Kinect camera, our method could produce the 3D facial animation with detailed wrinkles for each frame based on these wrinkled blendshapes. Experiments on a variety of facial performance videos showed that our method can produce plausible results, approximating the wrinkles in an accurate way. Low-cost and convenient for common users make our technique even more general for living environments.

Second, we have proposed a fast-to-train multi-streamed CNN architecture to estimate fine-scaled depth maps for indoor scene images. We introduced a set loss jointly over multiple images as a regularizer that minimizes differences in estimates of related images, resulting better accuracy than previous work. Given an indoor scene image, we estimated both the underlying depth and depth gradients. This offered explicit consideration for local detailing which might lost in a single depth estimation loss. Finally, we proposed two methods, one CNN-based and one optimization-based, to fuse the depth and gradient estimates into an accurate and clean depth output with local detailing. Experiments on the NYU Depth v2 dataset showed that our depth predictions are not only competitive with state-of-the-art but also lead to 3D projections that are more accurate and richer with details.

Third, to handle highly structured man-made objects, like chairs, tables, planes, we introduced a novel recursive neural net (RvNN) based autoencoder, mapping a box layout with an arbitrary number into a fixed length code capturing the salient features of a shape. The encoder repeatedly collapsed a pair of code into a merged one in a bottom-up fashion. The final code representing the entire box layout was then decoded to recover the boxes by an inverse process. We extended the RvNN architecture to recover 3D shapes from single images. After encoding the input image to a compact code, we developed an associated decoder to map the code back to a full hierarchy and generated a set of 3D bounding boxes representing the main structure. We synthesized fine-grained part geometry in each bounding boxes by a second trained module finally, and converted the entire volumes to a smooth surface mesh. Experiments showed that our method can generate high quality 3D geometry from single RGB images of man-made objects.

From our investigation of 3D geometry recovery for three different categories, we have learned some insights which may be instructive for the future research.

- Exploring the relations between different estimates. Higher performance usually could be achieved if we carefully explore the relations between different estimates like depth, depth gradients, normals, labels, *etc*. Such relations can be considered as additional constraints or information stream during the training, which is helpful to enhance each other and improve the accuracy.
- Coarse-to-fine approaches are helpful for high resolution outputs. Due to the limited memory of current GPUs, it is hard to use high resolution directly for deep learning approaches. For example, the state-of-the-art of 3D generative methods usually can only produce 32x32x32 voxel grid as output, which is not possible to preserve features and details. Multi-scale and coarse-to-fine approaches are given more and more attentions to produce high resolution outputs, since it is straightforward and can be integrated into the deep learning architecture very well.
- Learning structure of objects like a human. Human would never create a chair with a single armrest or three legs. Why? Because we know the relations between parts to form the real functionality. However machines do not know only if we design

algorithm to learn such structures. Once the structures have been learned, machines would become much more smart to understand the objects in the world and recover or generate them with a structural perspective.

- Different learning strategies depending on your own task. There are different learning methods that may take inputs in different forms like single-view or multi-view images, voxel grids, meshes, boxes, *etc*. We think there is no single method is best but you could choose one depending on your own task. Like for object classification, multi-view images might be a better choice as you can have higher resolution images captured the object information; While when you need to do semantic labeling on mesh surfaces, graph-based convolutional methods are worth to be investigated.

6.1 Feature Work

In future, we plan to explore more relations between different estimates for fine-scaled geometry recovery, such as the relations among depths, normals, gradients, semantic labels and structures like symmetry. The co-occurrence of objects or local features could also be considered as additional information during 3D geometry recovery. We believe that benefiting from such relation explorations, one can further improve the accuracy.

Another potential work is to design more elegant coarse-to-fine framework to enhance the performance or speed the training. A key point in coarse-to-fine approaches is the upsampling strategies from low resolution features to high resolution ones. Properly adding feature fusions between blocks could also boost the convergence.

We also want to investigate using adversarial loss in fine-scaled 3D geometry recovery. That means rather than defining a distance to ground truth by users, we train a discriminator network to judge whether the 3D geometry recovery is good or bad. The discriminator takes image and estimate as input, and implicitly measure the accuracy. This might further help the estimation network to be more smart and improve the performance.

Last but not least, recursive neural network is very powerful to handle irregular input with arbitrary numbers and worth further investigation. A very interesting direction is to study how to use recursive neural network for irregular image inputs, like superpixels or local image patches. Each superpixel is a leaf node in the hierarchy tree which can be group bottom-to-up. Then we can perform 3D geometry recovery top-to down after we get a global understanding of the input image.

List of Figures

List of Tables

Bibliography

Alhashim, I., Li, H., Xu, K., Cao, J., Ma, R., and Zhang, H. (2014). Topology-varying 3D shape creation via structural blending. In *Proc. SIGGRAPH*.

Allen, B., Curless, B., and Popović, Z. (2003). The space of human body shapes: reconstruction and parameterization from range scans. *ACM Transations on Graphics (SIGGRAPH)*, 22(3):587–594.

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. (2005). Scape: shape completion and animation of people. *ACM Transations on Graphics (SIGGRAPH)*, 24(3):408–416.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *arXiv preprint arXiv:1701.07875*.

Averkiou, M., Kim, V., Zheng, Y., and Mitra, N. J. (2014). ShapeSynth: Parameterizing model collections for coupled shape exploration and synthesis. *Eurographics*.

Bansal, A., Russell, B., and Gupta, A. (2016). Marr revisited: 2D-3D alignment via surface normal prediction. In *CVPR*.

Barron, J. T. and Poole, B. (2016). The fast bilateral solver. *ECCV*.

Beeler, T., Bickel, B., Beardsley, P., Sumner, B., and Gross, M. (2010). High-quality single-shot capture of facial geometry. *ACM Transactions on Graphics (SIGGRAPH)*, 29(4):40:1–40:9.

Beeler, T., Hahn, F., Bradley, D., Bickel, B., Beardsley, P., Gotsman, C., Sumner, R. W., and Gross, M. (2011). High-quality passive facial performance capture using anchor frames. *ACM Transactions on Graphics (SIGGRAPH)*, 30(4):75:1–75:10.

Bickel, B., Botsch, M., Angst, R., Matusik, W., Otaduy, M., Pfister, H., and Gross, M. (2007). Multi-scale capture of facial geometry and motion. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3).

Bickel, B., Lang, M., Botsch, M., Otaduy, M. A., and Gross, M. (2008). Pose-space animation and transfer of facial details. SCA '08, pages 57–66.

Blanz, V. and Vetter, T. (1999a). A morphable model for the synthesis of 3D faces. In *Proceedings of SIGGRAPH*, pages 187–194, New York, NY, USA. ACM Press.

Blanz, V. and Vetter, T. (1999b). A morphable model for the synthesis of 3D faces. In *Proc. SIGGRAPH*, pages 187–194.

Bo, L., Shen, C., Dai, Y., van den Hengel, A., and He, M. (2015). Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs. In *CVPR*.

Bokeloh, M., Wand, M., and Seidel, H.-P. (2010). A connection between partial symmetry and inverse procedural modeling. In *Proc. SIGGRAPH*.

Bradley, D., Heidrich, W., Popa, T., and Sheffer, A. (2010). High resolution passive facial performance capture. *ACM Transactions on Graphics (SIGGRAPH)*, 29(4):41:1–41:10.

Cao, C., Weng, Y., Lin, S., and Zhou, K. (2013). 3d shape regression for real-time facial animation. *ACM Transactions on Graphics (SIGGRAPH)*, 32(4):41:1–41:10.

Chakrabarti, A., Shao, J., and Shakhnarovich, G. (2016). Depth from a single image by harmonizing overcomplete local network predictions. In *NIPS*.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An information-rich 3D model repository.

Chaudhuri, S., Kalogerakis, E., Guibas, L., and Koltun, V. (2011). Probabilistic reasoning for assembly-based 3D modeling. In *Proc. SIGGRAPH*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *arXiv preprint arXiv:1606.00915*.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*.

Delage, E., Lee, H., and Ng, A. Y. (2005). *Automatic Single-Image 3d Reconstructions of Indoor Manhattan World Scenes*. Springer.

Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. (2015). FlowNet: learning optical flow with CNNs. In *ICCV*.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. SIGGRAPH '01, pages 341–346. ACM Press.

Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *ICCV*, volume 2, pages 1033–1038.

Eigen, D. and Fergus, R. (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*.

Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *NIPS*.

Ekman, P. and Friesen, W. (1978). *Facial Action Coding System: A Technique for the Measurement of Facial Movement*. Consulting Psychologists Press.

Fish, N., Averkiou, M., van Kaick, O., Sorkine-Hornung, O., Cohen-Or, D., and Mitra, N. J. (2014). Meta-representation of shape families. In *Proc. SIGGRAPH*.

Garg, R., Kumar, B. V., Carneiro, G., and Reid, I. (2016). Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *ECCV*.

Garrido, P., Valgaerts, L., Wu, C., and Theobalt, C. (2013). Reconstructing detailed dynamic face geometry from monocular video. In *ACM Transactions on Graphics (SIGGRAPH ASIA)*, volume 32, pages 158:1–158:10.

Girdhar, R., Fouhey, D. F., Rodriguez, M., and Gupta, A. (2016). Learning a predictable and generative vector representation for objects. In *Proc. Euro. Conf. on Comp. Vis. (ECCV)*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets.

Han, F. and Zhu, S.-C. (2003). Bayesian reconstruction of 3d shapes and scenes from a single image. In *Higher-Level Knowledge in 3D Modeling and Motion Analysis*, pages 12–20.

Hane, C., Ladicky, L., and Pollefeys, M. (2015). Direction matters: Depth estimation with a surface normal classifier. In *CVPR*.

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge Univ. Press.

Hassner, T. and Basri, R. (2006). Example based 3d reconstruction from single 2d images. In *CVPR*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163.

Hoiem, D., Efros, A. A., and Hebert, M. (2005). Automatic photo pop-up. In *SIGGRAPH*.

Huang, H., Chai, J., Tong, X., and Wu, H.-T. (2011). Leveraging motion capture and 3d scanning for high-fidelity facial performance acquisition. *ACM Transactions on Graphics (SIGGRAPH)*, 30(4):74:1–74:10.

Huang, H., Kalogerakis, E., and Marlin, B. (2015). Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. In *Proc. Symp. on Geom. Processing (SGP)*.

Huang, Q., Adams, B., Wicke, M., and Leonidas, J. G. (2008). Non-rigid registration under isometric deformations. In *Proceedings of the Symposium on Geometry Processing*, pages 1449–1457. Eurographics Association.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2016). Image-to-image translation with conditional adversarial networks. In *CVPR*.

Jain, A., Thormählen, T., Ritschel, T., and Seidel, H.-P. (2012). Exploring shape variations by 3D-model decomposition and part-based recombination.

Kalogerakis, E., Chaudhuri, S., Koller, D., and Koltun, V. (2012). A probabilistic model for component-based shape synthesis. 31(4).

Karpenko, O., Hughes, J. F., and Raskar, R. (2002). Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3):585–594.

Karpenko, O. A. and Hughes, J. F. (2006). Smoothsketch: 3d free-form shapes from complex sketches. *ACM Transactions on Graphics*, 25(3):589–598.

Karsch, K., Liu, C., and Kang, S. (2012). Depth extraction from video using non-parametric sampling. In *ECCV*.

Kim, S., Park, K., Sohn, K., and Lin, S. (2016). Unified depth prediction and intrinsic image decomposition from a single image via joint convolutional neural fields. In *ECCV*.

Kim, V. G., Li, W., Mitra, N. J., Chaudhuri, S., DiVerdi, S., and Funkhouser, T. (2013). Learning part-based templates from large collections of 3D shapes. In *Proc. SIGGRAPH*.

Köhler, W. (1929). *Gestalt Psychology*. Liveright.

Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. (2005). Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, 24(3):795–802.

Ladick, L., Shi, J., and Pollefeys, M. (2014). Pulling things out of perspective. In *CVPR*.

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *3DV*.

Larsen, A. B. L., Sønderby, S. K., and Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*.

Li, H., Yu, J., Ye, Y., and Bregler, C. (2013). Realtime facial animation with on-the-fly correctives. *ACM Transactions on Graphics*, 32(4):42:1–42:9.

Li, J., Klein, R., and Yao, A. (2017a). A two-streamed network for estimating fine-scaled depth maps from single rgb images. In *ICCV*.

Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., and Guibas, L. (2017b). Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)*, 36(4):to appear.

Li, J., Xu, W., Cheng, Z., Xu, K., and Klein, R. (2015). Lightweight wrinkle synthesis for 3d facial modeling and animation. *Computer Aided Design*, 58:117–122.

Liang, L., Liu, C., Xu, Y.-Q., Guo, B., and Shum, H.-Y. (2001). Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150.

Liu, B., Gould, S., and Koller, D. (2010). Single image depth estimation from predicted semantic labels. In *CVPR*.

Liu, F., Shen, C., and G. Lin, I. D. R. (2015). Learning depth from single monocular images using deep convolutional neural fields. In *TPAMI*.

Liu, G., Yang, C., Li, Z., Ceylan, D., and Huang, Q. (2016). Symmetry-aware depth estimation using deep neural networks. *arXiv preprint arXiv:1604.06079*.

Liu, M., Salzmann, M., and He, X. (2014). Discrete-continuous depth estimation from a single image. In *CVPR*.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *CVPR*.

Ma, W.-C., Jones, A., Chiang, J.-Y., Hawkins, T., Frederiksen, S., Peers, P., Vukovic, M., Ouhyoung, M., and Debevec, P. (2008). Facial performance synthesis using deformation-driven polynomial displacement maps. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 27(5):121:1–121:10.

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*.

Mitra, N., Wand, M., Zhang, H., Cohen-Or, D., and Bokeloh, M. (2013). Structure-aware shape processing. In *Eurographics State-of-the-art Report (STAR)*.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2006). Procedural modeling of buildings. In *Proc. SIGGRAPH*.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *Proc. ICML*.

Oswald, M. R., Töppe, E., Kolev, K., and Cremers, D. (2009). Non-parametric single view reconstruction of curved objects using convex optimization. In *DAGM*.

Ovsjanikov, M., Li, W., Guibas, L., and Mitra, N. J. (2011). Exploration of continuous variability in collections of 3D shapes. In *Proc. SIGGRAPH*.

Prados, E. and Faugeras, O. (2005). Shape from shading: a well-posed problem? In *CVPR*.

Prasad, M., Zisserman, A., and Fitzgibbon, A. (2006). Single view reconstruction of curved surfaces. In *CVPR*.

Qi, C. R., Su, H., Niessner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view CNNs for object classification on 3D data. In *Proc. IEEE Conf. on CVPR*.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.

Roy, A. and Todorovic, S. (2015). Monocular depth estimation using neural regression forest. In *ICCV*.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. In *International Conference on 3D Digital Imaging and Modeling*, pages 145–152. IEEE Computer Society.

Saragih, J., Lucey, S., and Cohn, J. (2011). Real-time avatar animation from a single image. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 117–124, Santa Barbara, CA, USA. IEEE Computer Society.

Saxena, A., Chung, S., and Ng, A. (2005). Learning depth from single monocular images. In *NIPS*.

Saxena, A., Sun, M., and Ng, A. Y. (2008). Make3d: Learning 3-d scene structure from a single still image. *TPAMI*.

Schulz, A., Shamir, A., Baran, I., Levin, D. I. W., Sitthi-Amorn, P., and Matusik, W. (2016). Retrieval on parametric shape collections. *ACM Trans. Graph. (to appear)*.

Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. IEEE Conf. on CVPR*.

Serre, T. (2013). Hierarchical models of the visual system. In Jaeger, D. and Jung, R., editors, *Encyclopedia of Computational Neuroscience*. Springer NY.

Silberman, N., Kohli, P., Hoiem, D., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *ECCV*.

Simard, P. and Denker, J. (1992). Tangent prop-a formalism for specifying selected invariances in an adaptive net. In *NIPS*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Sinha, A., Bai, J., and Ramani, K. (2016). Deep learning 3d shape surfaces using geometry images. In *Proc. Euro. Conf. on Comp. Vis. (ECCV)*.

Socher, R. (2014). *Recursive Deep Learning for Natural Language Processing and Computer Vision*. PhD thesis, Stanford University.

Socher, R., Huval, B., Bhat, B., Manning, C. D., and Ng, A. Y. (2012). Convolutional-recursive deep learning for 3D object classification.

Socher, R., Lin, C. C., Ng, A. Y., and Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proc. ICML*.

Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3D shape recognition. In *Proc. Int. Conf. on Comp. Vis. (ICCV)*.

Super, B. J. and Bovik, A. C. (1995). Shape from texture using local spectral moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:333–343.

Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., and Měch, R. (2012). Learning design patterns with Bayesian grammar induction. In *Proc. UIST*, pages 63–74.

Talton, J. O., Gibson, D., Yang, L., Hanrahan, P., and Koltun, V. (2009). Exploratory modeling with collaborative design spaces.

Torralba, A. and Oliva, A. (2002). Depth estimation from image structure. *TPAMI*.

Töppe, E., Oswald, M. R., Cremers, D., and Rother, C. (2010). Image-based 3d modeling via cheeger sets. In *ACCV*.

Tulsiani, S., Su, H., Guibas, L. J., Efros, A. A., and Malik, J. (2017). Learning shape abstractions by assembling volumetric primitives. In *Proc. IEEE Conf. on CVPR*.

Valgaerts, L., Wu, C., Bruhn, A., Seidel, H.-P., and Theobalt, C. (2012). Lightweight binocular facial performance capture under uncontrolled lighting. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 31(6):187:1–187:11.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. *CoRR*, abs/1601.06759.

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016c). Conditional image generation with PixelCNN decoders. *CoRR*, abs/1606.05328.

van Kaick, O., Xu, K., Zhang, H., Wang, Y., Sun, S., Shamir, A., and Cohen-Or, D. (2013). Co-hierarchical analysis of shape structures. In *Proc. SIGGRAPH*.

Vicon (2012). Vicon homepage. http://www.vicon.com/.

Vlasic, D., Brand, M., Pfister, H., and Popović, J. (2005). Face transfer with multilinear models. *ACM Transactions on Graphics*, 24(3):426–433.

Wang, G., Su, W., and Song, Y. (2011a). A new shape from shading approach for specular surfaces. In *International Conference on Artificial Intelligence and Computational Intelligence*, pages 71–78.

Wang, P., Shen, X., Lin, Z., Cohen, S., Price, B., and Yuille, A. (2015). Towards unified depth and semantic prediction from a single image. In *CVPR*.

Wang, Y., Xu, K., Li, J., Zhang, H., Shamir, A., Liu, L., Cheng, Z., and Xiong, Y. (2011b). Symmetry hierarchy of man-made objects. In *EG*.

Wei, L.-Y. and Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. SIGGRAPH '00, pages 479–488, New York, NY, USA. ACM Press.

Weise, T., Bouaziz, S., Li, H., and Pauly, M. (2011). Realtime performance-based facial animation. *ACM Transactions on Graphics (SIGGRAPH)*, 30(4):77:1–77:10.

Weise, T., Li, H., Van Gool, L., and Pauly, M. (2009). Face/off: Live facial puppetry. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–16. ACM Press.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. (2016). Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3D ShapeNets: A deep representation for volumetric shapes. In *Proc. IEEE Conf. on CVPR*.

Xu, K., Zhang, H., Cohen-Or, D., and Chen, B. (2012). Fit and diverse: Set evolution for inspiring 3D shape galleries. *ACM Trans. on Graphics*, 31(4):57:1–10.

Xue, T., Liu, J., , and Tang, X. (2011). Symmetric piecewise planar object reconstruction from a single image. In *CVPR*.

XYZRGB (2012). Xyz rgb system. http://www.xyzrgb.com.

Yan, X., Yang, J., Yumer, E., Guo, Y., and Lee, H. (2016). Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision.

Yu, Y. and Chang, J. T. (2002). Shadow graphs and surface reconstruction. In *ECCV*, pages 31–45.

Yumer, M. E. and Mitra, N. J. (2016). Learning semantic deformation flows with 3D convolutional networks. In *Proc. Euro. Conf. on Comp. Vis. (ECCV)*.

Zhang, L., Dugas-Phocion, G., Samson, J. S., and Seitz, S. M. (2001). Singleview modelling of free-form scenes. In *CVPR*.

Zhang, L., Snavely, N., Curless, B., and Seitz, S. M. (2004). Spacetime faces: high resolution capture for modeling and animation. *ACM Transactions on Graphics (SIGGRAPH)*, 23(3):548–558.

Zhuo, W., Salzmann, M., He, X., and Liu, M. (2015). Indoor scene structure analysis for single image depth estimation. In *CVPR*.

Appendix:
Precedence rules for symmetry hierarchy

We reproduce the precedence rules stipulated in Wang et al. 2011b for sorting symmetry grouping and assembly operations:

M1 (Grouping before assembly): Grouping by symmetry takes precedence over assembly operations, with an exception given by the next rule (**M**2).

M2 (Assembly before grouping): Assemble before grouping if and only if the assembled nodes belong to symmetry cliques which possess equivalent grouping symmetries.

G1 (Clique order): If there are still symmetry cliques of order greater than two in the contraction graph, then higher-order cliques are grouped before lower-order ones.

G2 (Reflectional symmetry): If there are only order-2 cliques in the graph, then group by reflectional symmetry before rotational symmetry and translational symmetries.

G3 (Proximity in symmetry clique): If **G**1 and **G**2 cannot set a precedence, e.g., between rotational and translational symmetries of the same order, then grouping of part ensembles closer in proximity takes precedence.

A1 (Symmetry preservation): Symmetry-preserving assembly takes precedence over symmetry-breaking assembly.

A2 (Connectivity strength): If **A**1 cannot set a precedence, then order assembly operations according to a geometric *connectivity strength* measure.