DEEP-LEARNED GENERATIVE REPRESENTATIONS OF 3D SHAPE FAMILIES

A Dissertation Presented

by

HAIBIN HUANG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2017

College of Information and Computer Sciences

DEEP-LEARNED GENERATIVE REPRESENTATIONS OF 3D SHAPE FAMILIES

A Dissertation Presented

by

HAIBIN HUANG

Approved as to style and content by:

Evangelos Kalogerakis, Chair

Rui Wang, Member

Benjamin Marlin, Member

Benjamin Jones, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

To my grandma.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor, Prof. Evangelos Kalogerakis, for his help and advice during my Ph.D. study. The past five years have been the most intense and successful times in my career. We have worked together on many research projects and it won't be possible to achieve these results without your support and vigilance. Thank you!

I thank Prof. Rui Wang who offered me the opportunity to join UMass and gave valuable suggestions during my study. I thank my other defense committee members, Prof. Benjamin Marlin and Prof.Benjamin Jones for their insightful feedbacks during my thesis writing.

Furthermore, I would like to acknowledge my close collaborators: Li-Yi Wei for the first SIGGRAPH project I joined, Chongyang Ma for the unforgettable time during our style transfer project. My thanks also go to : Alla Sheffer, Siddhartha Chaudhuri, Xiaoguang Han, Zhen Li, Yizhou Yu, Amir Arsalan Soltani, Jiajun Wu, Tejas Kulkarni, Joshua Tenenbaum, Changqing Zou, Marie-Paule Cani, Ping Tan, Hao Zhang. I am very lucky to work with so many talented researchers and learn from them. Thanks for your excellent collaborative effort and working with you has been a wonderful experience and a great source of inspiration.

A very special thank you to Adobe Research and Autodesk Research. I would like to acknowledge my manager Radomir Mech and Adrian Butscher for offering these great opportunities. My thanks go to my mentors, M. Ersin Yumer, Duygu Ceylan and Vladimir Kim for the exciting project ideas and guidances. The one year I had in San Jose and Toronto is one of the best times during my Ph.D., thank you for making that happen.

I also would like to acknowledge my lab mates and friends for the wonderful times at UMass: Zhaoliang Lun, Yahan Zhou, Hang Su, Huaizu Jiang, Chenyun Wu, SouYoung Jin, Chang Liu, Jingyi Guo, Yue Wang, Dan Zhang, Kun Tu, Weize Kong, Bo Jiang, Tian Guo, Tao Sun, Zhaoyu Gao, Tianbo Gu and many more that I cannot list all their names here. It has been a great time with you in this beautiful town. I wish you all the best for future endeavors.

Finally, I want to take this opportunity to thank my parents for their support in whatever path I chose to pursue. It would be impossible for me to finish my Ph.D. study without your encouraging and love from the other side of the ocean.

My research is under support from the NSF grants CHS-1422441 and CHS-1617333. Part of the experiments was performed in the UMass GPU cluster obtained under a grant from the Collaborative R&D Fund managed by the Massachusetts Technology Collaborative

ABSTRACT

DEEP-LEARNED GENERATIVE REPRESENTATIONS OF 3D SHAPE FAMILIES

SEPTEMBER 2017

HAIBIN HUANG

B.Sc., ZHEJIANG UNIVERSITY, ZHEJIANG, CHINA

M.Sc., ZHEJIANG UNIVERSITY, ZHEJIANG, CHINA

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Evangelos Kalogerakis

Digital representations of 3D shapes are becoming increasingly useful in several emerging applications, such as 3D printing, virtual reality and augmented reality. However, traditional modeling softwares require users to have extensive modeling experience, artistic skills and training to handle their complex interfaces and perform the necessary low-level geometric manipulation commands. Thus, there is an emerging need for computer algorithms that help novice and casual users to quickly and easily generate 3D content. In this work, I will present deep learning algorithms that are capable of automatically inferring parametric representations of shape families, which can be used to generate new 3D shapes from high-level user specifications, such as input sketches.

I will first present a deep learning technique that generates 3D shapes by translating an input sketch to parameters of a predefined procedural model. The inferred procedural

model parameters then yield multiple, detailed output shapes that resemble the user's input sketch. At the heart of our approach is a deep convolutional network trained to map sketches to procedural model parameters.

Procedural models are not readily available always, thus I will present a deep learning algorithm that is capable of automatically learning parametric models of shape families from 3D model collections. The parametric models are built from dense point correspondences between shapes. To compute correspondences, we propose a probabilistic graphical model that learns a collection of deformable templates that can describe a shape family. The probabilistic model is backed by a deep convolutional network that learns surface point descriptors such that accurate point correspondences are established between shapes.

Based on the estimated shape correspondence, I will introduce a probabilistic generative model that hierarchically captures statistical relationships of corresponding surface point positions and parts as well as their existence in the input shapes. A deep learning procedure is used to capture these hierarchical relationships. The resulting generative model is used to produce control point arrangements that drive shape synthesis by combining and deforming parts from the input collection.

With these new data driven modeling algorithms, I hope to significantly shorten the design cycle of 3D products and let detail-enriched visual content creation become easy for casual modelers.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

3D content creation is becoming increasingly important in several emerging applications, such as 3D printing, collaborative virtual environments, simulation, augmented reality, computer-aided design, digital entertainment and architecture, to name a few. Despite decades of advances in graphics research, it remains a big challenge to provide tools that are simple to use, easy to learn, enable rapid prototyping, generate detailed outputs, and encourage interactive exploration of design alternatives. Unlike traditional media 2D images and videos, 3D modeling software such as 3DSMax, Blender, Maya, and ZBrush require users to have extensive modeling experience, artistic skills and training to handle their complex interfaces and perform the necessary low-level geometric manipulation commands. For novice and casual users, such tools have a steep learning curve.

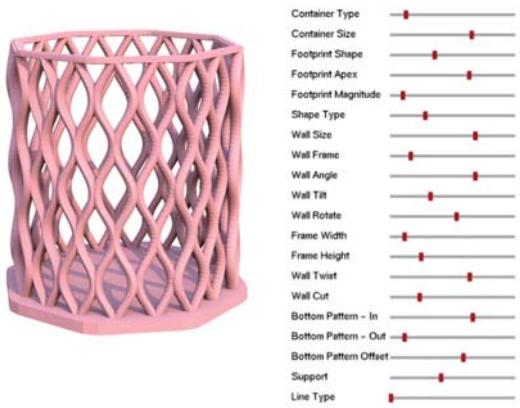

Figure 1.1: An example of procedural model for creating containers. Users can create 3D containers by adjusting the parameters of the procedural model

To make modeling easier for novice users, one possibility is to use procedural modeling techniques. In procedural modeling techniques, experts specify parametric rule sets able to produce high-quality shapes, as shown in Figure 1.1. Users can then create detailed 3D models through adjusting parameters instead of manipulating low-level geometry. However, controlling the outputs of these techniques for design purposes is often notoriously difficult for users due to the large number of parameters involved in these rule sets and also their non-linear relationship to the resulting content. Moreover, hand-designed parametric models are not always readily available for several 3D man-made shape families. Hand-engineering such rules is time-consuming and cumbersome even for experts. They are are also limited to produce certain shape variations (i.e., those that can be created from the hand-designed rules).

To circumvent these problems, I will first present a deep learning technique that generates 3D shapes by translating an input sketch to parameters of a predefined procedural model. The inferred procedural model parameters then yield multiple, detailed output shapes that resemble the user's input sketch. Then I will present a method that is capable of automatically inferring parametric representations of 3D shape families. Our method first learns part-based templates such that an optimal set of fuzzy point and part correspondences is computed between the shapes of an input collection based on a probabilistic deformation model. Based on the estimated shape correspondence, our method also learns a probabilistic generative model that hierarchically captures statistical relationships of corresponding surface point positions and parts as well as their existence in the input shapes. A deep learning procedure is used to capture these hierarchical relationships. The resulting generative model is used to produce control point arrangements that drive shape synthesis by combining and deforming parts from the input collection.

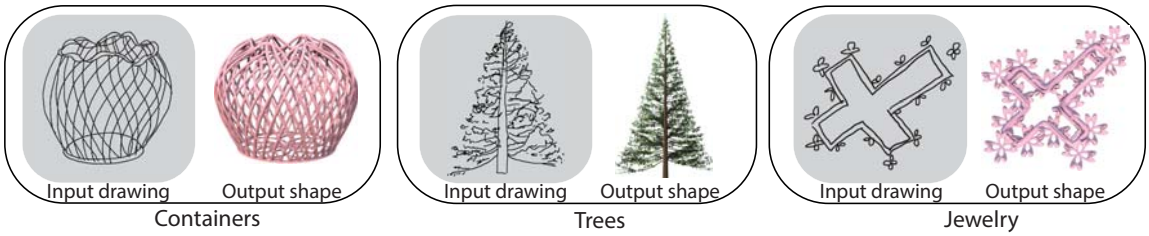

Figure 1.2: Given freehand sketches drawn by casual modelers, our method learns to synthesize procedural model parameters that yield detailed output shapes.

1.2 Mapping sketches to predefined procedural models

The first part of this thesis is an approach that simplifies the modeling process for novices by mapping sketches to predefined procedural models. Procedural modeling techniques can produce high quality visual content through complex rule sets. However, controlling the outputs of these techniques for design purposes is often notoriously difficult for users due to the large number of parameters involved in these rule sets and also their non-linear relationship to the resulting content. To circumvent this problem, we present a sketch-based approach to procedural modeling. Given an approximate and abstract hand-drawn 2D sketch provided by a user, our algorithm automatically computes a set of procedural model parameters, which in turn yield multiple, detailed output shapes that resemble the user's input sketch. The user can then select an output shape, or further modify the sketch to explore alternative ones. At the heart of our approach is a deep Convolutional Neural Network (CNN) that is trained to map sketches to procedural model parameters. The network is trained by large amounts of automatically generated synthetic line drawings. By using an intuitive medium i.e., freehand sketching as input, users are set free from manually adjusting procedural model parameters, yet they are still able to create high quality content. We demonstrate the accuracy and efficacy of our method in a variety of procedural modeling scenarios including design of man-made and organic shapes. Examples of input sketches and output shapes are shown in Figure 1.2.

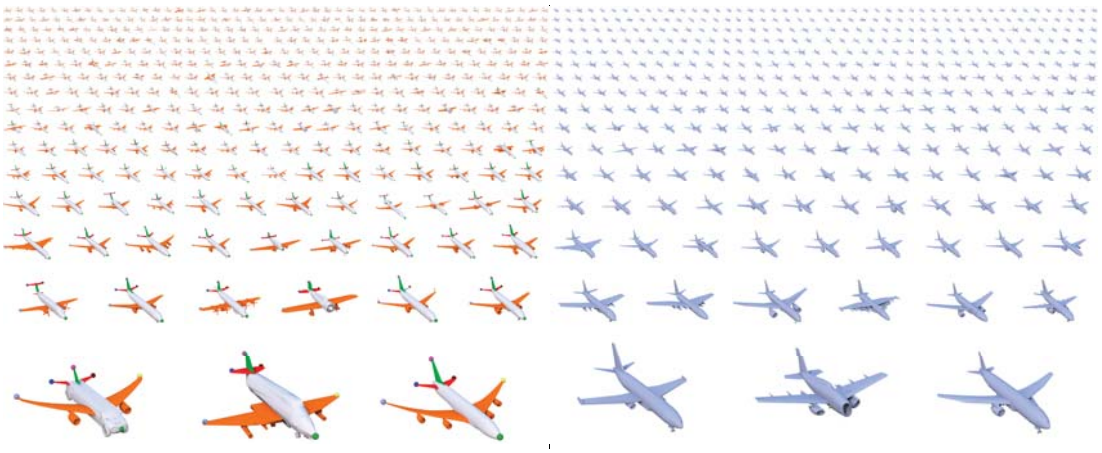

Figure 1.3: Given a collection of 3D shapes, we train a probabilistic model that performs joint shape analysis and synthesis. (Left) Semantic parts and corresponding points on shapes inferred by our model. (Right) New shapes synthesized by our model.

1.3 Learning parametric models

The second part of this thesis is an algorithm to learn parametric models from geometrically diverse 3D shape families where there are no predefined parametric models. We represent each 3D model as a point cloud and our algorithm parameterizes shape families in terms of corresponding point positions across shapes. Specifically, the method learns part-based templates such that an optimal set of fuzzy point and part correspondences is computed between the shapes of an input collection based on a probabilistic deformation model. In contrast to previous template-based approaches, the geometry and deformation parameters of our part-based templates are learned from scratch.

The probabilistic deformation model is backed by a deep convolutional neural network that learns local surface descriptors for 3D shapes. We developed a new local descriptor for 3D shapes, that are applicable to a wide range of shape analysis problems such as point correspondences, semantic segmentation, affordance prediction, and shape-to-scan matching, as shown in Figure 1.4. The descriptor is produced by a convolutional network that is trained to embed geometrically and semantically similar points close to one another in descriptor space. The network processes surface neighborhoods around points on a shape that

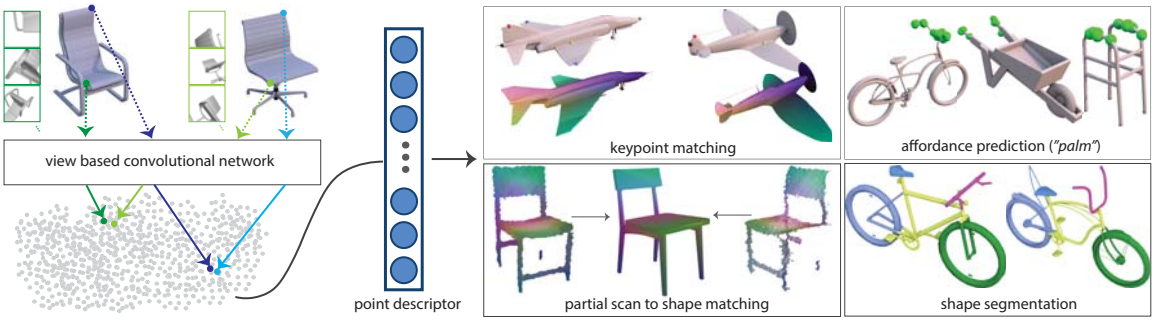

Figure 1.4: We present a view-based convolutional network that produces local, point-based shape descriptors. The network is trained such that geometrically and semantically similar points across different 3D shapes are embedded close to each other in descriptor space (left). Our produced descriptors are quite generic - they can be used in a variety of shape analysis applications, including dense matching, prediction of human affordance regions, partial scan-to-shape matching, and shape segmentation (right).

are captured at multiple scales by a succession of progressively zoomed out views, taken from carefully selected camera positions. our network effectively encodes multi-scale local context and fine-grained surface detail. The learned descriptor is suitable for establishing accurate point correspondence between shapes with large variations in geometry and shape structures.

We then build a probabilistic deformation model based on the learned local shape descriptor that jointly estimates fuzzy point correspondences and part segmentations of shapes. Examples of learned point correspondence and segmentation are shown in Figure 1.3. The learned templates give us a parametric model for the input shape families, controlled by point correspondences and existence.

1.4 Shape synthesis via the learned parametric models

In the third part of this thesis, I will describe a generative probabilistic model whose goal is to characterize surface variability within a shape family. Based on the estimated shape correspondence, our method learns a probabilistic generative model that hierarchically captures statistical relationships of corresponding surface point positions and parts as

well as their existence in the input shapes. A deep learning procedure is used to capture these hierarchical relationships. The resulting generative model is used to produce control point arrangements that drive shape synthesis by combining and deforming parts from the input collection. Examples of synthesized new shapes are shown in Figure 1.3. Further more, the generative model also yields compact shape descriptors that are used to perform fine-grained classification. It can be also coupled with the probabilistic deformation model to further improve shape correspondence. By jointly training the probabilistic deformation model and generative probabilistic model, we demonstrate correspondence and segmentation results than previous state-of-the-art approaches.

1.5 Contribution

This thesis is focused on the study of 3D shape modeling and 3D shape synthesis. With these new modeling algorithms, I hope to significantly shorten the design cycle of 3D products and make it easy for users to create complex and plausible shapes. It consists of the following technical contributions:

- A procedural modeling technique for 2D/3D shape design using human sketches as input, without the need for direct, manual parameter editing.
- A method to generate input sketches from procedural models simulating aspects of simplifications and exaggerations found in human sketches, making our system scalable and improving the generalization ability of machine learning algorithms to process human sketches.
- A new point-based feature descriptor for general 3D shapes, directly applicable to a wide range of shape analysis tasks, that is sensitive to both fine-grained local information and context.
- A massive synthetic dataset of corresponding point pairs for training deep learning methods for 3D shape analysis.

- A probabilistic deformation model that estimates fuzzy point and part correspondences within structurally and geometrically diverse shape families. Our method learns the geometry and deformation parameters of templates within a fully probabilistic framework to optimally achieve these tasks instead of relying on fixed primitives or pre-existing shapes.

- A deep-learned probabilistic generative model of 3D shape surfaces that can be used to further optimize shape correspondences, synthesize surface point arrangements, and produce compact shape descriptors for fine-grained classification.

CHAPTER 2

SHAPE SYNTHESIS FROM SKETCHES VIA CONVOLUTIONAL NETWORKS AND PREDEFINED PARAMETRIC MODELS

Procedural Modeling (PM) allows synthesis of complex and non-linear phenomena using conditional or stochastic rules [91, 137, 84]. A wide variety of 2D or 3D models can be created with PM e.g., vases, jewelry, buildings, trees, to name a few [117]. PM frees users from direct geometry editing and helps them to create a rich set of unique instances by manipulating various parameters in the rule set. However, due to the complexity and stochastic nature of rule sets, the underlying parametric space of PM is often very high-dimensional and nonlinear, making outputs difficult to control through direct parameter editing. PM is therefore not easily approachable by non-expert users, who face various problems such as where to start in the parameter space and how to adjust the parameters to reach outputs that match their intent. We address this problem by allowing users to perform PM through freehand sketching rather than directly manipulating high-dimensional parameter spaces. Sketching is often a more natural, intuitive and simpler way for users to communicate their intent.

We introduce a technique that takes 2D freehand sketches as input and translates them to corresponding PM parameters, which in turn yield detailed shapes[1]. The users of our technique are not required to have artistic and professional skills in drawing. We aim to synthesize PM parameters from approximate, abstract sketches drawn by casual modelers who are interested in quickly conveying design ideas. A main challenge in recognizing

[1]This work was published in IEEE Transactions on Visualization and Computer Graphics 2017. Source code and more results are available on our project page: `http://people.cs.umass.edu/~hbhuang/publications/srpm/`

such sketches and converting them to high quality visual content is the fact that humans often perform dramatic abstractions, simplifications and exaggerations to convey the shape of objects [28]. Developing an algorithm that factors out these exaggerations, is robust to simplifications and approximate line drawing, and captures the variability of all possible abstractions of an object is far from a trivial task. Hand-designing an algorithm and manually tuning all its internal parameters or thresholds to translate sketch patterns to PM outputs seems extremely hard and unlikely to handle the enormous variability of sketch inputs.

We resort to a *machine learning* approach that automatically learns the mapping from sketches to PM parameters from a large corpus of training data. Collecting training human sketches relevant to given PM rule sets is hard and time-consuming. We instead automatically generate *synthetic training data* to train our algorithm. We exploit key properties of PM rule sets to generate the synthetic datasets. We simplify PM output shapes based on structure, density, repeated patterns and symmetries to simulate abstractions and simplifications found in human sketches. Given the training data, the mapping from sketches to PM parameters is also far from trivial to learn. We found that common classifiers and regression functions used in sketch classification and sketch-based retrieval [29, 109], such as Support Vector Machines, Nearest Neighbors or Radial Basis Function interpolation, are largely inadequate to reliably predict PM parameters. We instead utilize a *deep Convolutional Neural Network* (CNN) architecture to map sketches to PM parameters. CNNs trained on large datasets have demonstrated large success in object detection, recognition, and classification tasks [27, 37, 99, 24, 124]. Our key insight is that CNNs are able to capture the complex and nonlinear relationships between sketches and PM parameters through their hierarchical network structure and learned, multi-resolution image filters that can be optimized for PM parameter synthesis.

Since the input sketches represent abstractions and simplifications of shapes, they often cannot be unambiguously mapped to a single design output. Through the CNN, our algo-

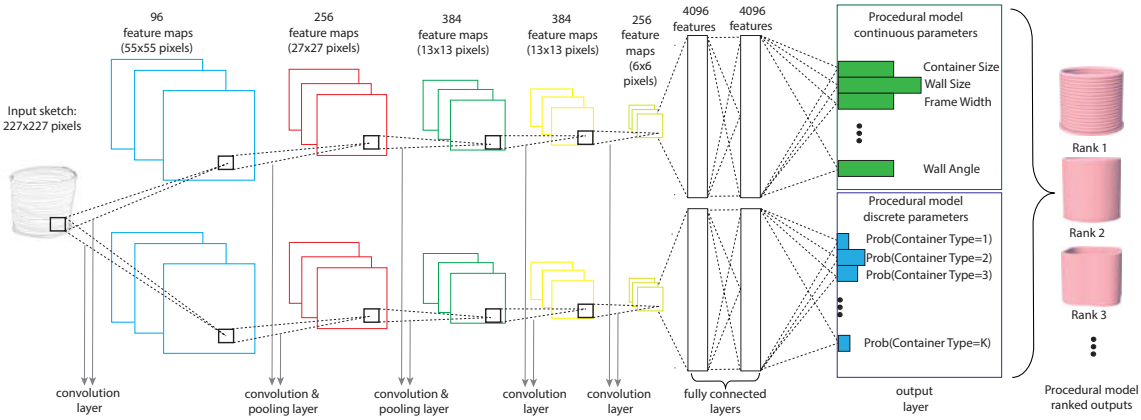

Figure 2.1: Convolutional Neural Network (CNN) architecture used in our method. The CNN takes as input a sketch image and produces a set of PM parameters, which in turn yield ranked design outputs.

rithm provides *ranked, probabilistic outputs*, or in other words *suggestions* of shapes, that users can browse and select the ones they prefer most.

2.1 Related Work

Our work is related to prior work in procedural modeling, in particular targeted design of procedural models and exploratory procedural modeling techniques, as well as sketch-based shape retrieval and convolutional neural networks we discuss in the following paragraphs.

Procedural Modeling. Procedural models were used as early as in sixties for biological modeling based on L-systems [69]. L-systems were later extended to add geometric representations [93], parameters, context, or environmental sensitivity to capture a wide variety of plants and biological structures [94, 79]. Procedural modeling systems were also used to generate shapes with shape grammars [123, 71], for modeling cities[91], buildings [84], furniture arrangements [36], building layouts [80], and lighting design [111].

Procedural models often expose a set of parameters that can control the resulting visual content. Due to the recursive nature of the procedural model, some of those parameters often have complex, aggregate effects on the resulting geometry. On one hand, this is

an advantage of procedural models i.e., an unexpected result emerges form a given set of parameters. On another hand, if a user has a particular design in mind, recreating it using these parameters results in a tedious modeling experience. To address this problem, there has been some research focused on targeted design and exploratory systems to circumvent the direct interaction with PM parameters.

Targeted design of procedural models. Targeted design platforms free users from interacting with PM parameters. Lintermann and Deussen [70] presented an interactive PM system where conventional PM modeling is combined with free-form geometric modeling for plants. McCrae and Singh [77] introduced an approach for converting arbitrary sketch strokes to 3D roads that are automatically fit to a terrain. Vanegas et al. [133] perform inverse procedural modeling by using Monte Carlo Markov Chains (MCMC) to guide PM parameters to satisfy urban design criteria. Stava et al. [122] also use a MCMC approach to tree design. Their method optimizes trees, generated by L-systems, to match a target tree polygonal model. Talton et al.[128] presented a more general method for achieving high-level design goals (e.g., city sky-line profile for city procedural modeling) using inverse optimization of PM parameters based on a Reversible Jump MCMC formulation so that the resulting model conforms to design constraints. MCMC-based approaches receive control feedback from the completely generated models, hence suffer from significantly higher computational cost at run-time. Alternative approaches incrementally receive control feedback from intermediate states of the PM based on Sequential Monte Carlo, allowing them to reallocate computational resources and converge more quickly [100].

In the above kinds of systems, users prescribe target models, indicator functions, silhouettes or strokes, but their control over the rest of the shape or its details is very limited. In the case of inverse PM parameter optimization (e.g., [128]) producing a result often requires significant amount of computation (i.e., several minutes or hours). In contrast, users of our method have direct control over the output shape and its details based on their input sketch. Our approach trivially requires a single forward pass in a CNN network at run-

time, hence resulting in significantly faster computation for complex procedural models, and producing results at near interactive rates.

Concurrently to our work, Nishida et al. [85] introduced a CNN-based urban procedural model generation from sketches. However, instead of solving directly for the final shape, their method suggests potentially incomplete parts that require further user input to produce the final shape. In contrast, our approach is end-to-end, requiring users to provide only an approximate sketch of the whole shape.

Exploratory systems for procedural models. Exploratory systems provide the user with previously computed and sorted exemplars that help users study the variety of models and select seed models they wish to further explore. Talton et al. [127] organized a set of precomputed samples in a 2D map. The model distribution in the map is established by a set of landmark examples placed by expert users of the system. Lienhard et al. [68] sorted precomputed sample models based on a set of automatically computed views and geometric similarity. They presented the results with rotational and linear thumbnail galleries. Yumer et al. [152] used autoencoder networks to encode the high dimensional parameter spaces into a lower dimensional parameter space that captures a range of geometric features of models.

A problem with these exploratory techniques is that users need to have an exact understanding of the procedural model space to find a good starting point. As a result, it is often hard for users to create new models with these techniques.

Sketch-based shape retrieval. Our work is related to previous methods for retrieving 3D models from a database using sketches as input using various matching strategies to compare the similarity of sketches to database 3D models [34, 42, 95, 29, 110]. However, these systems only allow retrieval of existing 3D models and provide no means to create new 3D models. Our method is able to synthesize new outputs through PM.

Convolutional Neural Networks. Our work is based on recent advances in object recognition with deep CNNs [64]. CNNs are able to learn hierarchical image representa-

tions optimized for image processing performance. CNNs have demonstrated large success in many computer vision tasks, such as object detection, scene recognition, texture recognition and fine-grained classification [64, 27, 37, 99, 24]. Sketch-based 3D model retrieval has also been recently demonstrated through CNNs. Su et al. [124] performed sketch-based shape retrieval by adopting a CNN architecture pre-trained on images, then fine-tuning it on a dataset of sketches collected by human volunteers [28]. Wang et al. [134] used a Siamese CNN architecture to learn a similarity metric to compare human and computer generated line drawings. In contrast to these techniques, we introduce a CNN architecture capable of generating PM parameters, which in turn yield new 2D or 3D shapes.

2.2 Overview

Our algorithm aims to learn a mapping from input approximate, abstract 2D sketches to the PM parameters of a given rule set, which in turn yield 2D or 3D shape suggestions. For example, given a rule set that generates trees, our algorithm produces a set of discrete (categorical) parameters, such as tree family, and continuous parameters, such as trunk width, height, size of leaves and so on. Our algorithm has two main stages: a *training stage*, which is performed offline and involves training a CNN architecture that maps from sketches to PM parameters, and a *runtime synthesis stage*, during which a user provides a sketch, and the CNN predicts PM parameters to synthesize shape suggestions presented back to the user. We outline the key components of these stages below.

CNN architecture. During the training stage, a CNN is trained to capture the highly non-linear relationship between the input sketch and the PM parameter space. A CNN consists of several inter-connected "layers" (Figure 2.1) that process the input sketch hierarchically. Each layer produces a set of feature representations maps, given the maps produced in the previous layer, or in the case of the first layer, the sketch image. The CNN layers are "convolutional", "pooling', or "fully connected layers". A convolutional layer consists of learned filters that are convolved with the input feature maps of the previ-

ous layer (or in the case of the first convolutional layer, the input sketch image itself). A pooling layer performs subsampling on each feature map produced in the previous layer. Subsampling is performed by computing the max value of each feature map over spatial regions, making the feature representations invariant to small sketch perturbations. A fully connected layer consists of non-linear functions that take as input all the local feature representations produced in the previous layer, and non-linearly transforms them to a global sketch feature representation.

In our implementation, we adopt a CNN architecture widely used in computer vision for object recognition, called AlexNet [64]. AlexNet contains five convolutional layers, two pooling layers applied after the first and second convolutional layer, and two fully connected layers. Our CNN architecture is composed of two processing paths (or sub-networks), each following a distinct AlexNet CNN architecture (Figure 2.1). The first sub-network uses the AlexNet set of layers, followed by a regression layer, to produce the continuous parameters of the PM. The second sub-network uses the AlexNet set of layers, followed by a classification layer, to produce probabilities over discrete parameter values of the PM. The motivation behind using these two sub-networks is that the the continuous and discrete PM parameters are predicted more effectively when the CNN feature representations are optimized for classification and regression separately, as opposed to using the same feature representations for both the discrete and continuous PM parameters. Using a CNN versus other alternatives that process the input in one stage (i.e., "shallow" classifiers or regressors, such as SVMs, nearest neighbors, RBF interpolation and so on) also proved to be much more effective in our experiments.

CNN training. As in the case of deep architectures for object recognition in computer vision, training the CNN requires optimizing millions of weights (110 million weights in our case). Training a CNN with fewer number of layers decreases the PM parameter synthesis performance. We leverage available massive image datasets widely used in computer vision, as well as synthetic sketch data that we generated automatically. Specifically, the

convolutional and fully connected layers of both sub-networks are first pre-trained on ImageNet [104] (a publically available image dataset containing 1.2 million photos) to perform generic object recognition. Then each sub-network is further fine-tuned for PM discrete and continuous parameter synthesis based on our synthetic sketch dataset. We note that adapting a network trained for one task (object recognition from images) to another task (PM-based shape synthesis from sketches) can be seen as an instance of transfer learning [150].

Synthetic training sketch generation. To train our CNN, we could generate representative shapes by sampling the parameters of the PM rule set, then ask human volunteers to draw sketches of these shapes. This approach would provide us training data with sketches and corresponding PM parameters. However, such approach would require intensive human labor and would not be scalable. Instead, we followed an automatic approach. We conducted an informal user study to gain insight how people tend to draw shapes generated by PMs. We randomly sampled a few shapes generated by PM rules for containers, jewelry and trees, then asked a number of people to provide freehand drawings of them. Representative sketches and corresponding shapes are shown in Figure 2.2. The sketches are abstract, approximate with noisy contours, as also observed in previous studies on how humans draw sketches [28]. We also found that users tend to draw repetitive patterns only partially. For example, they do not draw all the frame struts in containers, but a subset of them and with varying thickness (i.e., spacing between the outlines of struts). We took into account this observation while generating synthetic sketches. We sampled thousands of shapes for each PM rule set, then for each of them, we generated automatically several different line drawings, each with progressively sub-sampled repeated patterns and varying thickness for their components.

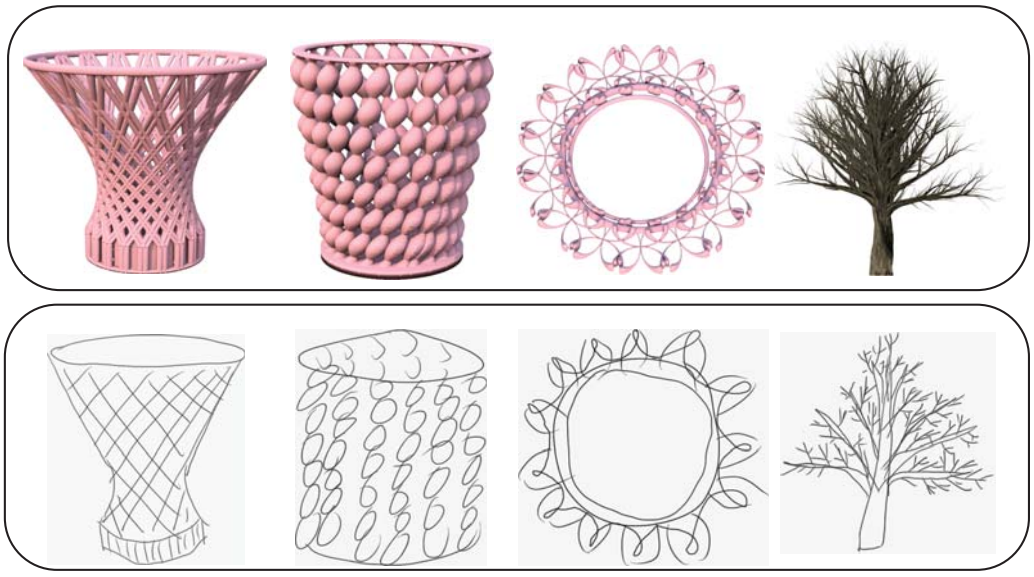

Figure 2.2: Freehand drawings (bottom row) created by users in our user study. The users were shown the corresponding shapes of the top row.

2.3 Method

We now describe the CNN architecture we used for the PM parameter synthesis, then we explain the procedure for training the CNN, and finally our runtime stage.

2.3.1 CNN architecture

Given an input sketch image, our method processes it through a neural network composed of convolutional and pooling layers, fully connected layers, and finally a regression layer to produce PM continuous parameter values. The same image is also processed by a second neural network with the same sequence of layers, yet with different learned filters and weights, and a classification (instead of regression) layer to produce PM discrete parameter values. We now describe the functionality of each type of layer. Implementation details are provided in Appendix B.

Convolutional layers. Each convolutional layer yields a stack of feature maps by applying a set of learned filters that are convolved with the feature representations produced in the previous layer. As discussed in previous work in computer vision [64, 27, 37, 99, 24],

after training, each filter often becomes sensitive to certain patterns observed in the input image, or in other words yield high responses in their presence.

Pooling layers. Each pooling layer subsamples each feature map produced in the previous layer. Subsampling is performed by extracting the maximum value within regions of each input feature map, making the resulting output feature representation invariant to small sketch perturbations. Subsampling also allows subsequent convolutional layers to efficiently capture information originating from larger regions of the input sketch.

Fully Connected Layers. Each fully connected layer is composed of a set of learned functions (known as "neurons" or "nodes" in the context of neural networks) that take as input all the features produced in the previous layer and non-linearly transforms them in order to produce a global sketch representation. The first fully connected layer following a convolutional layer concatenates ("unwraps") the entries of all its feature maps into a single feature vector. Subsequent fully connected layers operate on the feature vector produced in their previous fully connected layer. Each processing function k of a fully connected layer l performs a non-linear transformation of the input feature vector as follows:

$$h_{k,l} = \max(\mathbf{w}_{k,l} \cdot \mathbf{h}_{l-1} + b_{k,l}, 0) \tag{2.1}$$

where $\mathbf{w}_{k,l}$ is a learned weight vector, $\mathbf{h}_{l-1}$ is the feature vector originating from the previous layer, $b_{k,l}$ is a learned bias weight, and $\cdot$ denotes dot product here. Concatenating the outputs from all processing functions of a fully connected layer produces a new feature vector that is used as input to the next layer.

Regression and Classification Layer. The feature vector produced in the last fully connected layer summarizes the captured local or global patterns in the input image. As shown in prior work in computer vision, the final feature vector can be used for image classification, object detection, or texture recognition [64, 27, 37, 99, 24]. In our case, we use this feature vector to predict continuous or discrete PM parameters.

To predict continuous PM parameters, the top sub-network of Figure 2.1 uses a regression layer following the last fully connected layer. The regression layer consists of processing functions, each taking as input the feature vector of the last fully connected layer and non-linearly transforming it to predict each continuous PM parameter. We use a sigmoid function to perform this non-linear transformation, which worked well in our case:

$$O_c = \frac{1}{1 + \exp(-\mathbf{w}_c \cdot \mathbf{h}_L - b_c)} \tag{2.2}$$

where O_c is the predicted value for the PM continuous parameter c, $\mathbf{w}_c$ is a vector of learned weights, $\mathbf{h}_L$ is the feature vector of the last fully connected layer, and b_c is the learned bias for the regression. We note that all our continuous parameters are normalized within the $[0, 1]$ interval.

To predict discrete parameters, the bottom sub-network of Figure 2.1 uses a classification layer after the last fully connected layer. The classification layer similarly consists of processing functions, each taking as input the feature vector of the last fully connected layer and non-linearly transforming it towards a probability for each possible value d of each discrete parameter D_r ($r = 1...R$, where R is the total number of discrete parameters). We use a softmax function to predict these probabilities, as commonly used in multi-class classification methods:

$$Prob(D_r = d) \ = \frac{\exp(\mathbf{w}_{d,r} \cdot \mathbf{h}_L + b_{d,r})}{\exp(\sum_{d'} \mathbf{w}_{d',r} \cdot \mathbf{h}_L + b_{d',r})} \tag{2.3}$$

where d' represent the rest of the discrete values of that parameter, $\mathbf{w}_{d,r}$ is a vector of learned weights, and $b_{d,r}$ is the learned bias for classification.

2.3.2 Training

Given a training dataset of sketches, the goal of our training stage is to estimate the internal parameters (weights) of the convolutional, fully connected, regression and classification layers of our network such that they reliably synthesize PM parameters of a given rule set. There are two sets of trainable weights in our architecture. First, we have the set of weights for the sub-network used for regression $\boldsymbol{\theta}_1$, which includes the regression weights $\{\mathbf{w}_c, b_c\}$ (Equation 2.2) per each PM continuous parameter and the weights used in each convolutional and fully connected layer. Similarly, we have a second set of weights for the sub-network used in classification $\boldsymbol{\theta}_2$, which includes the classification weights $\{\mathbf{w}_{d,r}, b_{d,r}\}$ (Equation 2.3) per each PM discrete parameter value, and the weights of its own convolutional and fully connected layers. We first describe the learning of the weights $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ given a training sketch dataset, then we discuss how such dataset was generated automatically in our case.

CNN learning. Given a training dataset of S synthetic sketches with reference ("ground-truth") PM parameters for each sketch, we estimate the weights $\boldsymbol{\theta}_1$ such that the deviation between the reference and predicted continuous parameters from the CNN is minimized. Similarly, we estimate the weights $\boldsymbol{\theta}_2$ such that the disagreement between the reference and predicted discrete parameter values from the CNN is minimized. In addition, we want our CNN architecture to generalize to sketches not included in the training dataset. To prevent over-fitting our CNN to our training dataset, we "pre-train" the CNN in a massive image dataset for generic object recognition, then we also regularize all the CNN weights such that their resulting values are not arbitrarily large. Arbitrarily large weights in classification and regression problems usually yield poor predictions for data not used in training [12].

The cost function we used to penalize deviation of the reference and predicted continuous parameters as well as arbitrarily large weights is the following:

$$E_r(\boldsymbol{\theta}_1) = \sum_{s=1}^{S}\sum_{c=1}^{C}[\delta_{c,s} == 1]\|O_{c,s}(\boldsymbol{\theta}_1) - \hat{O}_{c,s}\|^2 + \lambda_1||\boldsymbol{\theta}_1||^2 \tag{2.4}$$

where C is the number of the PM continuous parameters, $O_{c,s}$ is the predicted continuous parameter c for the training sketch s based on the CNN, $\hat{O}_{c,s}$ is the corresponding reference parameter value, and $[\delta_{c,s} == 1]$ is a binary indicator function which is equal to 1 when the parameter c is available for the training sketch s, and 0 otherwise. The reason for having this indicator function is that not all continuous parameters are shared across different types (classes) of shapes generated by the PM. The regularization weight λ_1 (also known as weight decay in the context of CNN training) controls the importance of the second term in our cost function, which serves as a regularization term. We set $\lambda_1 = 0.0005$ through grid search within a validation subset of our training dataset.

We use the logistic loss function [12] to penalize predictions of probabilities for discrete parameter values that disagree with the reference values, along with a regularization term as above:

$$E_c(\boldsymbol{\theta}_2) = -\sum_{s=1}^{S}\sum_{r=1}^{R} ln(Prob(D_{s,r} = \hat{d}_{s,r}; \boldsymbol{\theta}_2)) + \lambda_2 ||\boldsymbol{\theta}_2||^2 \tag{2.5}$$

where R is the number of the PM discrete parameters, $Prob(D_{s,r} = \hat{d}_{s,r}; \boldsymbol{\theta}_2)$ is the output probability of the CNN for a discrete parameter r for a training sketch s, and $\hat{d}_{s,r}$ is the reference value for that parameter. We also set $\lambda_2 = 0.0005$ through grid search.

We minimize the above objective functions to estimate the weights $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ through stochastic gradient descent with step rate 0.0001 for $\boldsymbol{\theta}_1$ updates, step rate 0.01 for $\boldsymbol{\theta}_2$, and batch size of 64 training examples. The step rates are set empirically such that we achieve smoother convergence (for regression, we found that the step size should be much smaller to ensure convergence). We also use the dropout technique [121] during training that randomly excludes nodes along with its connections in the CNN with a probability 50% per each gradient descent iteration. Dropout has been found to prevent co-adaptation of the functions used in the CNN (i.e., prevents filters taking same values) and improves generalization [121].

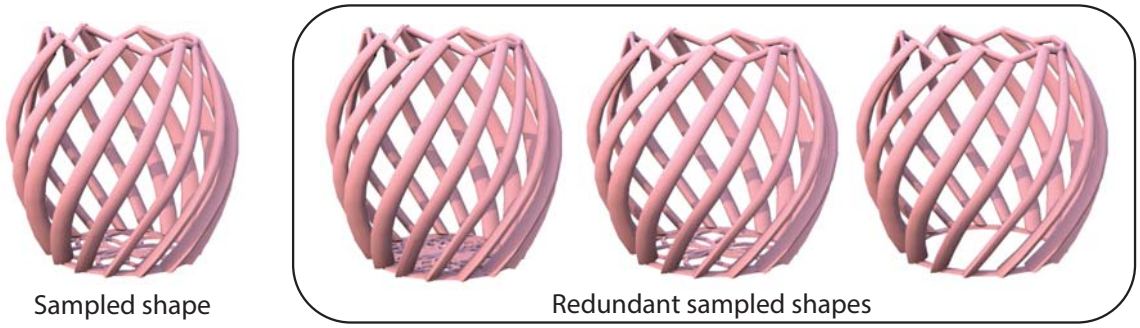

Figure 2.3: To reduce the noise and increase the training speed, our algorithm will remove redundant shapes from our training dataset, here are examples of highly redundant shapes removed from our training dataset.

Pre-training and fine-tuning. To initialize the weight optimization, one option is to start with random values for all weights. However, this strategy seems extremely prone to local undesired minima as well as over-fitting. We instead initialize all the weights of the convolutional and fully connected layers from the AlexNet weights [64] trained in the ImageNet1K dataset [104] (1000 object categories and 1.2M images). Even if the weights in AlexNet were trained for a different task (object classification in images), they already capture patterns (e.g., edges, circles etc) that are useful for recognizing sketches. Starting from the AlexNet weights is an initialization strategy that has also been shown to work effectively in other tasks as well (e.g., 3D shape recognition, sketch classification [150, 124]). We initialize the rest of the weights in our classification and regression layers randomly according to a zero-mean Gaussian distribution with standard deviation 0.01. Subsequently, all weights across all layers of our architecture are trained (i.e., fine-tuned) on our synthetic dataset. Specifically, we first fine-tune the sub-network for classification, then we fine-tune the sub-network for regression using the fine-tuned parameters of the convolutional and fully connected layers of the classification sub-network as a starting point. The difference in PM parameter prediction performance between using a random initialization for all weights versus starting with the AlexNet weights is significant (see experiments in Section 3.1.5).

Synthetic training sketch generation. To train the weights of our architecture, we need a training dataset of sketches, along with reference PM parameters per sketch. We generate such dataset automatically as follows. We start by generating a collection of shapes based on the PM rule set. To generate a collection that is representative enough of the shape variation that can be created through the PM set, we sample the PM continuous parameter space through Poisson disk sampling for each combination of PM discrete parameter values. We note that the number of discrete parameters representing types or styles of shapes in PMs is usually limited (no more than 2 in our rule sets), allowing us to try each such combination. This sampling procedure can still yield shapes that are visually too similar to each other. This is because large parameter changes can still yield almost visually indistinguishable PM outputs. We remove redundant shapes in the collection that do not contribute significantly to the CNN weight learning and unnecessarily increase its training time. To do this, we extract image-based features from rendered views of the shapes using the last fully connected layer of the AlexNet [64], then we remove shapes whose nearest neighbors based on their image features in any view is smaller than a conservative threshold we chose empirically (we calculate the average feature distance between all pairs of nearest neighboring samples, and set the threshold to 3 times of this distance.). Figure 2.3 shows examples of highly redundant shapes removed from our collection.

For each remaining shape in our collection, we generate a set of line drawings. We first generate a 2D line drawing using contours and suggestive contours [26] from a set of views per 3D shape. For shapes that are rotationally symmetric along the upright orientation axis, we only use a single view (we assume that all shapes generated by the PM rule set have a consistent upright orientation). In the case of 2D shapes, we generate line drawings through a Canny edge detector. Then we generate additional line drawings by creating variations of each 2D/3D shape as follows. We detect groups of symmetric components in the shape, then we uniformly remove half of the components per group. Removing more than half of the components degraded the performance of our system, since an increasingly larger

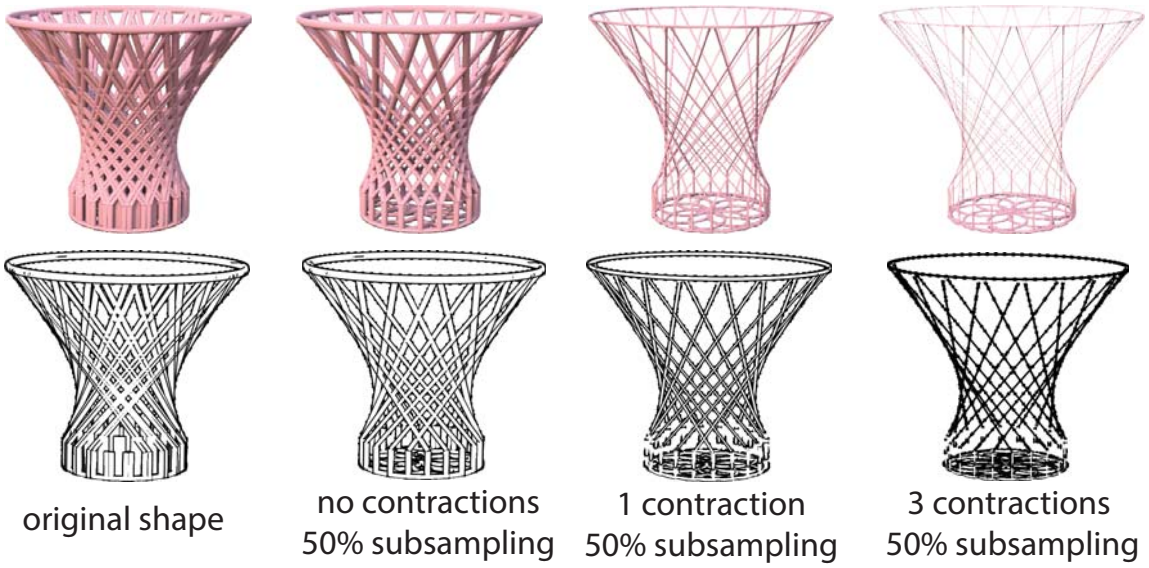

Figure 2.4: Synthetic sketch variations (bottom row) generated for a container training shape. The sketches are created based on symmetric pattern sub-sampling and mesh contractions on the container shape shown on the top.

number of training shapes tended to have too similar, sparse drawings. For the original and decimated shape, we also perform mesh contractions through the skeleton extraction method described in [6] using 1, 2 and 3 constrained Laplacian smoothing iterations. As demonstrated in Figure 2.4, the contractions yield patterns with varying thickness (spacing between contours of the same component).

The above procedure generate shapes with sub-sampled symmetric patterns and varying thickness for its components. The resulting line drawings simulate aspects of human line drawings of PM shapes. As demonstrated in Figure 2.2, humans tend to draw repetitive components only partially and with varying thickness, sometimes using only a skeleton. Figure 2.4 shows the shape variations generated with the above procedure for the leftmost container of Figure 2.2, along with the corresponding line drawings. Statistics for our training dataset per rule set is shown in Table 2.1.

2.3.3 Runtime stage

The trained CNN acts as a mapping between a given sketch and PM parameters. Given a new user input sketch, we estimate the PM continuous parameters and probabilities for

the PM discrete parameters through the CNN. We present the user with a set of shapes generated from the predicted PM continuous parameters, and discrete parameter values ranked by their probability. Our implementation is executed on the GPU. Predicting the PM parameters from a given sketch takes 1 to 2 seconds in all our datasets using a NVidia Tesla K40 GPU. Responses are not real-time in our current implementation based on the above GPU. Yet, users can edit or change their sketch, explore different sketches, and still get visual feedback reasonably fast at near-interactive rates.

2.4 Results

We now discuss the qualitative and quantitative evaluation of our method. We first describe our datasets used in the evaluation, then discuss a user study we conducted in order to evaluate how well our method maps human-drawn freehand sketches to PM outputs.

2.4.1 Datasets

We used three PM rule sets in our experiments: (a) 3D containers, (b) 3D jewelry, (c) 2D trees. All rule sets are built using the Deco framework [78]. The PM rule set for containers generates 3D shapes using vertical rods, generalized cylinders and trapezoids on the side walls, and a fractal geometry at the base. The PM rule set for jewelry generates shapes in two passes: one pass generates the main jewelry shape and the next one decorates the outline of the shape. The PM rule set for trees is currently available in Adobe Photoshop CC 2015. Photoshop includes a procedural engine that generates tree shapes whose parameters are controlled as any other Photoshop's filter.

For each PM rule set, we sample training shapes, then generate multiple training sketches per shape according to the procedure described in Section 2.3.2. The number of training shapes and line drawings in our synthetic sketch dataset per PM rule set is summarized in Table 2.1. We also report the number of continuous parameters, the number of discrete parameters, and total number of different discrete parameter values (i.e., number of classes or PM grammar variations) for each dataset. As shown in the table, all three rule sets

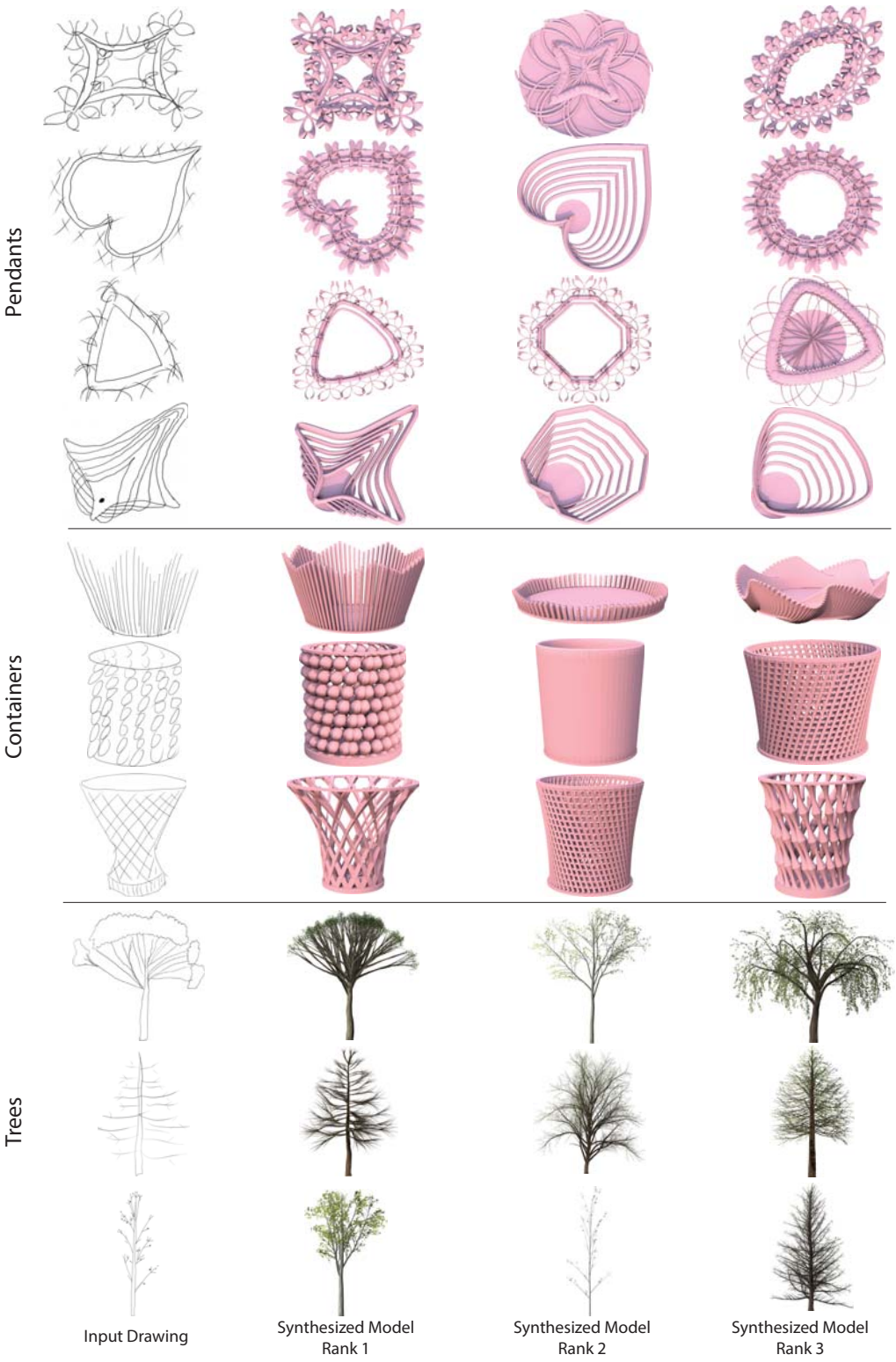

Figure 2.5: Input user line drawings along with the top three ranked output shapes generated by our method.

Statistics	Containers	Trees	Jewelry
# training shapes	$30k$	$60k$	$15k$
# training sketches	$120k$	$240k$	$60k$
# continuous parameters	24	20	15
# discrete parameters	1	1	2
# discrete parameter values/classes	27	34	13
training time (hours)	12	20	9
runtime stage time (sec)	1.5	1.6	1.2

Table 2.1: Dataset statistics

contain several parameters. Tuning these parameters by hand (e.g., with a slider per each parameter) would not be ideal especially for novice users. In the same table, we also report the total time to train our architecture and time to process a sketch during the runtime stage based on our experiments on a TitanX GPU card.

2.4.2 User study

We conducted a user study to evaluate the performance of our method on human line drawings. We presented 15 volunteers a gallery of example shapes generated by each of our PM rule sets (not included in the training datasets), then asked them to provide us with a freehand drawing of a given container, jewelry piece, and tree. None of the volunteers had professional training in arts or 3D modeling. We collected total 45 drawings (15 sketches per dataset). Each drawing had associated PM continuous and discrete parameters based on the used reference shape from the gallery. In this manner, we can evaluate quantitatively how reliably our method or alternative methods are able to synthesize shapes that users intend to convey through their line drawing. We compare the following methods:

Nearest neighbors. The simplest technique to predict PM parameters is nearest neighbors. For each input human drawing, we extract a popular feature representation, retrieve the nearest synthetic sketch in that feature space using Euclidean distances, then generate a shape based on this nearest retrieved sketch PM parameters. We used the Fisher vector representation to represent sketches, which has recently been shown to outperform several other baseline feature representations in sketch-based shape retrieval [109].

SVM classification and RBF interpolation. Instead of nearest neighbors, SVMs can be used for sketch classification based on the same Fisher vector representations, as suggested in [109]. To predict continuous parameters, we used RBF interpolation again on Fisher vector representations. Here we use LIBSVM [21] for SVM classification and Matlab's RBF interpolation.

Standard CNNs. Instead of using Fisher vector representations, an alternative approach is to use a standard CNN trained on an image database as-is, extract a feature representation from the last fully connected layer, then use it in nearest neighbor or SVM classification, and RBF interpolation for regression. We used the features from the last fully connected layer of the AlexNet CNN trained on ImageNet1K.

Single CNN. Instead of using two sub-networks with distinct weights, one can alternatively train a single CNN network with shared filters and weights followed by a regression and a classification layer (i.e., classification and regression rely on the same feature representation extracted by the last fully connected layer).

No pre-training. We tested our proposed architecture of Figure 2.1 without the pre-training procedure. Instead of pre-training, we initialized all the CNN weights based on the random initialization procedure of [64], then trained the whole network from scratch.

Our method. We tested our proposed architecture including the pre-training procedure described in Section 2.3.2.

In Table 2.2, we report the classification accuracy i.e., percentage of times that the reference discrete parameter value (class) agrees with the top ranked discrete value predicted by each examined method for our user study drawings. Our method largely outperforms all alternatives. Yet, since the input sketches often represent significant abstractions and simplifications of shapes, it is often the case that an input drawing cannot be unambiguously mapped to a single output shape. If the shape that the user intended to convey with his approximate line drawing is similar to at least one of the highest ranked shapes returned by a method, we can still consider that the method succeeded. Thus, in Table 2.3 we also

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	20.0%	13.3%	26.7%	20.0%
Nearest neighbors (CNN)	20.0%	20.0%	26.7%	22.2%
SVM (Fisher)	26.7%	26.7%	46.7%	33.3%
SVM (CNN)	26.7%	20.0%	40.0%	28.9%
Single CNN	46.7%	40.0%	60.0%	48.9%
No pretraining	6.7%	6.7%	13.3%	8.9%
Our method	53.3%	53.3%	73.3%	60.0%

Table 2.2: Classification accuracy (top-1) for PM discrete parameters predicted by the examined methods on our user study line drawings.

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	33.3%	26.7%	46.7%	35.6%
Nearest neighbors (CNN)	26.7%	33.3%	40.0%	33.3%
SVM (Fisher)	33.3%	46.7%	60.0%	46.7%
SVM (CNN)	40.0%	33.3%	53.3%	42.2%
Single CNN	66.7%	60.0%	80.0%	68.9%
No pretraining	20.0%	13.3%	20.0%	17.8%
Our method	**80.0%**	**73.3%**	**86.7%**	**80.0%**

Table 2.3: Top-3 classification accuracy for PM discrete parameters predicted by the examined methods on our user study line drawings.

report the top-3 classification accuracy i.e., percentage of times that the reference discrete parameter value (class) is contained within the top three ranked discrete values (classes) predicted by each method. Our method again outperforms all alternatives. On average, it can return the desired class of the shape correctly within the top-3 results around 80% of the time versus 69% using a single CNN (a variant of our method), and 47% using SVM and the Fisher vector representation proposed in a recent state-of-the-art work in sketch-based shape retrieval [109]. In Table 2.4, we report regression error i.e., the relative difference between the predicted and reference values averaged over all continuous parameters for each method. Again, our method outperforms all the alternatives.

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	32.1%	36.3%	29.1%	32.5%
Nearest neighbors (CNN)	29.3%	34.7%	27.5%	30.3%
RBF (Fisher)	30.5%	35.6%	28.9%	37.1%
RBF (CNN)	31.4%	34.2%	27.6%	31.1%
Single CNN	15.2%	17.3%	11.6%	14.7%
No pretraining	35.2%	40.3%	30.5%	35.3%
Our method	**12.7%**	**15.6%**	**8.7%**	**12.3%**

Table 2.4: PM continuous parameter error (regression error) of the examined methods on our user study line drawings.

2.4.3 Evaluation on synthetic sketches

We also evaluate our method on how well it predicts PM parameters when the input is a very precise sketch, such as a computer-synthesized line drawing. Evaluating the performance of our method on synthetic sketches is not useful in practice, since we expect human line drawings as input. Yet, it is still interesting to compare the performance of our algorithm on synthetic sketches versus human line drawings to get an idea of how much important is to have a precise input line drawings as input. To evaluate the performance on synthetic sketches, we performed hold-out validation: we trained our CNN on a randomly picked subset of our original training dataset containing 80% of our shapes, and tested the performance on the remaining subset. The classification accuracy (top-1) was 92.5% for containers, 90.8% for trees, 96.7% for pendants, while the regression error was 1.7%, 2.3%, 1.2% respectively. As expected, performance is improved when input drawings are very precise.

2.4.4 Qualitative evaluation

We demonstrate human line drawings along with the top three ranked outputs generated by the predicted parameters using our method in Figure 2.5. In Figure 1.2, we show human line drawings together with the top ranked shape generated by our method.

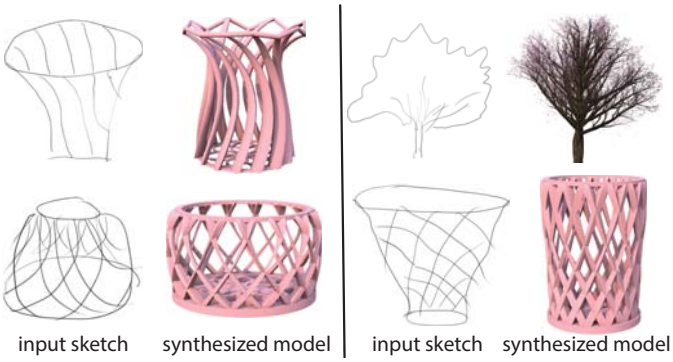

Figure 2.6: When the input sketch is extremely abstract, noisy or does not match well any shape that can be generated by the PM, then our method fails to generate a result that resembles the input drawing.

2.5 Summary and Future Extensions

In this chapter, we introduced a method that helps users to create and explore visual content with the help of procedural modeling and sketches. We demonstrated an approach based on a deep Convolutional Network that maps sketches to procedural model outputs. We showed that our approach is robust and effective in generating detailed, output shapes through a parametric rule set and human line drawings as input. Our method has a number of limitations. First, if the input drawings depict shapes that are different from the shapes that can be generated by the rule set, our method will produce unsatisfactory results (Figure 2.6). The user must have some prior knowledge of what outputs the rule set can produce. When the input drawings become too noisy, contain lots simplifications, exaggerations, or hatching strokes, then again our method will generate outputs that are not likely to be relevant to the users' intentions. We believe that generating synthetic line drawings that reliably simulate such features found in human line drawings would improve the performance of our system. Alternatively, users could be guided to provide more accurate sketches, potentially in 3D, through advanced sketching UIs [9, 155]. Another limitation is that during training we can only support limited number of PM discrete parameters, since we consider each possible combination of discrete values (shape types) to generate our training data (i.e., the number of training shapes grows exponentially with the the number of discrete

parameters). In the current implementation of our method, we used a single viewpoint for training the CNN and ask users to draw from the same viewpoint, which can limit their drawing perspective. Predicting PM parameters from multiple different viewpoints e.g., using view-pooling layers [124] would be a useful extension of our system. Finally, our method does not provide real-time feedback to the user while drawing. An interesting future direction would be to provide such feedback and interactively guide users to draw sketches according to underlying shape variation learned from the procedural models.

CHAPTER 3

LEARNING PARAMETRIC MODELS OF SHAPE FAMILIES

Parametric models of shapes, such as the ones presented in the previous chapter, are not always readily available. Writing procedural rule sets is often cumbersome even for experts, and capturing all shape variability existing in a shape category through hand-engineered parametric models is extremely hard. Thus, in this chapter, I will discuss an approach to learn a parametric model of shape families automatically from raw 3D model collections. The key idea of my approach is to discover geometric and semantic relationships of 3D shapes in the form of point correspondences. During the recent years, due to the growing number of online 3D shape repositories (Trimble Warehouse, Turbosquid and so on), a number of algorithms have been proposed to jointly analyze shapes in large collections to discover such correspondences. The main advantage of these algorithms is that they do not treat shapes in complete isolation from each other, but rather extract useful mappings and correlations between shapes to produce results that are closer to what a human would expect. I will also follow the same trend for building shape correspondences i.e., co-analyze shapes, yet in contrast to prior work, the key ingredient of my approach is to drive shape correspondences through deep-learned local shape descriptors and a probabilistic model for part-aware non-rigid alignment.

Prior work in local shape descriptors has focused on specific scenarios by encoding both local analytical properties like curvature as well as some notion of the larger context of the point within the shape, such as the relative arrangement of other shape regions. Yet the aim in shape analysis is frequently not geometric but functional, or "semantic", similarity of points and local shape regions. Most existing descriptors rely on two main as-

sumptions: (a) one can directly specify which geometric features are relevant for semantic similarity, and (b) strong post-process regularization, such as dense surface alignment, can compensate for all situations when the first assumption fails [144]. The latter is typically computationally expensive, difficult to develop, and task-specific: it benefits greatly from access to better features that reduce the post-processing burden.

To overcome these challenges, we introduce a multi-scale, view-based, projective representation for per-point descriptor learning on 3D shapes[1]. Given a mesh or a point cloud, our method produces a feature descriptor for any point on the shape. We represent the query point by a set of rendered views around it, inspired by the approach of Su et al. [124] for global shape classification. To better capture local and global context, we render views at multiple scales and propose a novel method for viewpoint selection that avoids undesired self-occlusions. This representation naturally lends itself to 2D convolutional neural networks (CNN) operating on the views. The final layer of the base network produces a feature vector for each view, which are then combined across views via a pooling layer to yield a single descriptive vector for the point. The network is trained in Siamese fashion [18] on training pairs of corresponding points.

We then build a probabilistic deformation model based on the learned local shape descriptor to to build accurate geometric correspondences between shapes within a family. Although there has been significant progress on analysis of shapes with similar structure (e.g., human bodies) and similar types of deformations (e.g, isometries or near-isometries), dealing with collections of structurally and geometrically diverse shape families (e.g., furniture, vehicles) still remains an open problem. A promising approach to handle such collections is to iteratively compute part or/and point correspondences and existence using part-based or local non-rigid alignment [59, 44]. The rationale for such approach is that

[1]This work has been accepted to ACM Transactions on Graphics 2017. Source code and data are available on our project page: `http://people.cs.umass.edu/~hbhuang/publications/mvcnn_point/`

part correspondences can help localize point correspondences and improve local alignment, point correspondences can refine segmentations and local alignment, and in turn more accurate local alignment can further refine segmentations and point correspondences. One of the drawbacks of existing methods following this approach is that strict point-to-point correspondences are not that suitable for collections of shapes whose parts exhibit significant geometric variability, such as airplanes, bikes and so on. In addition, these methods use templates made out of basic primitives (e.g., boxes in [59]) or other mediating shapes [44], whose geometry may drastically differ from the surface geometry of several shapes in the collection leading to inaccurate correspondences. In the case of template fitting, approximate statistics on the used primitives can be used to penalize unlikely alignments (e.g., Gaussian distributions on box positions and scales). However, these statistics capture rather limited information about the actual surface variability in the shapes of the input collection. Similarly, in the context of shape synthesis, shape variability is often modeled with statistics over predefined part descriptors that cannot be directly mapped back to surfaces [54] or parameters of simple basic primitives, such as boxes [31, 8].

The dense correspondences established through this deformation model allows us to parameterize shapes in terms of corresponding point position and existences. Based on this shape parameterization, I will build a deep-learned generative model of shapes on top of these point correspondences. To the best of our knowledge, this model is the first to apply deep neural networks on 3D point sets. The key idea is to learn relationships in the surface data hierarchically: our model learns geometric arrangements of points within individual parts through a first layer of latent variables. Then it encodes interactions between the latent variables of the first layer through a second layer of latent variables whose values correspond to relationships of surface arrangements across different parts. Subsequent layers mediate high-level relationships related to the overall shape geometry, semantic attributes and structure. The hierarchical architecture of our model is well-aligned with the compositional nature of shapes: shapes usually have a well-defined structure, their struc-

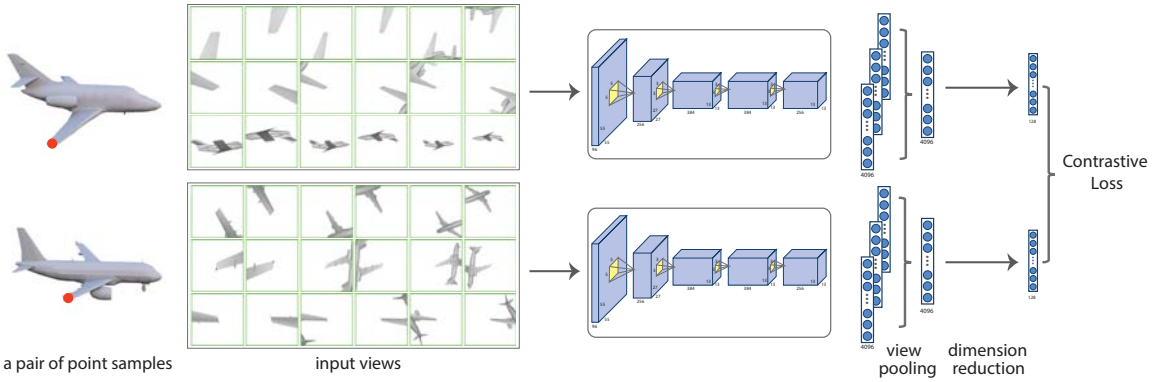

Figure 3.1: Given a pair of surface points (shown in red color) on two shapes, we generate a set of rendered images that capture the local context around each point in multiple scales. Each set of images is processed through an identical stack of convolutional, pooling and non-linear transformation layers resulting in a 4096 dimensional descriptor per point. We aggregate these descriptors across all the input views by using a *view pooling* strategy and further reduce the dimensionality of the descriptor. We train this network with an objective function (contrastive loss) to ensure that geometrically and semantically similar (or dissimilar) point pairs are assigned similar (or dissimilar) descriptors.

ture is defined through parts, parts are made of patches and point arrangements with certain regularities.

In this chapter, I will first describe our pipeline and network for learning local shape descriptors. Then I will present our probabilistic deformation model for learning part-based templates. I will show experiments that demonstrate superior correspondence results than previous state-of-the-art approaches. Finally, I will discuss the deep-learned generative model on 3D point shapes.

3.1 Learning local shape descriptors with view-based convolutional networks

Feature descriptors for surface points are at the heart of a huge variety of 3D shape analysis problems such as keypoint detection, shape correspondence, semantic segmentation, region labeling, and 3D reconstruction [144].

Our goal is to automatically *learn* a point feature descriptor that implicitly captures a notion of semantic correspondence between points, while remaining fully robust to geometric variations, noise, and differences in data sources and representations. We would like to achieve this in a data-driven manner, relying on nothing other than examples of corresponding points on pairs of different shapes. Thus, we do not need to manually guess what geometric features may be relevant for correspondence: we can simply deduce it from examples.

Recent works have explored the possibility of learning descriptors, producing point descriptors robust to natural deformations [75] or adaptable to new data sources [153]. While this corresponds to findings of the image analysis community where features are ubiquitous, learned 3D descriptors have not been as powerful as 2D counterparts because they suffer from (1) limited training data [124], (2) significantly lower spatial resolution resulting in loss of detail sensitivity, or (3) being limited to operate on specific shape classes, such as deformable shapes.

To overcome these challenges, we introduce a multi-scale, view-based, projective representation for per-point descriptor learning on 3D shapes. Given a mesh or a point cloud, our method produces a feature descriptor for any point on the shape. We represent the query point by a set of rendered views around it, inspired by the approach of Su et al. [124] for global shape classification. To better capture local and global context, we render views at multiple scales and propose a novel method for viewpoint selection that avoids undesired self-occlusions. This representation naturally lends itself to 2D convolutional neural networks (CNN) operating on the views. The final layer of the base network produces a feature vector for each view, which are then combined across views via a pooling layer to yield a single descriptive vector for the point. The network is trained in Siamese fashion [18] on training pairs of corresponding points.

The advantages of our approach are two-fold. *First*, the spatial resolution of the projected views is significantly higher than that of voxelized shape representations, which is

crucial to encoding local surface details while factoring in global context. *Second*, 2D rendered views are similar to natural images, allowing us to repurpose neural network architectures that have achieved spectacular success in 2D computer vision tasks. We initialize our framework with filters from a base network for classifying natural images [64], whose weights are pretrained on over a million image exemplars [104]. Such initializations cannot be used by direct 3D methods, for which correspondence training data and established standard architectures are much harder to come by.

To fine-tune the network architecture for descriptor learning, we require access to training pairs of semantically similar points. Here, we make another critical observation. Although correspondence data is available only in limited quantities, large repositories of consistently segmented and labeled shapes have recently become available [148]. We can rely on semantic *segmentation* to guide a part-aware, non-rigid alignment method for semantic *correspondence*, in order to generate very large amounts of dense training data. Our synthetic dataset of $\sim$230M corresponding point pairs is, to our knowledge, the largest such repository, by several orders of magnitude, assembled so far for learning.

We evaluate our method on sparse correspondence and shape segmentation benchmarks and demonstrate that our point descriptors are significantly better than traditional and voxel-based shape descriptors.

3.1.1 Related work

Representing 3D geometric data with global or local descriptors is a longstanding problem in computer graphics and vision with several applications. Global analysis applications, such as recognition and retrieval, require global descriptors that map every *shape* to a point in a descriptor space. Local analysis, such as segmentation and correspondence, needs local descriptors that map every *point on a shape* to a point in descriptor space. In this section we present a brief overview of traditional hand-crafted descriptors and learned representations.

Traditional shape descriptors. Traditionally, researchers and engineers relied on their intuition to hand-craft shape descriptors. Examples of global shape descriptors include 3D shape histograms [3], spherical harmonics [107], shape distributions [88], light-field descriptors [22], 3D Zernike moments [86], and symmetry-based features [57]. Examples of local per-point shape descriptors include spin images [51], shape contexts [11], geodesic distance functions [154], curvature features [35], histograms of surface normals [131], shape diameter [112], PCA-based descriptors [55], heat kernel descriptors [19], and wave kernel signatures [7, 101]. These descriptors capture low-level geometric information, which often cannot be mapped reliably to functional or "semantic" shape properties. In addition, these descriptors often lack robustness to noise, partial data, or large structural shape variations. They frequently rely on a small set of hand-tuned parameters, which are tailored for specific datasets or shape processing scenarios. We refer the reader to the recent survey of Xu et al. [144] for a more detailed discussion on hand-crafted descriptors.

Learned global descriptors. With the recent success of learning based methods, specifically deep neural networks, there has been a growing trend to learn descriptors directly from the data itself instead of manually engineering them. This trend has become apparent in the vision community with the proposal of both global [64] and local [114, 41, 147] deep image descriptors.

With the availability of large 3D shape collections [20], we have seen a similar trend to learn 3D shape descriptors, mostly focusing on learning global descriptors. Early efforts involved shallow metric learning [87] and dictionary learning based on clustering (bags-of-words models) [73, 67] or sparse coding [72]. As a direct extension of the widely successful deep learning techniques in 2D image grids to 3D, voxel-based shape representations have been widely adopted in deep learning for 3D shape recognition [76, 139, 96, 119]. An alternative approach is to first extract a set of hand-crafted geometric features, then further process them through a deep neural network for shape classification and retrieval [142]. Sinha et al. [116] apply deep networks on global shape embeddings in the form of

geometry images. In the context of RGB-D images, neural networks have been proposed to combine features learned from both RGB images and depth data [118, 14, 66]. All the above-mentioned learned global 3D features have shown promising results for 3D object detection, classification, and retrieval tasks. Our focus in this paper, on the other hand, is to learn point-based shape descriptors that can be used for dense feature matching, keypoint detection, affordance labeling and other local shape analysis applications.

Our method is particularly inspired by Su et al.'s multi-view convolutional network for shape classification [124], which has demonstrated high performance in recent shape classification and retrieval benchmarks [108]. Multi-view architectures have also been employed recently for shape segmentation [53]. These architectures are designed to produce a single representation for an entire shape [124], or part label confidences [53]. In contrast, our architecture is designed to produce surface point descriptors. Instead of optimizing for a shape classification or part labeling objective, we employ a siamese architecture that compares geometric similarity of points during training. Instead of fixed views [124], or views selected to maximize surface coverage [53], we use local views adaptively selected to capture multi-scale context around surface points. We also propose an automatic method to create a massive set of point-wise correspondence data to train our architecture based on part-guided, non-rigid registration.

Learned surface descriptors. Recently, there have been some efforts towards learning surface point descriptors. In the context of deformable models, such as human bodies, deep learning architectures have been designed to operate on intrinsic surface representations [75, 16, 17, 83]. The learned intrinsic descriptors produced by these architectures exhibit invariance to isometric or near-isometric deformations. However, in several shape classes, particularly man-made objects, even rigid rotations of parts can change their underlying functionality and semantic correspondences to other parts (e.g. rotating a horizontal tailplane 90 degrees in an airplane would convert it into a vertical stabilizer). Our network attempts to learn the invariance to shape deformations if and when such invariance

exists. We also note that in a recent large-scale benchmark [102], developed concurrently to our work, learning-based extrinsic methods seem to still outperform learning-based intrinsic methods even in the case of deformable shapes. In another concurrent work, Yi et al.[149] synchronize the spectral domains of shapes to learn intrinsic descriptors that are more robust to large structural variations of shapes. To initialize this synchronization, they assume pre-existing extrinsic alignments between shapes. In our case, we do not make any assumptions about consistent shape alignment or orientation.

Wei et al. [136] learn feature descriptors for each pixel in a depth scan of a human for establishing dense correspondences. Their descriptors are tuned to classify pixels into 500 distinct regions or 33 annotated keypoints on human bodies. They are extracted per pixel in a single depth map (and view). In contrast, we produce a single, compact representation of a 3D point by aggregating information across multiple views. Our representation is tuned to compare similarity of 3D points between shapes of different structure or even functionality, going well beyond human body region classification. The method of Guo et al. [39] first extracts a set of hand-crafted geometric descriptors, then utilizes neural networks to map them to part labels. Thus, this method still inherits the main limitations of hand-crafted descriptors. In a concurrent work, Qi et al. [97] presented a classification-based network architecture that directly receives an unordered set of input point positions in 3D and learns shape and part labels. However, this method relies on augmenting local per-point descriptors with global shape descriptors, making it more sensitive to global shape input. Zeng et al. [153] learns a local volumetric patch descriptor for RGB-D data. While they show impressive results for aligning depth data for reconstruction, limited training data and limited resolution of voxel-based surface neighborhoods still remain key challenges in this approach. Instead, our deep network architecture operates on view-based projections of local surface neighborhoods at multiple scales, and adopts image-based processing layers pre-trained on massive image datasets. We refer to the evaluation section (Section 3.1.5) for a direct comparison with this approach.

Also related to our work is the approach by Simo-Serra et al. [114], which learns representations for 64×64 natural image patches through a Siamese architecture, such that patches depicting the same underlying 3D surface point tend to have similar (not necessarily same) representation across different viewpoints. In contrast, our method aims to learn surface descriptors such that geometrically and semantically similar points across different shapes are assigned similar descriptors. Our method learns a single, compact representation for a 3D surface point (instead of an image patch) by explicitly aggregating information from multiple views and at multiple scales through a view-pooling layer in a much deeper network. Surface points can be directly compared through their learned descriptors, while Simo-Serra et al. would require comparing image descriptors for all pairs of views between two 3D points, which would be computationally very expensive.

3.1.2 Overview

The goal of our method is to provide a function f that takes as input any surface point p of a 3D shape and outputs a descriptor $X_p \in \mathbb{R}^D$ for that point, where D is the output descriptor dimensionality. The function is designed such that descriptors of geometrically and semantically similar surface points across shapes with different structure are as close as possible to each other (under the Euclidean metric). Furthermore, the function is designed to be rotationally invariant i.e. we do not restrict our input shapes to have consistent alignment or any particular orientation. Our main assumption is that the input shapes are represented as polygon meshes or point clouds without any restrictions on their connectivity or topology.

We follow a machine learning approach to automatically infer this function from training data. The function can be learned either as a category-specific one (e.g. tuned for matching points on chairs) or as a cross-category one (e.g. tuned to match human region affordances across chairs, bikes, carts and so on). At the heart of our method lies a

convolutional neural network (CNN) that aims to encode this function through multiple, hierarchical processing stages involving convolutions and non-linear transformations.

View-based architecture. The architecture of our CNN is depicted in Figure 3.1 and described in detail in Section 3.1.3. Our network takes as input an unordered set of 2D perspective projections (rendered views) of surface neighborhoods capturing local context around each surface point in multiple scales. At a first glance, such input representation might appear non-ideal due to potential occlusions and lack of desired surface parametrization properties, such as isometry and bijectivity. On the other hand, this view-based representation is closer to human perception (humans perceive projections of 3D shape surfaces), and allows us to directly re-purpose image-based CNN architectures trained on massive image datasets. Since images depict shapes of photographed objects (along with texture), convolutional filters in these image-based architectures already partially encode shape information. Thus, we initialize our architecture using these filters, and further fine-tune them for our task. This initialization strategy has provided superior performance in other 3D shape processing tasks, such as shape classification and retrieval [124]. In addition, to combat surface information loss in the input view-based shape representation, our architecture takes as input multiple, local perspective projections per surface point, carefully chosen so that each point is always visible in the corresponding views. Figure 3.1 shows the images used as input to our network for producing a descriptor on 3D airplane wingtip points.

Learning. Our method automatically learns the network parameters based on a training dataset. To ensure that the function encoded in our network is general enough, a large corpus of automatically generated shape training data is used, as described in Section 3.1.4. Specifically, the parameters are learned such that (i) pairs of semantically similar points are embedded nearby in the descriptor space, and (ii) pairs of semantically dissimilar points are separated by a minimal constant margin in the descriptor space. To achieve this, during the

learning stage, we sample pairs of surface points from our training dataset, and process their view-based representation through two identical, or "Siamese" branches of our network to output their descriptors and measure their distance (see Figure 3.1).

Applications. We demonstrate the effectiveness of the local descriptors learned by our architecture on a variety of geometry processing applications including labeling shape parts, finding human-centric affordances across shapes of different categories, and matching shapes to depth data (see Section 3.1.6).

3.1.3 Architecture

We now describe our pipeline and network architecture (Figure 3.1) for extracting a local descriptor per surface point. In our implementation, for training the network we uniformly sample the input shape surface with 1024 surface points, and compute a descriptor for each of these points. We note that during test time we can sample any arbitrary point on a shape and compute its descriptor.

Pre-processing. In the pre-processing stage, we first uniformly sample viewing directions around the shape parameterized by spherical coordinates (θ, ϕ) (150 directions in our implementation). We render the shape from each viewing direction such that each pixel stores indices to surface points mapped onto that pixel through perspective projection. As a result, for each surface sample point, we can retrieve the viewing directions from which the point is visible. Since neighboring viewing directions yield very similar rendered views of surface neighborhoods, we further prune the set of viewing directions per surface sample point, significantly reducing the number of redundant images fed as input into our network. Pruning is done by executing the K-mediods clustering algorithm on the selected viewing directions (we use their spherical coordinates to measure spherical distances between them). We set $K = 3$ to select representative viewing directions (Figure 3.2). To capture multi-scale contexts for each surface point and its selected viewing directions, we create

$M = 3$ viewpoints placed at distances $0.25, 0.5, 0.75$ of the shape's bounding sphere radius. We experimented with various viewpoint distance configurations. The above configuration yielded the most robust descriptors, as discussed in Section 3.1.5. Increasing the number of viewing directions K or number of distances M did not offer significant improvements.

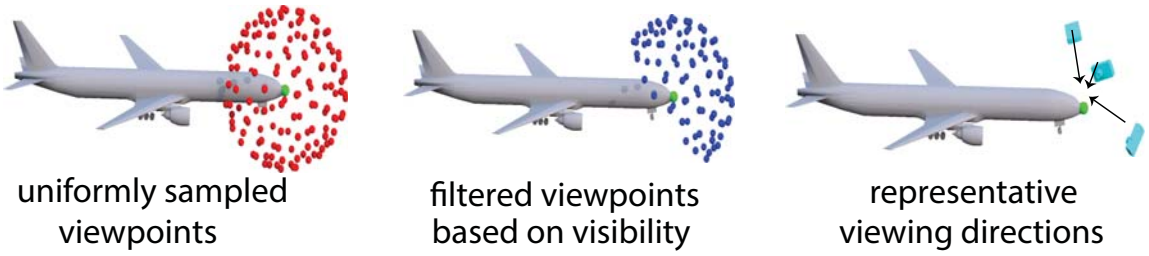

Figure 3.2: For a given point (in green), we show uniformly sampled viewpoints around the shape (in red). We identify the subset of these viewing directions the point is visible from (in blue). Then we perform view clustering to select 3 representative viewing directions.

Input. The input to our network is a set of rendered images depicting local surface neighborhoods around a surface sample point p based on the viewpoint configuration described above. Specifically, we render the shape surface around p from each of the selected viewpoints, using a Phong shader and a single directional light (light direction is set to viewpoint direction). Since rotating the 3D shape would result in rotated input images, to promote rotational invariance, we rotate the input images $L = 4$ times at 90 degree intervals (i.e, 4 in-plane rotations), yielding in total $K \times M \times L = 36$ input images per point. Images are rendered at 227×227 resolution.

View-pooling. Each of the above input images is processed through an identical stack of convolutional, pooling and non-linear transformation layers. Specifically, our CNN follows the architecture known as AlexNet [64]. It includes two convolutional layers, interchanged with two pooling layers and ReLu nonlinearities, followed by three additional convolutional layers with ReLu nonlinearities, a pooling layer and a final fully connected layer.

We exclude the last two fully connected layers of Alexnet, ("fc7"), ("fc8"). The last layer is related to image classification on ImageNet, while using the penultimate layer did not improve the performance of our method as shown in our experiments (see Section 3.1.5).

Passing each rendered image through the above architecture yields a 4096 dimensional descriptor. Since we have 36 rendered views in total per point, we need to aggregate these 36 image-based descriptors into a single, compact point descriptor. The reason is that evaluating the distance of every single image-based descriptor of a point with all other 36 image-based descriptors of another point (1296 pairwise comparisons) would be prohibitively expensive. Note that these 36 views are not ordered in any manner, thus there is no one-to-one correspondence between views and the image-based descriptors of different points (as discussed earlier, shapes are not consistently aligned, points are unordered, thus viewing directions are not consistent across different points).

To produce a single point descriptor, we aggregate the descriptors across the input 36 rendered views, by using an element-wise maximum operation that selects the most discriminative descriptors (largest feature entries) across views. A similar strategy of "max-view pooling" has also been effectively used for shape recognition [124] - in our case, the pooling is applied to local (rather than global) image-based descriptors. Mathematically, given 36 image-based descriptors $Y_{v,p} \in \mathbb{R}^{4096}$ of a point p for views $v = 1 \dots 36$, max view-pooling yields a single descriptor $Y_p \in \mathbb{R}^{4096}$ as follows: $Y_p = \underset{v}{max}(Y_{v,p})$.

We also experimented with taking the average image-based descriptor values across views ("average" view-pooling) as a baseline. However, compared to "max" view-pooling, this strategy led to worse correspondence accuracy, as discussed in Section 3.1.5. An alternative strategy would be to concatenate all the view descriptors, however, this would require an explicit ordering of all views. Ordering the views would require consistent local coordinate frames for all surface points, which is not trivial to achieve.

Dimensionality reduction. Given the point descriptor Y_p produced by view-pooling aggregation, we further reduce its dimensionality to make nearest neighbor queries more effi-

cient and also down-weigh any dimensions that contain no useful information (e.g. shading information). Dimensionality reduction is performed by adding one more layer in our network after view pooling that performs a linear transformation: $X_p = W \cdot Y_p$, where W is a learned matrix of size $K \times 4096$, where K is the dimensionality of the output descriptor. The output dimensionality K was selected by searching over a range of values $K = 16, 32, 64, ..., 512$ and examining performance in a hold-out validation dataset. Based on our experiments, we selected $K = 128$ (see Section 3.1.5).

3.1.4 Learning

Our learning procedure aims to automatically estimate the parameters of the function encoded in our deep architecture. The key idea of our learning procedure is to train the architecture such that it produces similar descriptor values for points that are deemed similar in geometric and semantic sense. To this end, we require training data composed of corresponding pairs of points. One possibility is to define training correspondences by hand, or resort to crowd-sourcing techniques to gather such correspondences. Our function has millions of parameters (40M), thus gathering a large enough dataset with millions of correspondences, would require a significantly large amount of human labor, plus additional human supervision to resolve conflicting correspondences. Existing correspondence benchmarks [60] have limited number of shapes, or focus on specific cases, e.g. deformable shapes [15].

We instead generate training correspondences automatically by leveraging highly structured databases of consistently segmented shapes with labeled parts. The largest such database is the segmented ShapeNetCore dataset [148] that includes 17K man-made shapes distributed in 16 categories. Our main observation is that while these man-made shapes have significant differences in the number and arrangement of their parts, individual parts with the same label are often related by simple deformations [89]. By computing these deformations through non-rigid registration executed on pairs of parts with the same la-

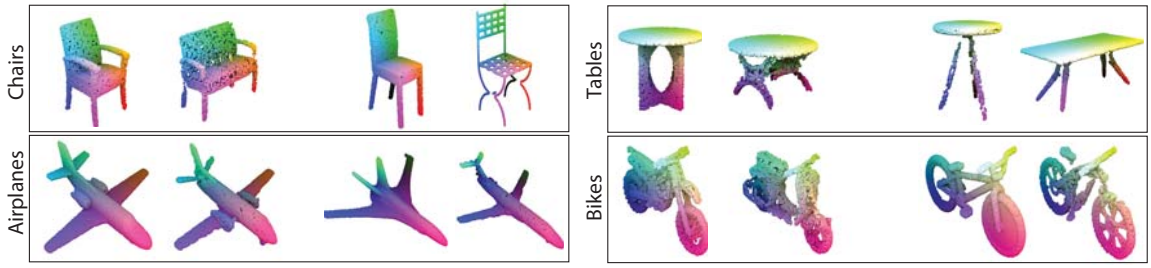

Figure 3.3: Visualization of training point correspondences computed through part-based registration for pairs of shapes from ShapeNetCore. Corresponding points are shown in similar colors.

bel, we can get a large dataset of training point correspondences. Even if the resulting correspondences are potentially not as accurate as carefully human-annotated point correspondences, their massive quantity tends to counterbalance any noise and imperfections. In the next paragraphs, we discuss the generation of our training dataset of correspondences and how these are used to train our architecture.

Part-based registration. Given a pair of consistently segmented shapes A and B, our registration algorithm aims to non-rigidly align all pairs of segments with the same label in the two shapes. First, we sample $10K$ points P_A and P_B from their mesh representation. The points are tagged with the part labels of their corresponding faces. Let P_A^ℓ, P_B^ℓ denote the sets of points originating from a pair of corresponding parts with the same label ℓ in A and B respectively. For each part, we compute an initial affine transformation T_ℓ for P_A^ℓ so that it has the same oriented bounding box as P_B^ℓ. Then for each point $a \in P_A^\ell$, we seek a translation (offset) $o(a)$ that moves it as-close-as possible to the surface represented by P_B^ℓ, while offsets for neighboring points in P_A^ℓ are as similar as possible to ensure a smooth deformation. In the same manner, for each point $b \in P_B^\ell$, we compute an offset $o(b)$ that smoothly deforms the part P_B^ℓ towards P_A^ℓ. To compute the offsets, we minimize a deformation energy that penalizes distances between the point sets of the two parts, and inconsistent offsets between neighboring points:

$$E_{def} = \sum_{a \in P_A^\ell} dist^2(a + o(a), P_B^\ell) + \sum_{b \in P_B^\ell} dist^2(b + o(b), P_A^\ell)$$
$$+ \sum_{a,a' \in N(a)} ||o(a) - o(a')||^2 + \sum_{b,b' \in N(b)} ||o(b) - o(b')||^2 \quad (3.1)$$

where $N(a), N(b)$ are neighborhoods for each point a, b respectively (in our implementation, we use 6 nearest neighbors per point), and $dist$ computes the distance of a translated point to the closest compatible point of the other point set. The energy can be minimized using an ICP-based procedure: given closest pairs of compatible points on the two parts initially, offsets are computed by minimizing the above energy, then closest pairs are updated. The final offsets provide a dense correspondence between closest compatible points of A and B.

Altough alternative deformation procedures could be used (e.g. as-rigid-as possible deformations [120, 125]), we found that our technique provides satisfactory pairs (Figure 3.3), and is fast enough to provide a massive dataset: $100K$ pairs of parts with $10K$ points each were aligned in 12 hours on 3 CPUs with 14 hyperthreaded cores each. Table 3.1 lists statistics about the training dataset we created for 16 ShapeNetCore classes.

Network training. All the parameters $\mathbf{w}$ of our deep architecture are estimated by minimizing a cost function, known as contrastive loss [40] in the literature of metric and deep learning. The cost function penalizes large descriptor differences for pairs of corresponding points, and small descriptor differences for pairs of non-corresponding points. We also include a regularization term in the cost function to prevent the parameter values from becoming arbitrarily large. The cost function is formulated as follows:

$$L(\mathbf{w}) = \sum_{a,b \in C} D^2(X_a, X_b) + \sum_{a,c \notin C} \max(m - D(X_a, X_c), 0)^2 + \lambda ||\mathbf{w}||^2 \quad (3.2)$$

where C is a set of corresponding pairs of points derived from our part-based registration process and D measures the Euclidean distance between a pair of input descriptors. The

ShapeNetCore Category	# shapes used	# aligned shape pairs	# corresponding point pairs
Airplane	500	9699	97.0M
Bag	76	1510	15.1M
Cap	55	1048	10.5M
Car	500	10000	100.0M
Chair	500	9997	100.0M
Earphone	69	1380	13.8M
Guitar	500	9962	99.6M
Knife	392	7821	78.2M
Lamp	500	9930	99.3M
Laptop	445	8880	88.8M
Motorbike	202	4040	40.4M
Mug	184	3680	36.8M
Pistol	275	5500	55.0M
Rocket	66	1320	13.2M
Skateboard	152	3032	30.3M
Table	500	9952	99.5M

Table 3.1: Our training dataset statistics.

regularization parameter (known also as weight decay) λ_1 is set to 0.0005. The quantity m, known as margin, is set to 1 - its absolute value does not affect the learned parameters, but only scales distances such that non-corresponding point pairs tend to have a margin of at least one unit distance.

We initialize the parameters of the convolution layers (i.e., convolution filters) from AlexNet [64] trained on the ImageNet1K dataset (1.2M images) [104]. Since images contain shapes along with texture information, we expect that filters trained on massive image datasets already partially capture shape information. Initializing the network with filters pre-trained on image datasets proved successful in other shape processing tasks, such as shape classification [124].

The cost function is minimized through batch gradient descent. At each iteration, 32 pairs of corresponding points $a, b \in C$ are randomly selected. The pairs originate from random pairs of shapes for which our part-based registration has been executed beforehand.

In addition, 32 pairs of non-corresponding points $a, c \notin C$ are selected, making our total batch size equal to 64. To update the parameters at each iteration, we use the Adam update rule [62], which tends to provide faster convergence compared to other stochastic gradient descent schemes.

Implementation and running time Our method is implemented using the Caffe deep learning library [50]. Our method takes 150 hours to train on the largest categories (mixed 16 categories). Computing a descriptor per point takes 0.57 seconds. Since each shape is sampled with 256 points, it takes 2 minutes to produce local descriptors per shape. Running times are reported on a Tesla GPU card

3.1.5 Evaluation

In this section we evaluate the quality of our learned local descriptors and compare them to state-of-the-art alternatives.

Dataset. We evaluate our descriptors on Kim et al.'s benchmark [60], known as the BHCP benchmark. The benchmark consists of 404 man-made shapes including bikes, helicopters, chairs, and airplanes originating from the Trimble Warehouse. The shapes have significant structural and geometric diversity. Each shape has 6-12 consistently selected feature points with semantic correspondence (e.g. wingtips). Robust methods should provide descriptor values that discriminate these feature points from the rest, and embed corresponding points close in descriptor space.

Another desired descriptor property is rotational invariance. Most shapes in BHCP are consistently upright oriented, which might bias or favor some descriptors. In general, 3D models available on the web, or in private collections, are not expected to always have consistent upright orientation, while existing algorithms to compute such orientation are not perfect even in small datasets (e.g. [33]). Alternatively, one could attempt to consistently align all shapes through a state-of-the-art registration algorithms [45], however, again such

methods often require human expert supervision, or crowd-sourced corrections, for large datasets [20]. To ensure that competing descriptors do not take advantage of any hand-specified orientation or alignment in the data, and to test their rotational invariance, we apply a random 3D rotation to each BHCP shape. We also discuss results for our method when consistent upright-orientation is assumed (Section 3.1.5).

Methods. We test our method against various state-of-the-art techniques, including the learned descriptors produced by the volumetric CNN of 3DMatch [153], and several hand-crafted alternatives: PCA-based descriptors used in [55, 60], Shape Diameter Function (SDF) [112], geodesic shape contexts (SC) [11, 55], and Spin Images (SI) [51]. Although 3DMatch was designed for RGB-D images, the method projects the depth images back to 3D space to get Truncated Distance Function (TDF) values in a volumetric grid, where the volumetric CNN of that method operates on. We used the same type of input for the volumetric CNN in our comparisons, by extracting voxel TDF patches around 3D surface points. To ensure a fair comparison between our approach and 3DMatch, we trained the volumetric CNN of the 3DMatch on the same training datasets as our CNN. We experimented with two training strategies for 3DMatch: (a) training their volumetric CNN from scratch on our datasets, and (b) initializing the volumetric CNN with their publicly available model, then fine-tuning it on our datasets. The fine-tuning strategy worked better than training their CNN from scratch, thus we report results under this strategy.

Training settings. We evaluated our method against alternatives in two training settings. In our first training setting, which we call the "single-category" setting, we train our method on the point-wise correspondence data (described in Section 3.1.4) from a single category and test on shapes of the same or another category. In an attempt to build a more generic descriptor, we also trained our method in a "cross-category" setting, for which we train our method on training data across several categories of the segmented ShapeNetCore dataset, and test on shapes of the same or other categories. We discuss results for the "single-

category" setting in Section 3.1.5, and results for the "cross-category" setting in Section 3.1.5.

Metrics. We use two popular measures to evaluate feature descriptors produced by all methods. First, we use the *Cumulative Match Characteristic (CMC)* measure which is designed to capture the proximity of corresponding points in descriptor space. In particular, given a pair of shapes, and an input feature point on one of the shapes, we retrieve a ranked list of points on the other shape. The list is ranked according to the Euclidean distance between these retrieved points and the input point in descriptor space. By recording the rank for all feature points across all pairs of shapes, we create a plot whose Y-axis is the fraction of ground-truth corresponding points whose rank is equal or below the rank marked on the X-axis. Robust methods should assign top ranks to ground-truth corresponding points.

Another popular measure is correspondence accuracy, also popularized as the Princeton's protocol [60]. This metric is designed to capture the proximity of predicted corresponding points to ground-truth ones in 3D space. Specifically, given a pair of shapes, and an input feature point on one of the shapes, we find the nearest feature point in descriptor space on the other shape, then measure the Euclidean distance between its 3D position and the position of the ground-truth corresponding point. By gathering Euclidean distances across all pairs of shapes, we create a plot whose Y-axis demonstrates the fraction of correspondences predicted correctly below a given Euclidean error threshold shown on the X-axis. Depending on the application, matching symmetric points can be acceptable. Thus, for both metrics, we discuss below results where we accept symmetric (e.g. left-to-right wingtip) matches, or not accepting them.

Results: single-category training.

In this setting, to test our method and 3DMatch on BHCP airplanes, we train both methods on training correspondence data from ShapeNetCore airplanes. Similarly, to test on BHCP chairs, we train both methods on ShapeNetCore chairs. To test on BHCP bikes,

Figure 3.4: Visualization of point correspondence based on our learned descriptors for representative point-sampled BHCP shapes. Corresponding points have the same RGB color.

we train both methods on ShapeNetCore bikes. Since both the BHCP and ShapeNetCore shapes originate from 3D Warehouse, we ensured that the test BHCP shapes were excluded from our training datasets. There is no helicopter class in ShapeNetCore, thus to test on BHCP helicopters, we train both methods on ShapeNetCore airplanes, a related but different class. We believe that this test on helicopters is particularly interesting since it demonstrates the generalization ability of the learning methods to another class. We note that the hand-crafted descriptors are class-agnostic, and do not require any training, thus we simply evaluate them on the BHCP shapes.

Figures 3.5 and Figures 3.6 demonstrate the CMC plots for all the methods on the BHCP dataset for both symmetric and non-symmetric cases. Figure 3.7 and Figure 3.8 show the corresponding plots for the corresponding accuracy measure. According to both the CMC and correspondence accuracy metrics, and in both symmetric and non-symmetric cases, we observe that our learned descriptors outperform the rest, including the learned descriptors of 3DMatch, and the hand-engineered local descriptors commonly used in 3D shape analysis. Based on these results, we believe that our method successfully embeds semantically similar feature points in descriptor space closer than other methods. Figure 3.4 visualizes predicted point correspondences produced by our method for the BHCP test shapes. We observed that our predicted correspondences appear visually plausible, al-

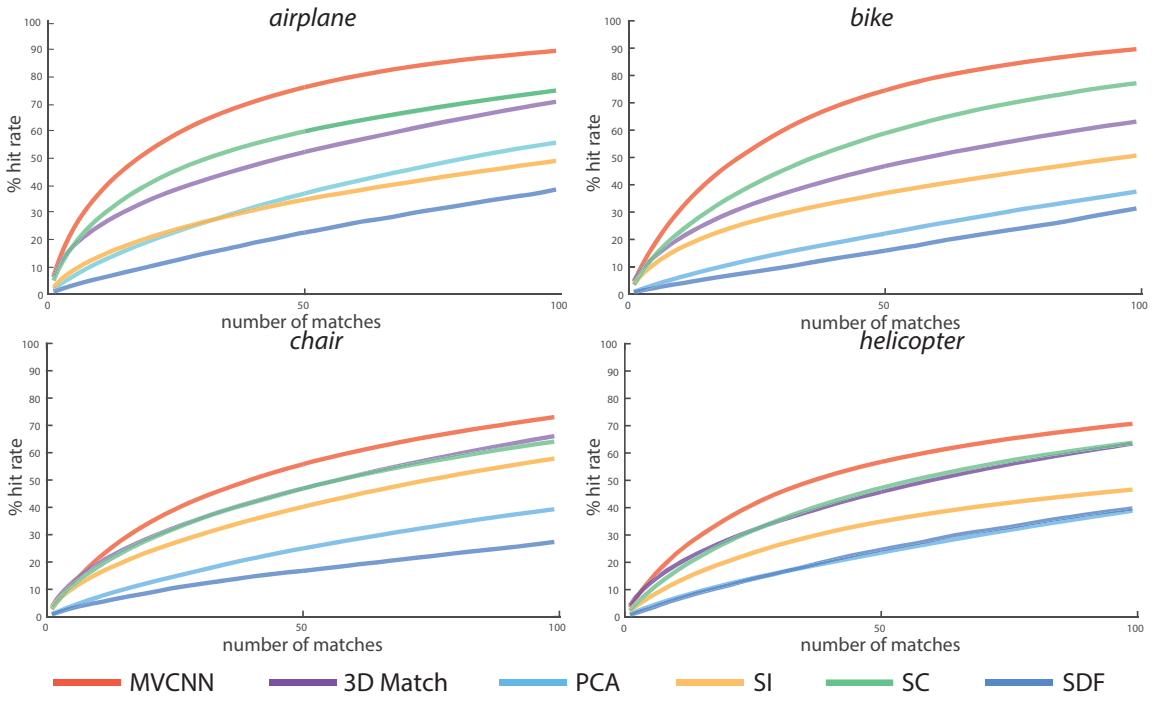

Figure 3.5: CMC plots for each category in the BHCP benchmark for all competing methods (single-category training for learning methods, no symmetric case).

though for bikes we also see some inconsistencies (e.g., at the pedals). We believe that this happens because our automatic non-rigid alignment method tends to produce less accurate training correspondences for the parts of these shapes whose geometry and topology vary significantly.

Results: cross-category training.

In this setting, we train our method on the training correspondence data generated for all 16 categories of the segmented ShapeNetCore dataset (∼977M correspondences), and evaluate on the BHCP shapes (again, we ensured that the test BHCP shapes were excluded from this training dataset.) Figure 3.10 demonstrates the CMC plots and Figure 3.9 demonstrates the correspondence accuracy plots for our method trained across all 16 ShapeNetCore categories ("Mixed 16") against the best performing alternative descriptor (shape contexts) for the symmetric and non-symmetric case averaged over all BHCP classes. As a reference,

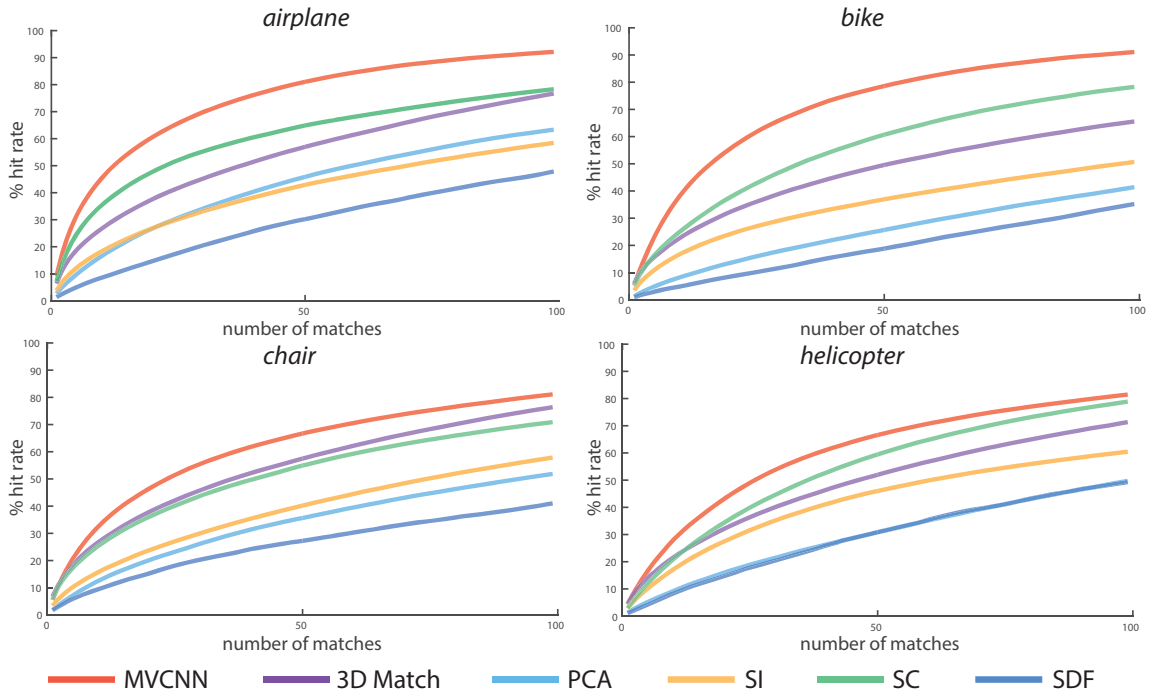

Figure 3.6: CMC plots for each category in the BHCP benchmark for all competing methods (single-category training for learning methods, symmetric case).

we also include the plots for our method trained in the single-category setting ("Single Class"). We observed that the performance of our method slightly drops in the case of cross-category training, yet still significantly outperforms the best alternative method.

We further stretched the evaluation of our method to the case where we train it on 13 categories of the segmented ShapeNetCore ("Mixed 13"), excluding airplanes, bikes, and chairs, i.e. the categories that also exist in BHCP. Even though our method is not explicitly designed to adapt to categories distinct from those used in training (such a capability would belong more to the realm of techniques known in machine learning as "zero-shot learning" [141]), we observe that the performance of our method is still comparable to shape contexts (a bit lower in terms of correspondence accuracy, but a bit higher in terms of CMC). This means that in the worst case where our method is used to extract descriptors on categories not even observed during training, it can still perform favorably compared to hand-crafted descriptors. This indicates that our descriptors are fairly general. We emphasize that the

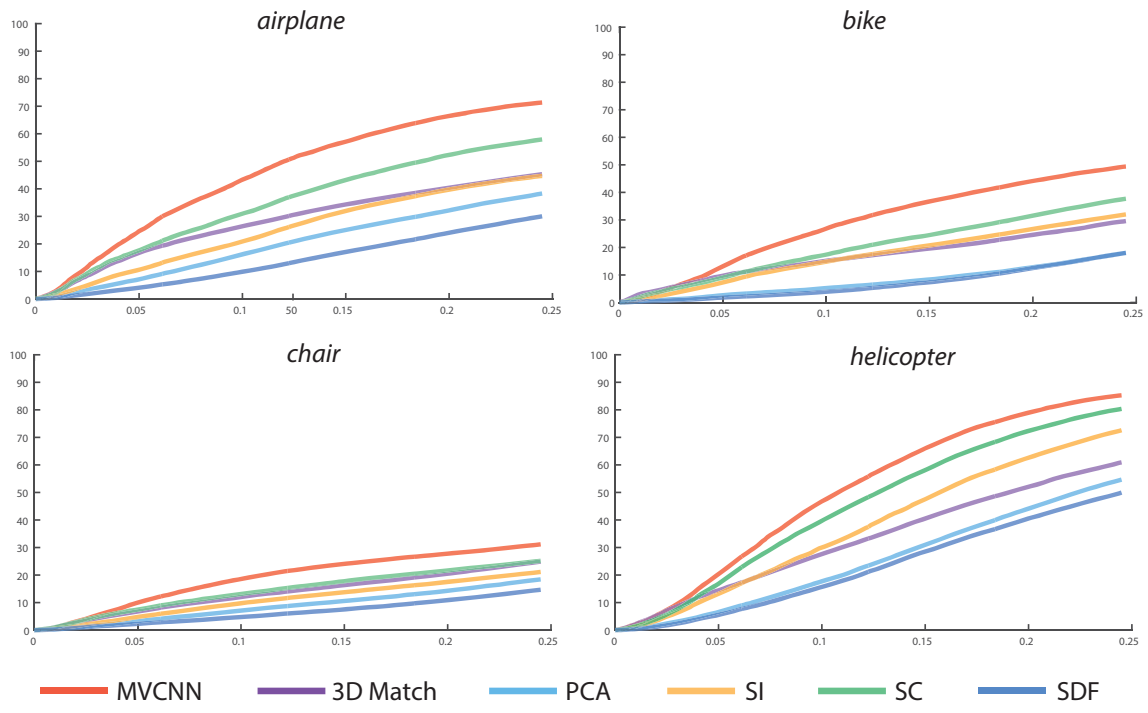

Figure 3.7: Correspondence accuracy for each category in the BHCP benchmark for all competing method (single-category training for learning methods, no, symmetric case).

best performance is achieved when our method is tested on shapes from a category observed in the training data, either alone (single-category setting) or with other categories (cross-category setting). As discussed earlier, the performance of our method is much higher than alternative methods in this case.

Results: alternative algorithmic choices.

In Figure 3.11 and Figure 3.12, we demonstrate correspondence accuracy (symmetric case) under different choices of architectures and viewpoint configurations for our method. Specifically, Figure 3.11(a) shows results with view-pooling applied after the pool5, fc6 and fc7 layer of AlexNet. View-pooling after fc6 yields the best performance. We also demonstrate results with an alternative deeper network, known as VGG16 [115], by applying view-pooling after its fc6 layer. Using VGG16 instead of AlexNet offers marginally better performance than AlexNet, at the expense of slower training and testing. Fig-

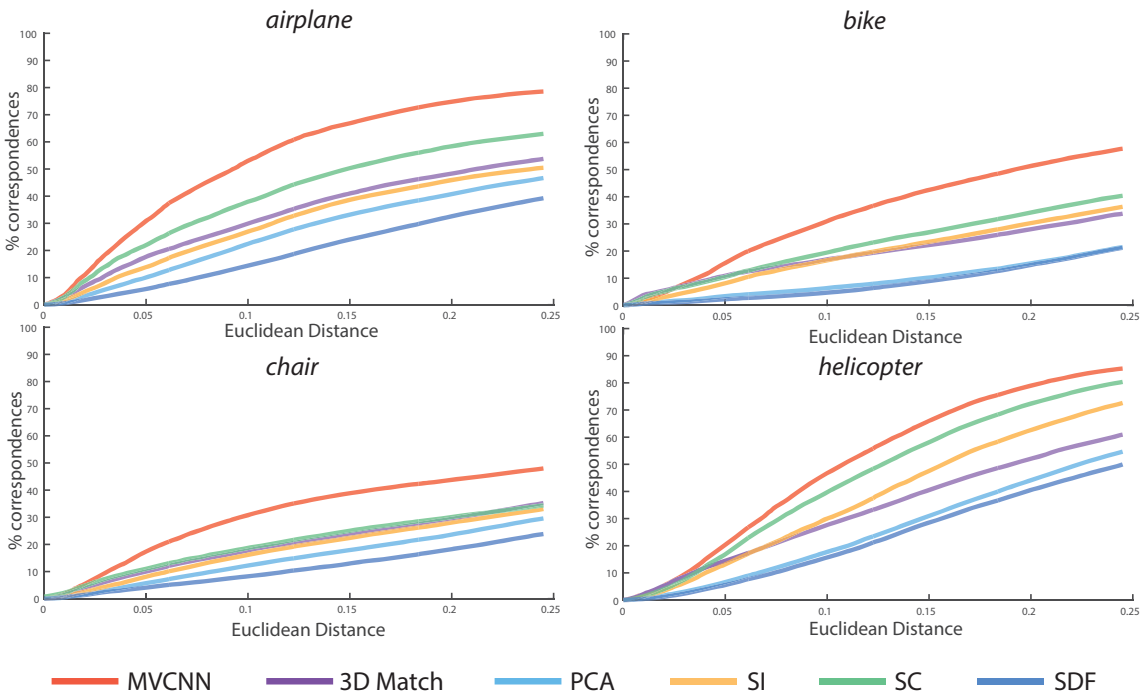

Figure 3.8: Correspondence accuracy for each category in the BHCP benchmark for all competing method (single-category training for learning methods, symmetric case).

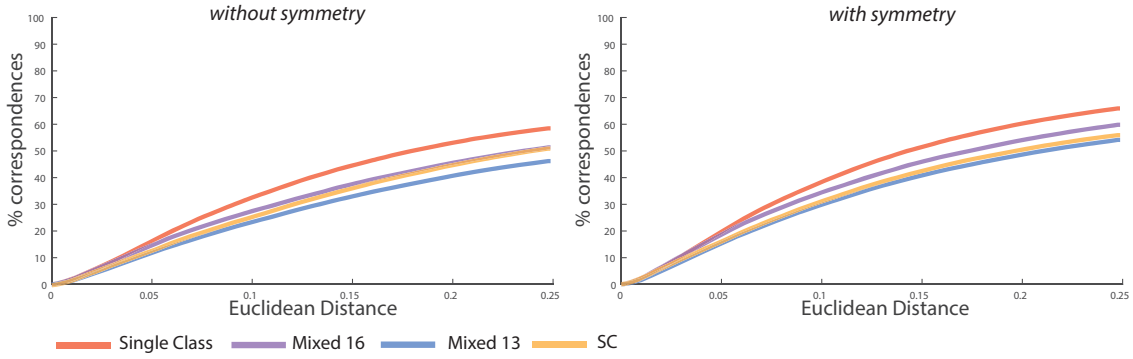

Figure 3.9: Correspondence accuracy for each category in the BHCP benchmark under cross-category training.

ure 3.11(b) shows results with different output dimensionalities for our descriptor, Figure 3.11(c) shows results when we fix the AlexNet layers and update only the weights for our dimensionality reduction layer during training ("no AlexNet fine-tuning"), and when we remove the dimensionality reduction layer and we just perform view-pooling on the raw

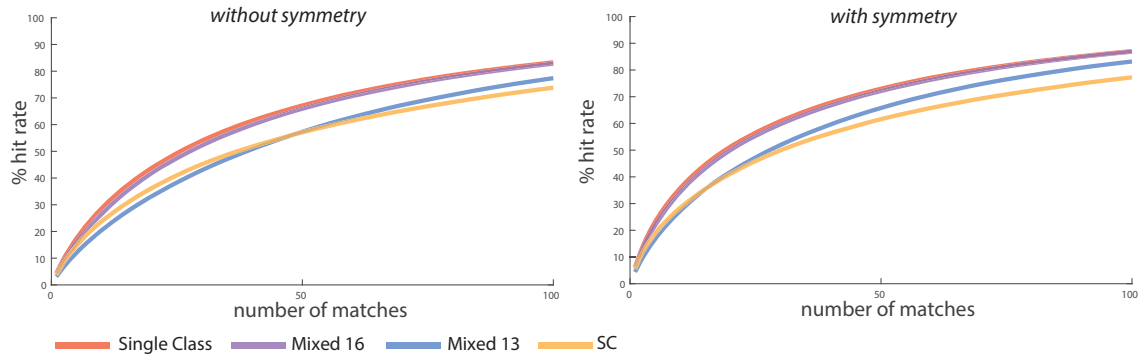

Figure 3.10: Cumulative Match Characteristic for each category in the BHCP benchmark under cross-category training.

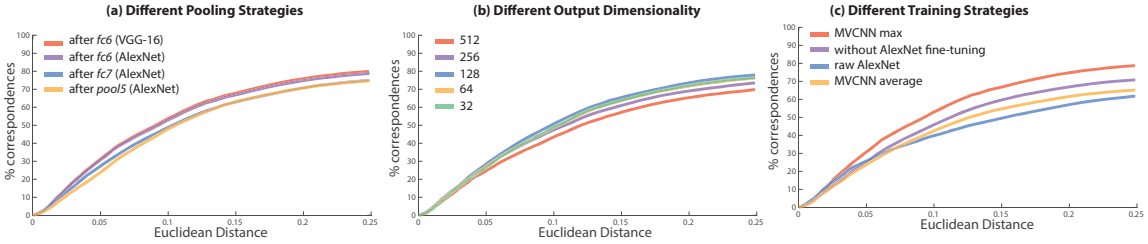

Figure 3.11: Evaluation of alternative algorithmic choices of architectures on the BHCP dataset (airplanes).

4096-D features produced by AlexNet again without fine-tuning ("raw AlexNet"). It is evident that fine-tuning the AlexNet layers and using the dimensionality reduction layer are both crucial to achieve high performance. Figure 3.11(c) also shows performance when "average" view pooling is used in our architecture instead of "max" view pooling. "Max" view pooling offers significantly higher performance than "average" view pooling.

Figure 3.12(a) shows results with different viewpoint distance configurations. Our proposed configuration offers the best performance. Figure 3.12(b) demonstrates results under different numbers of sampled views, and results assuming consistent upright orientation (i.e. we apply random rotation to BHCP shapes only about the upright axis). The plots indicate that the performance of our method is very similar for both the arbitrary orientation and consistent upright orientation cases (slightly higher in the upright orientation case,

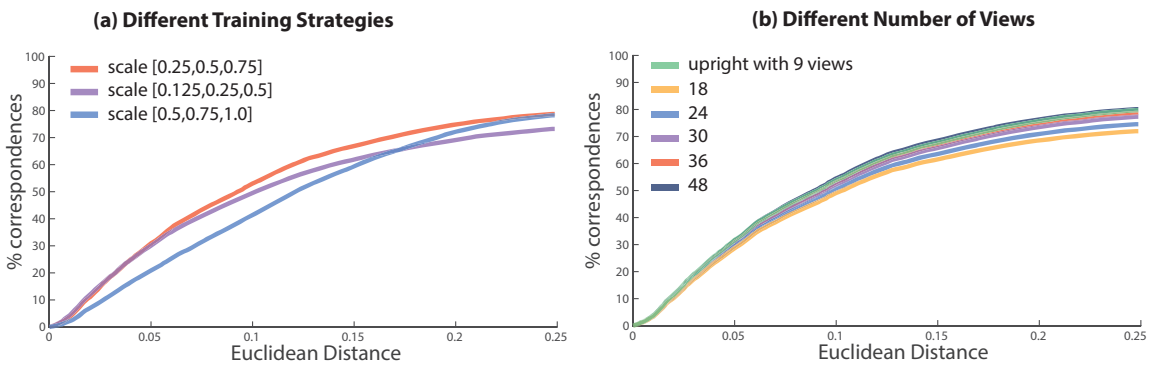

Figure 3.12: Evaluation of alternative algorithmic choices of viewpoint on the BHCP dataset (airplanes).

which makes sense since the test views would tend to be more similar to the training ones in this case). In the presence of fewer sampled views (e.g. 18 or 24), the performance of our method drops slightly, which is an expected behavior since less surface information is captured. The performance of our method is quite stable beyond 30 views. Given a 3D model, nothing technically prevents our method from producing and processing more views per point. The only overhead is the execution time to produce and process these renderings through our architecture. Table 3.2 reports execution times with respect to the used number of views per point. The execution time tends to scale linearly with the number of views. Note that multiple surface points can be processed in parallel.

3.1.6 Applications

In this section, we utilize our learned point descriptors for a wide variety of shape analysis applications, and evaluate with respect to existing methods, and benchmarks. Specifically, we discuss applications of our descriptors to shape segmentation, affordance region detection, and finally matching depth data with 3D models.

view size	18	24	30	36	48
execution time	0.27s	0.44s	0.51s	0.57s	0.91s

Table 3.2: Execution time wrt different numbers of used views per point

Category	JointBoost our descriptors	JointBoost hand-crafted	Guo et al.
Bikes	**77.3%**	72.4%	69.6%
Chairs	**71.8%**	65.7%	60.1%
Helicopters	91.7%	91.1%	**95.6%**
Airplanes	**85.8%**	85.1%	81.7%
Average	**81.7%**	78.6%	76.7%

Table 3.3: Mesh labeling accuracy on BHCP test shapes.

Shape segmentation. We first demonstrate how our descriptors can benefit shape segmentation. Given an input shape, our goal is to use our descriptors to label surface points according to a set of part labels. We follow the graph cuts energy formulation by [55]. The graph cuts energy relies on unary terms that assesses the consistency of mesh faces with part labels, and pairwise terms that provide cues to whether adjacent faces should have the same label. To evaluate the unary term, the original implementation relies on local hand-crafted descriptors computed per mesh face. The descriptors include surface curvature, PCA-based descriptors, local shape diameter, average geodesic distances, distances from medial surfaces, geodesic shape contexts, and spin images. We replaced all these hand-crafted descriptors with descriptors extracted by our method to check whether segmentation results are improved.

Figure 3.13: Examples of shape segmentation results on the BHCP dataset.

Specifically, we trained our method on ShapeNetCore classes as described in the previous section, then extracted descriptors for 256 uniformly sampled surface points for each shape in the corresponding test classes of the BHCP dataset. Then we trained a JointBoost classifier using the same hand-crafted descriptors used in [55] and our descriptors. We also trained the CNN-based classifier proposed in [39]. This method proposes to regroup the above hand-crafted descriptors in a 30x20 image, which is then fed into a CNN-based classifier. Both classifiers were trained on the same training and test split. We used 50% of the BHCP shapes for training, and the other 50% for testing per each class. The classifiers extract per-point probabilities, which are then projected back to nearest mesh faces to form the unary terms used in graph cuts.

We measured labeling accuracy on test meshes for all methods (JointBoost with our learned descriptors and graph cuts, JointBoost with hand-crafted descriptors and graph cuts, CNN-based classifier on hand-crafted descriptors with graph cuts). Table 3.3 summarizes the results. Labeling accuracy is improved on average with our learned descriptors, with significant gains for chairs and bikes in particular.

Matching shapes with 3D scans. Another application of our descriptors is dense matching between scans and 3D models, which can in turn benefit shape and scene understanding techniques. Figure 3.14 demonstrates dense matching of partial, noisy scanned shapes with manually picked 3D database shapes for a few characteristic cases. Corresponding (and symmetric) points are visualized with same color. Here we trained our method on ShapeNetCore classes in the single-category training setting, and extracted descriptors for input scans and shapes picked from the BHCP dataset. Note that we did not fine-tune our network on scans or point clouds. To render point clouds, we use a small ball centered at each point. Even if the scans are noisy, contain outliers, have entire parts missing, or have noisy normals and consequent shading artifacts, we found that our method can still produce robust descriptors to densely match them with complete shapes.

Figure 3.14: Dense matching of partial, noisy scans (even columns) with 3D complete database shapes (odd columns). Corresponding points have consistent colors.

Predicting affordance regions. Finally, we demonstrate how our method can be applied to predict human affordance regions on 3D shapes. Predicting affordance regions is particularly challenging since regions across shapes of different functionality should be matched (e.g. contact areas for hands on a shopping cart, bikes, or armchairs). To train and evaluate our method, we use the affordance benchmark with manually selected contact regions for people interacting with various objects [58] (e.g. contact points for pelvis and palms). Starting from our model trained in the cross-category setting, we fine-tune it based on corresponding regions marked in a training split we selected from the benchmark (we use 50% of its shapes for fine-tuning). The training shapes are scattered across various categories, including bikes, chairs, carts, and gym equipment. Then we evaluate our method by extracting descriptors for the rest of the shapes on the benchmark. Figure 3.15 visualizes corresponding affordance regions for a few shapes for pelvis and palms. Specifically, given marked points for these areas on a reference shape (first column), we retrieve points on other shapes based on their distance to the marked points in our descriptor space. As we can see from these results, our method can also generalize to matching local regions across shapes from different categories with very different global structure.

Figure 3.15: Corresponding affordance regions for pelvis and palms.

3.1.7 Summary and Future Extensions

In this section, we introduced a method that computes local shape descriptors by taking multiple rendered views of shape regions in multiple scales and processing them through a learned deep convolutional network. Through view pooling and dimensionality reduction, we produce compact local descriptors that can be used in a variety of shape analysis applications. Our results confirm the benefits of using such view-based architecture. We also presented a strategy to generate training data to automate the learning procedure. There are a number of avenues of future directions that can address limitations of our method. Currently, we rely on a heuristic-based viewing configuration and rendering procedure. It would be interesting to investigate optimization strategies to automatically select best viewing configurations and rendering styles to maximize performance. We currently rely on perspective projections to capture local surface information. Other local surface parameterization schemes might be able to capture more surface information that could be further processed through a deep network. Our automatic non-rigid alignment method tends to produce less accurate training correspondences for parts of training shapes whose geome-

Figure 3.16: Our learned descriptors are less effective in shape classes for which training correspondences tend to be erroneous.

try and topology vary significantly. Too many erroneous training correspondences will in turn affect the discriminative performance of our descriptors (Figure 3.16). Instead of relying on synthetic training data exclusively, it would be interesting to explore crowdsourcing techniques for gathering human-annotated correspondences in an active learning setting. Rigid or non-rigid alignment methods could benefit from our descriptors, which could in turn improve the quality of the training data used for learning our architecture. This indicates that iterating between training data generation, learning, and non-rigid alignment could further improve performance.

3.2 Probabilistic deformation model

In this section, I will discuss how to build a probabilistic deformation model for estimating shape correspondences based on the learned local shape descriptor and part-aware, non-rigid alignment. I will first present a brief overview of related work on shape correspondences.

3.2.1 Related Work

Our method is related to prior work on data-driven methods for computing shape correspondences in collections with large geometric and structural variability. A complete review of previous research in shape correspondences and segmentation is out of the scope of this paper. We refer the reader to recent surveys in shape correspondences [132], segmentation [130], and structure-aware shape processing [81].

Data-driven shape correspondences. Analyzing shapes jointly in a collection to extract useful geometric, structural and semantic relationships often yields significantly better results than analyzing isolated single shapes or pairs of shapes, especially for classes

of shapes that exhibit large geometric variability. This has been demonstrated in previous data-driven methods for computing point-based and fuzzy correspondences [61, 46, 44, 47]. However, these methods do not leverage the part structure of the input shapes and do not learn a model of surface variability. As a result, these methods often do not generalize well to collections of shapes with significant structural diversity. A number of data-driven methods have been developed to segment shapes and effectively parse their structure [56, 48, 113, 43, 135, 65, 146, 143]. However, these methods build correspondences only at a part level, thus cannot be used to find more fine-grained point or region correspondences within parts. Some of these methods require several training labeled segmentations [56, 132, 143] as input, or require users to interactively specify tens to hundreds of constraints [135].

Our work is closer to that of Kim et al. [59]. Kim et al. proposed a method that estimates point-level correspondences, part segmentations, rigid shape alignments, and a statistical model of shape variability based on template fitting. The templates are made out of boxes that iteratively fit to segmented parts. Boxes are rather coarse shape representations and in general, shape parts frequently have drastically different geometry than boxes. Our method also makes use of templates to estimate correspondences and segmentations, however, their geometry and deformations are learned from scratch. Our produced templates are neither pre-existing parts nor primitives, but new learned parts equipped with probabilities over their point-based deformations. Kim et al.'s statistical model learns shape variability only in terms of individual box parameters (scale and position) and cannot be used for shape synthesis. In contrast, our statistical model encodes both shape structure and actual surface geometry, thus it can be used to generate shapes. Kim et al.'s method computes hard correspondences via closest points, which are less suitable for typical online shape repositories of inanimate objects. Our method instead infers probabilistic, or fuzzy, correspondences and segmentations via a probabilistic deformation model that combines non-rigid surface

alignment, feature-based matching, as well as a deep-learned statistical model of surface geometry and shape structure.

3.2.2 Overview

Given a 3D model collection representative of a shape family, our goal is to compute probabilistic point correspondences and part segmentations of the input shapes (Figure 1.3, left), as well as learn a generative model of 3D shape surfaces (Figure 1.3, right). At the heart of our method lies a probabilistic deformation model that learns part templates and uses them to compute fuzzy point correspondences and segmentations. We now provide an overview of our part template learning concept, our probabilistic deformation model, and our generative surface model.

Learned part templates. Our method computes probabilistic surface correspondences by learning suitable part templates from the input collection. As part template, we denote a learned arrangement of surface points that can be optimally deformed towards corresponding parts of the input shapes under a probabilistic deformation model. To account for structural differences in the shapes of the input collection, the part templates are learned with a hierarchical procedure. Our method first clusters the input collection into groups containing structurally similar shapes, such as benches, four-legged chairs and office chairs (Figure 3.17c). Then a template for each semantic part per cluster is learned (Figure 3.17b). Given the learned group-specific part templates, our method learns higher-level templates for semantic parts that are common across different groups e.g. seats, backs, legs, armrests in chairs (Figure 3.17a). The top-level part templates allow our method to establish correspondences between shapes that belong to structurally different groups, yet share parts under the same label. If parts are unique to a group (e.g., office chair bases), we simply transfer them to the top level and do not establish correspondences to incompatible parts with different label coming from other clusters.

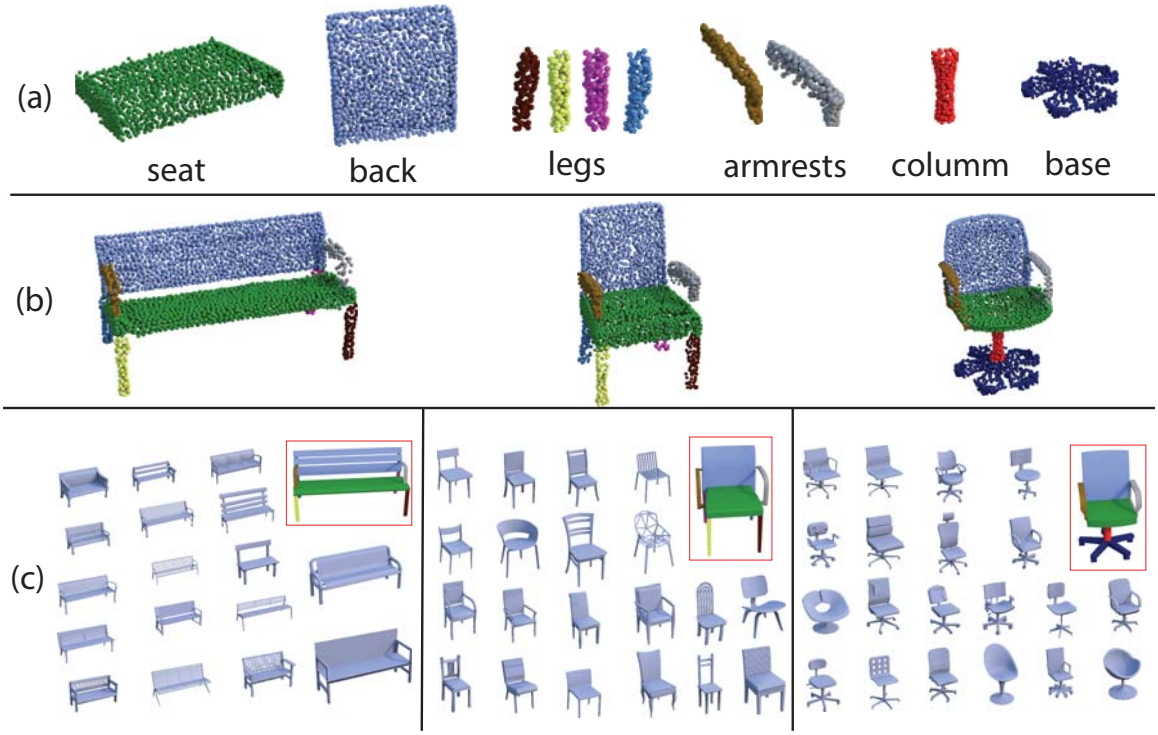

Figure 3.17: Hierarchical part template learning for a collection of chairs. (a) Learned high-level part templates for all chairs. (b) Learned part templates per chair type or group (benches, four-legged chairs, office chairs). (c) Representative shapes per group and exemplar segmented shapes (in red box).

Probabilistic deformation model. At the heart of our algorithm lies a probabilistic deformation model (Section 3.2.3). The model evaluates the probability of deformations applied on the part templates under different corresponding point and part assignments over the input shape surfaces. By performing probabilistic inference on this model, our method iteratively computes the most likely deformation of the part templates to match the input shape parts (Figure 3.18). At each iteration, our method deforms the part templates, and updates probabilities of point and part assignments over the input shape surfaces. The updated probabilistic point and part assignments iteratively guides the deformation and vice versa until convergence.

3.2.3 Probabilistic deformation model

Our method takes as input a collection of shapes, and outputs groups of structurally similar shapes together with learned part templates per group (Figure 3.17). Given the

group-specific part templates, our method also outputs high-level part templates for the whole collection. A probabilistic model is used to infer the part templates. The model evaluates the joint probability of hypothesized part templates, deformations of these templates towards the input shapes, as well as shape correspondences and segmentations. The probabilistic model is defined over the following set of random variables:

Part templates $\mathbf{Y} = \{\mathbf{Y}_k\}$ where $\mathbf{Y}_k \in \mathbb{R}^3$ denotes the 3D position of a point k on a latent part template. There are total K such variables, where K is the number of points on all part templates. The number of points per part template is determined from the provided exemplar shape parts.

Deformations $\mathbf{D} = \{\mathbf{D}_{t,k}\}$ where $\mathbf{D}_{t,k} \in \mathbb{R}^3$ represents the position of a point k on a part template as it deforms towards the shape t. Given T input shapes and K total points across all part templates, there are $K \cdot T$ such variables.

Point correspondences $\mathbf{U} = \{U_{t,p}\}$ where $U_{t,p} \in \{1, 2, ..., K\}$ represents the "fuzzy" correspondence of the surface point p on an input shape t with points on the part templates. In our implementation, each input shape is uniformly sampled with 5000 points, thus there are total $5000 \cdot T$ such random variables.

Surface segmentation $\mathbf{S} = \{S_{t,p}\}$ where $S_{t,p} \in \{1, ..., L\}$ represents the part label for a surface point p on an input shape t. L is the number of available part templates, corresponding to the total number of semantic part labels. There are also $5000 \cdot T$ surface segmentation variables.

Input surface points $\mathbf{X}_t = \{\mathbf{X}_{t,p}\}$ where $X_{t,p} \in \mathbb{R}^3$ represents the 3D position of a surface sample point p on an input shape t.

Our deformation model is defined through a set of factors, each representing the degree of compatibility of different assignments to the random variables it involves. The factors control the deformation of the part templates, the smoothness of these deformations, the fuzzy point correspondences between each part template and input shape, and the shape

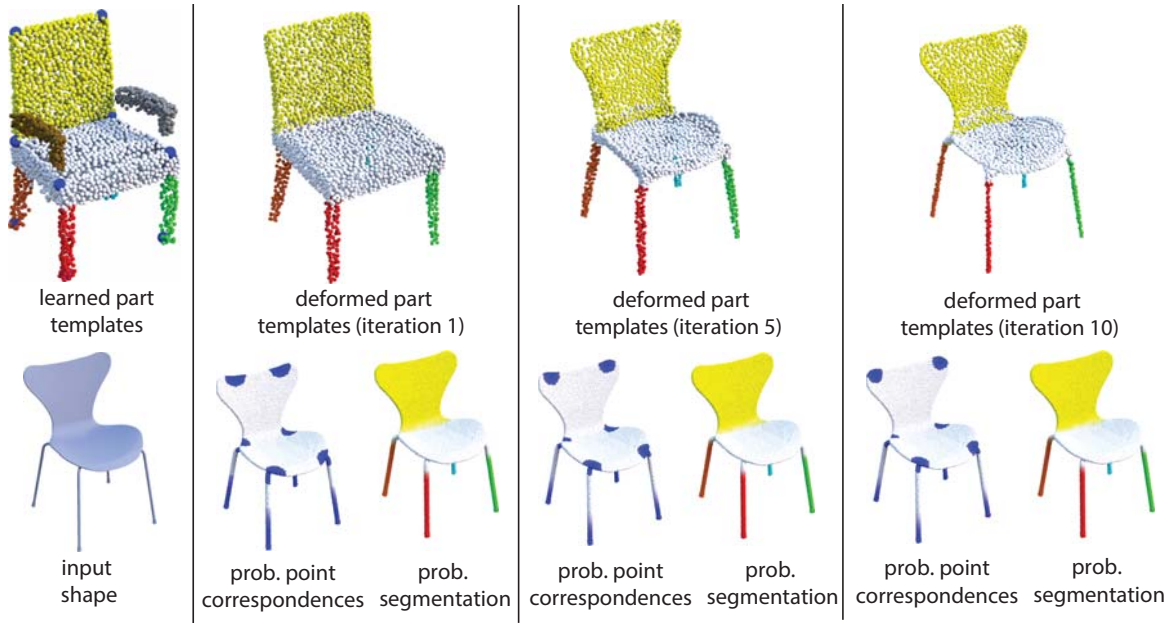

Figure 3.18: Given learned part templates for four-legged chairs, our method iteratively deforms them towards an input shape through probabilistic inference. At each iteration, probability distributions over deformations, point correspondences and segmentations are inferred according to our probabilistic model (probabilistic correspondences are shown only for the points appearing as blue spheres on the learned part templates).

segmentations. The factors are designed out of intuition and experimentation. We now explain the factors used in our model in detail.

Unary deformation factor. We first define a factor that assesses the consistency of deformations of individual surface points on the part templates with an input shape. Given an input shape t represented by its sampled surface points $\mathbf{X}_t$, the factor is defined as follows:

$$\phi_1(\mathbf{D}_{t,k}, \mathbf{X}_{t,p}, U_{t,p} = k) = \\ \exp\left\{ -.5(\mathbf{D}_{t,k} - \mathbf{X}_{t,p})^T \Sigma_1^{-1} (\mathbf{D}_{t,k} - \mathbf{X}_{t,p}) \right\}$$

where the parameter Σ_1 is a diagonal covariance matrix estimated automatically, as we explain below. The factor encourages deformations of points on the part templates towards the input shape points that are closest to them.

Deformation smoothness factor. This factor encourages smoothness in the deformations of the part templates. Given a pair of neighboring surface points k, k' on an input template, the factor is defined as follows:

$$\phi_2(\mathbf{D}_{t,k}, \mathbf{D}_{t,k'}, \mathbf{Y}_k, \mathbf{Y}_{k'}) = \exp\left\{ -.5((\mathbf{D}_{t,k} - \mathbf{D}_{t,k'}) - (\mathbf{Y}_k - \mathbf{Y}_{k'}))^T \Sigma_2^{-1} ((\mathbf{D}_{t,k} - \mathbf{D}_{t,k'}) - (\mathbf{Y}_k - \mathbf{Y}_{k'})) \right\}$$

The factor favors locations of deformed points relative to their deformed neighbors that are closer to the ones on the (undeformed) part templates. The covariance matrix Σ_2 is diagonal and is also estimated automatically. In our implementation, we use the 20 nearest neighbors of each point k to define its neighborhood.

Correspondence factor. This factor evaluates the compatibility of a point on a part template with an input surface point by comparing their local shape descriptors:

$$\phi_3(U_{t,p} = k, \mathbf{X}_t) = \exp\left\{ -.5(\mathbf{f}_k - \mathbf{f}_{t,p})^T \Sigma_3^{-1} (\mathbf{f}_k - \mathbf{f}_{t,p}) \right\}$$

where $\mathbf{f}_k$ and $\mathbf{f}_{t,p}$ are local shape descriptors as we discussed, in Section 3.1 evaluated on the points of the part template and the input surface $\mathbf{X}_t$ respectively, Σ_3 is a diagonal covariance matrix.

Segmentation factor. This factor assesses the consistency of each part label with an individual surface point on an input shape. The part label depends on the fuzzy correspondences of the point with each part template:

$$\phi_4(S_{t,p} = l, U_{t,p} = k) = \begin{cases} 1, & \text{if } label(k) = l \\ \epsilon, & \text{if } label(k) \neq l \end{cases}$$

where $label(k)$ represents the label of the part template with the point k. The constant ϵ is used to avoid numerical instabilities during inference, and is set to 10^{-3} in our implementation.

Segmentation smoothness. This factor assesses the consistency of a pair of neighboring surface points on an input shape with part labels:

$$\phi_5(S_{t,p} = l, S_{t,p'} = l', \mathbf{X}_t) = \begin{cases} 1 - \Phi_{t,p,p'}, & \text{if } l \neq l' \\ \Phi_{t,p,p'}, & \text{if } l = l' \end{cases}$$

where p' is a neighboring surface point to p and:

$$\Phi_{t,p,p'} = \exp\left\{ -.5(\mathbf{f}_{t,p} - \mathbf{f}_{t,p'})^T \Sigma_5^{-1} (\mathbf{f}_{t,p} - \mathbf{f}_{t,p'}) \right\}$$

To define the neighborhood for each point p, we first segment the input shape into convex patches based on the approximate convex segmentation algorithm [5]. Then we find the 20 nearest neighbors from the patch the point p belongs to. The use of information from convex patches helped our method compute smoother boundaries between different parts of the input shapes.

Deformation model. Our model is defined as a Conditional Random Field (CRF) [63] multiplying all the above factors together and normalizing the result to express a joint probability distribution over the above random variables.

$$\begin{aligned} P_{crf}(\mathbf{Y}, \mathbf{U}, \mathbf{S}, \mathbf{D} | \mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_t \Bigg[& \prod_{k,p} \phi_1(\mathbf{D}_{t,k}, \mathbf{X}_{t,p}, U_{t,p}) \\ & \cdot \prod_{k,k'} \phi_2(\mathbf{D}_{t,k}, \mathbf{D}_{t,k'}, \mathbf{Y}_k, \mathbf{Y}_{k'}) \cdot \prod_p \phi_3(U_{t,p}, \mathbf{X}_t) \\ & \cdot \prod_p \phi_4(S_{t,p}, U_{t,p}) \cdot \prod_{p,p'} \phi_5(S_{t,p}, S_{t,p'}, \mathbf{X}_t) \Bigg] \end{aligned} \tag{3.3}$$

Mean-field inference. Using the above model, our method infers probability distributions for part templates and deformations, as well as shape segmentations and correspondences. To perform inference, we rely on the mean-field approximation theory due to its efficiency and guaranteed convergence properties. We approximate the original probability distribution P_{crf} with another simpler distribution Q such that the KL-divergence between these two distributions is minimized:

$$P_{crf}(\mathbf{Y}, \mathbf{U}, \mathbf{S}, \mathbf{D}|\mathbf{X}) \approx Q(\mathbf{Y}, \mathbf{U}, \mathbf{S}, \mathbf{D}|\mathbf{X})$$

where the approximating distribution Q is a product of individual distributions associated with each variable:

$$Q(\mathbf{Y}, \mathbf{U}, \mathbf{S}, \mathbf{D}|\mathbf{X}) = \prod_{k} Q(\mathbf{Y}_k) \prod_{t,k} Q(\mathbf{D}_{t,k}) \prod_{t,p} Q(U_{t,p}) \prod_{t,p} Q(S_{t,p})$$

For continuous variables, we use Gaussians as approximating individual distributions, while for discrete variables, we use categorical distributions. We provide all the mean-field update derivations for each of the variables in Appendix C. Learning the part templates entails computing the expectations of part template variables $\mathbf{Y}_k$ with respect to their approximating distribution $Q(\mathbf{Y}_k)$. For each part template point $\mathbf{Y}_k$, the mean-field update is given by:

$$Q(\mathbf{Y}_k) \propto \exp\left\{ -.5(\mathbf{Y}_k - \boldsymbol{\mu}_k)^T \Sigma_2^{-1} (\mathbf{Y}_k - \boldsymbol{\mu}_k) \right\}$$

where:

$$\boldsymbol{\mu}_k = \frac{1}{|\mathcal{N}(k)|} \sum_{k'} \left(\mathbb{E}_Q[\mathbf{Y}_{k'}] + \frac{1}{T} \sum_{t} (\mathbb{E}_Q[\mathbf{D}_{t,k}] - \mathbb{E}_Q[\mathbf{D}_{t,k'}]) \right)$$

and $\mathcal{N}(k)$ includes all neighboring points k' of point k on the part template. As seen in the above equation, to compute the mean-field updates, we need to compute expectations over deformations of part templates. However, to compute these expectations, we require an initialization for the part templates, as described next.

Clustering. The first step of our method is to cluster the input shapes into groups of structurally similar shapes. For this purpose, we define a dissimilarity measure between two shapes based on our unary deformation factor. We measure the amount of deformation required to map the points of one shape towards the corresponding points of the other shape in terms of their Euclidean distance, and vice versa. For small datasets (with less than 100 shapes), we compute the dissimilarities between all-pairs of shapes, then use the affinity propagation clustering algorithm [32]. The affinity propagation algorithm takes as input dissimilarities between all pairs of shapes, and outputs a set of clusters together with a set of representative, or exemplar, shape per cluster. We note that another possibility would be to use all the factors of the model to define a dissimilarity measure, however, this proved to be computationally too expensive. For larger datasets, we compute a graph over the input shapes, where each node represents a shape, and edges connect shapes which are similar according to a shape descriptor [61]. We compute distances for pairs of shapes connected with an edge, then embed the shapes with the Isomap technique [129] in a 20-dimensional space. We use the distances in the embedded space as dissimilarity metric for affinity propagation.

As mentioned above, affinity propagation also identifies a representative, or exemplar, shape per cluster. In the case of manual initialization of our method, we ask the user to segment each identified exemplar shape per group, or let him select a different exemplar shape if desired. In the case of non-manual segmentation initialization, we rely on a co-segmentation technique to get an initial segmentation of each exemplar shape. In our implementation we use the co-segmentation results provided by Kim et al. [59]. Even if the initial segmentation of the exemplar shapes is approximate, our method updates and

improves the segmentations for all shapes in the collection based on our probabilistic deformation model, as demonstrated in the results. To ensure that the identified exemplar shape has all (or most) representative parts per cluster in the case of automatic initialization, we modify the clustering algorithm to force it to select an exemplar from the shapes with the largest number of parts per cluster based on the initial shape co-segmentations. Figure 3.17 shows the detected clusters for a small chair dataset and user-specified shape segmentations for each exemplar shape per cluster.

Inference procedure and parameter learning. Given the clusters and initially provided parts for exemplar shapes, the mean-field procedure follows the Algorithm 1. At line 1, we initialize the approximating distributions for the part templates according to a Gaussian centered at the position of the surface points on the provided exemplar parts. We then initialize the deformed versions of the part templates using the provided exemplar parts after aligning them with each exemplar shape (lines 2-5). Alignment is done by finding the least-squares affine transformation that maps the exemplar shapes with the shapes of their group. The affine transformation is used to account for anisotropic scale differences between shapes in each group. We initialize the approximating distributions for correspondences and segmentations with uniform distributions (lines 6-9). Then we start an outer loop (line 10) during which we update the covariance matrices (line 11) and execute an inner loop for updating the approximating distributions for segmentations, correspondences and deformations for each input shape (lines 12-24). The covariance matrices are computed through piecewise training [126] on each factor separately using maximum likelihood. We provide the parameter updates in Appendix C. Finally, we update the distributions on part templates (line 25). The outer loop for updating the part templates and parameters requires $5 - 10$ iterations to reach convergence in our datasets. Convergence is checked based on how much the inferred point position on the part template deviate on average from the ones of the previous iteration. For the inner loop, we practically found that running more mean-field iterations (10 in our experiments) for updating the deformations helps the algorithm

converge to better segmentations and correspondences. During the inference procedure, our method can infer negligible probability (below 10^{-3}) for one or more part labels for all points on an input shape. This happens when an input shape has parts that are subset of the ones existing in its group. In this case, the part templates missing from that shape are deactivated e.g., Figure 3.18 demonstrates this case where the input shape does not have armrests.

High-level part template learning. Learning the part templates at the top level of hierarchy follows the same algorithm as above with different input. Instead of the shapes in the collection, the algorithm here takes as input the learned part templates per group. For initialization, we try each part from the lower level to initialize each higher-level part template per label, and select the one with highest probability according to our model (Equation 3.3). The part templates, deformations and correspondences are updated according to Algorithm 1. For this step, we omit the updates for segmentations, since the algorithm works with individual parts. We note that it is straightforward to extend our method to handle more hierarchy levels of part templates (e.g., splitting the clusters into sub-clusters also leading to the use of more exemplars per shape type, or group) by applying the same algorithm hierarchically and using a hierarchical version of affinity propagation [38]. Experimentally, we did not see any significant benefit from using multiple exemplars per group at least in the datasets we used.

Implementation and running times. Our method is implemented in C++ and is CPU-based. Learning templates takes 6 hours for our largest dataset (3K chairs) with a E5-2697 v2 processor. Given a new shape at test time, we can estimate correspondences and segmentation based on the learned templates in 30 seconds (same CPU).

3.2.4 Evaluation

We now describe the experimental validation of our method for computing semantic point correspondences and shape segmentation.

input : Input collection and initially segmented parts of exemplar shapes
output: Learned part templates, shape correspondences and segmentations

Initialize $\mathbb{E}_Q[\mathbf{Y}_k]$ from the position of the exemplar shape part points;
for each shape $t \leftarrow 1$ **to** T **do**
    for each part template point $k \leftarrow 1$ **to** K **do**
        Initialize $\mathbb{E}_Q[\mathbf{D}_{t,k}]$ from the aligned part templates with the shape t;
    end
    for each surface point $p \leftarrow 1$ **to** P **do**
        Initialize $Q(U_{t,p})$ and $Q(S_{t,p})$ to uniform distributions;
    end
end
repeat
    Update covariance matrices $\mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathbf{\Sigma}_3, \mathbf{\Sigma}_5$;
    repeat
        for each shape $t \leftarrow 1$ **to** T **do**
            for each surface point $p \leftarrow 1$ **to** P **do**
                update correspondences $Q(\mathbf{U}_{t,p})$;
                update segmentations $Q(\mathbf{S}_{t,p})$;
            end
            for *iteration* $\leftarrow 1$ **to** 10 **do**
                for each part template point $k \leftarrow 1$ **to** K **do**
                    update deformations $Q(\mathbf{D}_{t,k})$;
                end
            end
        end
    until *convergence*;
    update part templates $Q(\mathbf{Y})$;
until *convergence*;

Algorithm 1: Mean-field inference procedure.

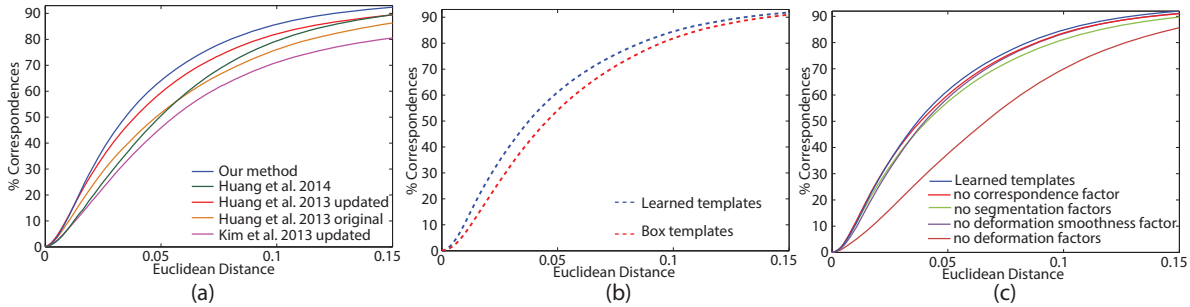

Figure 3.19: Correspondence accuracy of our method in Kim et al.'s benchmark versus (a) previous approaches, (b) using box templates (c) skipping factors from the CRF deformation model.

Correspondence accuracy. We evaluated the performance of our method on the BHCP benchmark by by Kim et al. [59] as in the previous section. We compared our algorithm with previous methods whose authors made their results publically available or agreed to share results with us on the same benchmark: Figure 3.19a demonstrates the performance of our method, the box template fitting method by Kim et al. [59], the local non-rigid registration method by Huang et al. [44], and the functional map network method also by Huang et al [49]. We report the performance of Huang et al.'s method [44] based on the originally published results as well as the latest updated results kindly provided by the authors. We stress that all methods are compared using the same protocol evaluated over all the pairs of the shapes contained in the benchmark, as also done in previous work. Our part templates were initialized based on the co-segmentations provided by Kim et al. (no manual segmentations were used). Our surface prior was learned in a subset of the large datasets used in Kim et al. (1509 airplanes, 408 bikes, 3701 chairs, 406 helicopters). We did not use their whole dataset because we excluded shapes whose provided template fitting error according to their method was above the median error value for airplanes and chairs, and above the 90th percentile for bikes and chairs indicating possible wrong rigid alignment. A few tens of models could also not be downloaded based on the provided original web links. To ensure a fair comparison, we updated the performance of Kim et al. by learning the template parameters in the same subset as ours. Their method had slightly better performance compared to using the original dataset (0.95% larger fraction of

correspondences predicted correctly at distance 0.05). Huang et al.'s reported experiments and results do not make use of the large datasets, but are based on pairwise alignments and networks within the ground-truth sets of the shapes in the benchmark. Figure 3.19a indicates that our method outperforms the other algorithms. In particular, we note that even if we initialized our method with Kim et al.'s segmentations, the final output of our method is significantly better: 18.2% more correct predictions at 0.05 distance than Kim et al.'s method.

We provide images of the corresponding feature points and labeled segmentations for the shapes of our large datasets in Figures 1.2 (left) and 3.20 (left). All these results were produced by initializing our method with the co-segmentations provided by Kim et al. (no manual shape segmentation was used).

Alternative formulations. We now evaluate the performance of our method compared to alternative formulations. We show the performance of our method in the case it does not learn part templates, but instead uses the same mean-field deformation procedure on the box templates provided by Kim et al. In other words, we deform boxes instead of learned parts. Figure 3.19b shows that the correspondence accuracy is significantly better with the learned part templates.

We also evaluate the performance of our method by testing the contribution of the different factors used in the CRF deformation model. Figure 3.19c shows the correspondence accuracy in the same benchmark by using all factors in our model (top curve), without using the unary deformation, deformation smoothness, correspondence or segmentation factors. As shown in the plot, all factors contribute to the improvement of the performance. In particular, skipping the deformation or segmentation parts of the model cause a noticeable performance drop.

Segmentation accuracy. We now report the performance of our method for shape segmentation. We evaluated the segmentation performance on the COSEG dataset [135] and a new dataset we created: we labeled the parts of the 404 shapes used in the BHCP

Category (Dataset)	Num. shapes	Kim et al. (our init.)	Our method	Num. groups
Bikes (BHCP)	100	76.8	82.3	2
Chairs (BHCP)	100	81.2	86.8	2
Helicopters (BHCP)	100	80.1	87.4	1
Planes (BHCP)	104	85.8	89.6	2
Lamps (COSEG)	20	95.2	96.5	1
Chairs (COSEG)	20	96.7	98.5	1
Vase (COSEG)	28	81.3	83.3	2
Quadrupeds (COSEG)	20	86.9	87.9	3
Guitars (COSEG)	44	88.5	89.2	1
Goblets (COSEG)	12	97.6	98.2	1
Candelabra (COSEG)	20	82.4	87.8	3
Large Chairs (COSEG)	400	91.2	92.0	5
Large Vases (COSEG)	300	85.6	83.0	5

Table 3.4: Labeling accuracy of our method versus Kim et al.

correspondences benchmark. We compared our method with Kim et al.'s segmentations in these datasets based on the publically available code and data. These are the same segmentations that we used to initialize our method (no manual segmentations were used). For both methods, we evaluate the labeling accuracy by measuring the fraction of faces for which the part label predicted by our method agrees with the ground-truth label. Since our method provides segmentation at a point cloud level, we transfer the part labels from points to faces using the same graph cuts as in Kim et al. Table 3.4 shows that our method yields better labeling performance. The difference is noticeable in complex shapes, such as helicopters, airplanes, bikes and candelabra. The same table reports the number of clusters (groups) used in our model. We note that our method could be initialized with any other unsupervised technique. This table indicates that our method tends to improve the segmentations it was initialized with.

3.2.5 Summary and Future Extensions

In this section, I described a method to learns part templates, computes shape correspondence and part segmentations. Our part template learning procedure relies on pro-

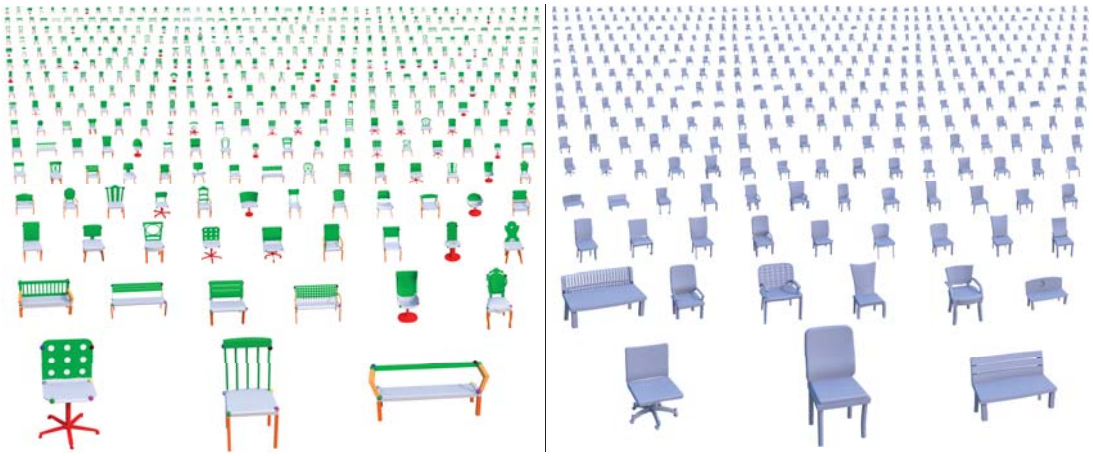

Figure 3.20: Left: Shape correspondences and segmentations for chairs. Right: Synthesis of new chairs

vided initial rigid shape alignments and segmentations, which can sometimes be incorrect. It would be better to fully exploit the power of our probabilistic model to perform rigid alignment. Our method relies on approximate inference for the CRF-based deformation, which might yield suboptimal results. However, as shown in the evaluation, our method still outperforms previous methods for point and part correspondences. In the following Section, I will reinforce this deformation model with a surface prior based on a deep-learned generative model.

3.3 Shape synthesis via learned parametric models of shapes

The dense correspondences established through the deformation model of the previous section allows us to parameterize shapes in terms of corresponding point position and existences. In this section, I will discuss a deep neural network that is built on top of these correspondences. First, I will first present a brief overview of related work on statistical models of 3D shapes and deep learning. Then, I will present how to build a generative model of shapes on top of these correspondences[2]. The key idea is to learn relationships in the surface data hierarchically: our model learns geometric arrangements of points within

[2]This work was published in Computer Graphics Forum 2015. Source code and data are available on our project page: `http://people.cs.umass.edu/~hbhuang/publications/bsm`

individual parts through a first layer of latent variables. Then it encodes interactions between the latent variables of the first layer through a second layer of latent variables whose values correspond to relationships of surface arrangements across different parts. Subsequent layers mediate higher-level relationships related to the overall shape geometry, semantic attributes and structure. The hierarchical architecture of our model is well-aligned with the compositional nature of shapes: shapes usually have a well-defined structure, their structure is defined through parts, parts are made of patches and point arrangements with certain regularities.

3.3.1 Related Work

Statistical models of 3D shapes. Early works developed statistical models of shapes in specific domains, such as faces [13], human bodies [1], and poses [2]. Yingze et al. [10] proposed a statistical model of sparse sets of anchor points by deforming a mean template shape using thin-plate spline (TPS) transformations for shapes, like cars, fruits, and keyboards. Our work can be seen as a generalization of these previous works to domains where shapes differ in both their structure and geometry and where no single template or mean shape can be used.

A number of approaches have tried to model shape variability using Principal Component Analysis or Gaussian Process Latent Variable models on Signed Distance Function (SDF) representations of shapes [106, 23, 92, 25]. However, the dimensionality of SDF voxel-based representations is very high, requiring the input shapes to be discretized at rather coarse voxel grids or use compression schemes that tend to eliminate surface features of the output shapes. Various ad-hoc weights are frequently used to infer coherent SDF values for the output shapes. Other methods use deformable templates made out of oriented boxes [90, 59, 8], and model shape variability in terms of the size and position of these boxes. Xu et al. [145] mixes-and-matches parts through set evolution with part mutations (or deformations) following the box parametrization approach of [90] and part

crossovers controlled by user-speficied fitness scores. Fish et al. [31] and Yumer et al. [151] represent shapes in terms of multiple basic primitives (boxes, planes, cylinders etc) and capture shape variability in terms of primitive sizes, relative orientations, and translations. As a result, these methods do not capture statistical variability at the actual surface level and cannot directly be used to generate new surface point arrangements. Kalogerakis et al. [54] proposed a generative model of shape structure and part descriptors (e.g., curvature histograms, shape diameter, silhouette features). However, these descriptors are not invertible i.e., they cannot be directly mapped back to surface points. As a result, their model cannot be used to output surface geometry and was only used for shape synthesis by retrieving and recombining database parts without further modifications. In constrast to the above approaches, our generative model jointly analyzes and synthesizes 3D shape families, outputs both structure and actual surface geometry, as well as learned class-specific shape descriptors.

Deep Boltzmann Machines. Our statistical model of surface variability follows the structure of a general class of generative models, known as Deep Boltzmann Machines (DBMs) in the machine learning literature. DBMs have been used for speech and image generation [98, 103, 82, 105], and can also be combined with discriminative models [52] for labeling tasks. In a concurrent work, Wu et al. [140] proposed DBMs built over voxel representations of 3D shapes to produce generic shape descriptors and reconstruct low-resolution shapes from input depth maps. Our model in inspired by prior work on DBMs to develop a deep generative model of 3D shape surfaces. Our model has several differences compared to previous DBM formulations. Our model aims at capturing continuous variability of surfaces instead of pixel or voxel interactions. To capture surface variability, we use Beta distributions to model surface point locations instead of Gaussian or Bernoulli distributions often used in DBMs. In addition, the connectivity of layers in our model takes into account the structure of shapes based on their parts. Since shape collections have typically a much smaller size than image or audio collections, and since the geometry of

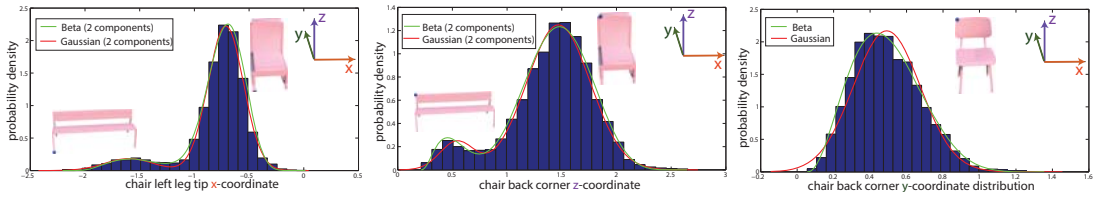

Figure 3.21: Examples of feature point coordinate histograms for chairs and fitted distributions. The fitted Beta distributions are bounded by the minimum and maximum values of the coordinates and fit the underlying probability density more accurately.

surfaces tend to vary smoothly across neighboring points, we use spatial smoothness priors during training. We found that all these adaptations were important to generate surface geometry.

3.3.2 Generative surface model

We now describe a generative probabilistic model whose goal is to characterize surface variability within a shape family. This model is built on the probabilistic shape correspondences and segmentations estimated with the technique described in the previous section. Learning a model that characterizes surface variability poses significant challenges. First, even if we consider individual corresponding surface points, there is no simple probability distribution that can model the variability in their position. For example, Figure 3.21 shows the distributions over coordinates of characteristic feature points across the shapes of four-legged chairs. The coordinates are expressed in an object-centered coordinate system whose origin is located on the chairs' center of mass. We observe that (a) the distributions are not unimodal (Figure 3.21, left and middle), (b) the modes usually correspond to different types of chairs, such as benches or single-person chairs, and similar modes appear for different feature points (Figure 3.21, left and middle), (c) even if the distributions appear to be unimodal, they might not be captured with commonly used symmetric distributions, such as Gaussians (red curves). For example, a Gaussian distribution would assign noticeable probability density for unrealistic chairs whose back would be aligned with the center of mass, or is in front of it (with respect to the frontal viewpoint used to display the chairs in the above figure). Furthermore, there are important correlations in the positions of different

corresponding points that would need to be captured by the generative model: for example, benches are usually associated with wide seats, short backs, and legs placed on the seat corners. Another complication is that shapes vary in structure: for example, some chairs have armrests, while some others do not. The existence of parts depends on the overall type of chairs. Thus, a generative model needs to capture such structural relationships between parts, as well as capture the dependencies of point positions conditioned on the existence of these parts in the input shapes. Finally, even if our 3D shapes are parameterized with a relatively moderate number of consistently localized points (about 5000 points), the dimensionality of our input data is still very large (15000), which also needs to be handled by the learning procedure of our generative model. Following the literature in deep learning, the key idea is to design the generative model such that it capture interactions in the surface data hierarchically through multiple layers of latent variables. We experimented with different numbers of hidden layers, and discuss performance differences in Section 3.3.3. Our surface variability model is defined over the following set of random variables:

Surface point positions $\mathbf{D} = \{\mathbf{D}_k\}$ where $\mathbf{D}_k \in \mathbb{R}^3$ represents the position of a consistently localized point k on an input shape expressed in object-centered coordinates. During the training of our model, these random variables are observed by using the inference procedure of the previous section i.e., they are given by the part template deformations per input shape. We note that we drop the shape index t since this variable is common across all shapes for our generative model.

Surface point existences $\mathbf{E} = \{\mathbf{E}_k\}$ where $\mathbf{E}_k \in \{0, 1\}$ represents whether a point k exists on an input shape. These variables are also observed during training the model of surface variability. They are determined by the inference procedure of the previous section based on which part templates were active or inactive per input shape.

Latent variables for geometry $\mathbf{H} = \{H_m^{(1)}, H_n^{(2)}, H_o^{(3)}\}$ encode relationships of surface point location coordinates at different hierarchical levels. The super-script is an index for the different hidden layers in our model.

Latent variables for structure $\mathbf{G} = \{G_r\}$ encode structural relationships of parts existence in shapes.

Model structure. The generative model is defined through factors that express the interaction degree between the above variables. The choice of the factors was motivated by experimentation and empirical observations. We start our discussion with the factors involving the observed surface point position variables. Given that these are continuously values variables, one way to model the distribution over point positions would be to use Gaussian distributions. However, as shown in Figure 3.21, a Gaussian may incorrectly capture the variability even for a single surface point position. Instead, we use Beta distributions that bound the range of values for a variable, and whose density function is more flexible. The Beta distribution over a single coordinate of an individual point location is defined as follows:

$$P(D_{k,\tau}) = \frac{1}{B} D_{k,\tau}^{a-1} \cdot (1 - D_{k,\tau})^{b-1}$$

where the index $\tau = 1, 2, 3$ refers to the x-,y-, or z-coordinate of the point respectively, B is a normalization factor, a, b are positive-valued parameters that control the shape of the distribution. The distribution is defined over the interval $[0, 1]$, thus in our implementation we normalize all the observed coordinate values within this range. As discussed above, using a single distribution to capture the statistical variability of a point location is still inadequate since the point locations on a surface are not independent of each other. As shown in Figure 3.21, the locations are multi-modal and modes are shared, thus we use latent variables to capture these common modes. Putting these empirical observations together, the interaction between point locations and latent variables can be modeled as follows:

$$\phi(\mathbf{D}, \mathbf{H}^{(1)}) =$$
$$\prod_{k,\tau} D_{k,\tau}^{a_{k,\tau,0}-1} \cdot (1 - D_{k,\tau})^{b_{k,\tau,0}-1}$$
$$\cdot \prod_{k,\tau} \prod_{m \in \mathcal{N}_k} D_{k,\tau}^{a_{k,\tau,m} H_m^{(1)}} \cdot (1 - D_{k,\tau})^{b_{k,\tau,m} H_m^{(1)}}$$
$$\cdot \prod_{k,\tau} \prod_{m \in \mathcal{N}_k} D_{k,\tau}^{c_{k,\tau,m}(1-H_m^{(1)})} \cdot (1 - D_{k,\tau})^{d_{k,\tau,m}(1-H_m^{(1)})}$$

where $a_{k,\tau,m}, b_{k,\tau,m}, c_{k,\tau,m}, d_{k,\tau,m}$ are positive weights expressing how strong is the interaction between each latent variable m with point k, $a_{k,\tau,0}, b_{k,\tau,0}$ are positive bias weights, $\mathcal{N}_k$ denotes the subset of latent random variables of the first layer connected to the variable D_k. To decrease the number of parameters and enforce some sparsity in the model, we split the latent variables in the first layer into groups, where each group corresponds to a semantic part. We model the interaction between each point position with the first hidden layer variables that correspond to the semantic part it belongs to (see also Figure 3.22 for a graphical representation of our model). In this manner, each latent variable of the first layer can be thought of as performing a spatially sensitive convolution on the input points per part (see also Appendix C for the inference equations indicating this convolution).

As shown in Figure 3.21, each group of shapes has its own distinctive parts e.g., benches are associated with wide and short backs. The latent variables of the second layer mediate the inter-dependencies of the part-level latent variables of the first layer. The choice of factors for those interactions are based on sigmoid functions that "activate" part-level geometry modes given certain global shape variability modes:

$$\phi(\mathbf{H}^{(1)}, \mathbf{H}^{(2)}) = \exp\left\{ \sum_m w_{m,0} H_m^{(1)} + \sum_{m,n} w_{m,n} H_m^{(1)} H_n^{(2)} + \sum_n w_{n,0} H_n^{(2)} \right\}$$

where the parameters $w_{m,n}$ control the amount of interaction between the latent variables of the first and second layer, $w_{m,0}$, $w_{n,0}$ are bias weights. Given the above factor, it can be

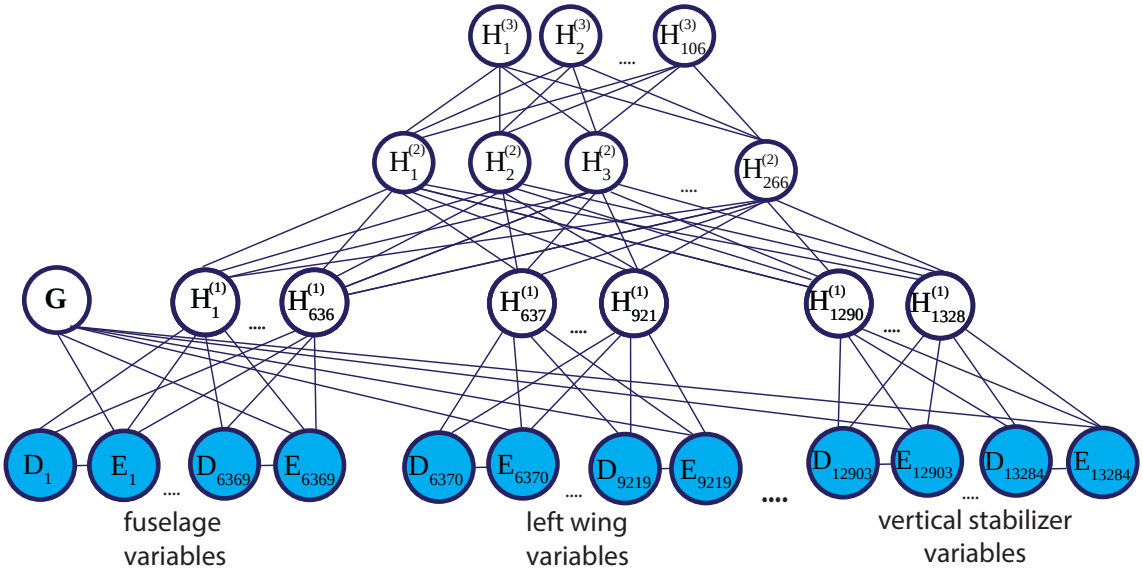

Figure 3.22: Graphical representation of the surface variability model based on our airplane dataset.

seen that each latent variable is activated through sigmoid functions:

$$P(H_m^{(1)} = 1|\mathbf{H}^{(2)}) = \sigma(w_{m,0} + \sum_n w_{m,n} H_n^{(2)})$$

$$P(H_n^{(2)} = 1|\mathbf{H}^{(1)}) = \sigma(w_{n,0} + \sum_m w_{m,n} H_m^{(1)})$$

where $\sigma(x) = 1/(1 + exp(-x))$ represents the sigmoid function.

We model the interactions of the latent variables of the subsequent hidden layers in the same manner. By multiplying all of the above factors, we combine them into a single probability distribution, which has the form of a deep Boltzmann machine [105]. By taking into account that some factors must be deactivated when parts are non-existing in shapes (i.e., when the existence variables for their points are 0), our final probability distribution has the following form:

$$
\begin{aligned}
P_{bsm}(\mathbf{D},\mathbf{H},\mathbf{G},\mathbf{E}) = \frac{1}{Z}\exp\Bigg\{ & \\
& \sum_{k,\tau}(a_{k,\tau,0}-1)\ln(D_{k,\tau})E_k + \sum_{k,\tau}(b_{k,\tau,0}-1)\ln(1-D_{k,\tau})E_k \\
& + \sum_{k,\tau,m\in\mathcal{N}_k} a_{k,\tau,m}\ln(D_{k,\tau})H_m^{(1)}E_k \\
& + \sum_{k,\tau,m\in\mathcal{N}_k} b_{k,\tau,m}\ln(1-D_{k,\tau})H_m^{(1)}E_k \\
& + \sum_{k,\tau,m\in\mathcal{N}_k} c_{k,\tau,m}\ln(D_{k,\tau})(1-H_m^{(1)})E_k \\
& + \sum_{k,\tau,m\in\mathcal{N}_k} d_{k,\tau,m}\ln(1-D_{k,\tau})(1-H_m^{(1)})E_k \\
& + \sum_m w_{m,0}H_m^{(1)} + \sum_{m,n} w_{m,n}H_m^{(1)}H_n^{(2)} + \sum_n w_{n,0}H_n^{(2)} \\
& + \sum_{n,o} w_{n,o}H_n^{(2)}H_o^{(3)} + \sum_o w_{o,0}H_o^{(3)} \\
& + \sum_k w_{k,0}E_k + \sum_{k,r} w_{k,r}E_kG_r + \sum_r w_{r,0}G_r \Bigg\}
\end{aligned}
\tag{3.4}
$$

where Z is a normalization constant. In the following paragraphs, we refer to this model as Beta Shape Machine (BSM) due to the use of Beta distributions to model surface data.

Parameter learning. Learning the parameters of the BSM model poses a number of challenges. Exact maximum likelihood estimation of the parameters is intractable in Deep Boltzmann machines, thus we resort to an approximate learning scheme, called contrastive divergence [63]. Contrastive divergence aims at maximizing the probability gap between the surface data of the input shapes and samples randomly generated by our model. The intuition is that by increasing the probability of the observed data relative to the probability of random samples, the parameters are tuned to model the input data better. The objective function of contrastive divergence is defined as follows:

$$
L_{CD} = \frac{1}{T}\sum_t [\ln \tilde{P}_{bsm}(\boldsymbol{\xi}_t) - \ln \tilde{P}_{bsm}(\boldsymbol{\xi}'_t)]
$$

Figure 3.23: Samples generated from the BSM for airplanes and chairs. The generated shapes are implicitly segmented into labeled parts since each point is colored according to the label of the part template it originated from.

where T is the number of training examples, $\tilde{P}_{bsm}$ is given by Equation 3.4 without the normalization factor Z (known as unnormalized measure), $\boldsymbol{\xi}_t$ is an assignment to the variables of the model given an input shape t and $\boldsymbol{\xi}'_t$ is an assignment to the variables according to a sample perturbed from the same input shape t. The assignments to the variables E_k and $\mathbf{D}_k$ per input shape t are set by checking if the part template exists in the input shape and finding the closest surface point to each point k on the deformed part templates for it respectively. The assignments for the latent variables are computed by performing mean-field inference and using the expectations of their approximating distributions, following Salakhutdinov et al. [105] (see Appendix C for more details). The perturbed sample is generated by inferring the approximating probability distribution of the top-layer binary variables given an input shape, then sampling these binary variables according to their distribution, and finally computing the expectations of the approximating distributions of all the other variables in the model given the sampled values of the top layer.

An additional complication in parameter learning is that even for collections whose shapes are parameterized with a relatively moderate number of points and even with the sparsity in the connections between the observed variables and the first hidden layer, the number of parameters ranges from 5 to 10 million. Yet the available organized 3D shape

collections are limited in size e.g., the corrections we used contain only a few thousand shapes. A key idea in our learning approach is to use strong spatial priors that favor similar weights on the variables representing point positions that are spatially close to each other on average across the input shapes of our collections. To favor spatial smoothness in the learned parameters, we change the above objective function with the following spatial priors and L^1 norm regularization terms. The L^1 norm was desired to eliminate weak hidden node associations causing higher noise when sampling the model and also prevent model overfitting. The new objective function is defined as follows:

$$
\begin{aligned}
L_{CD,spatial} = & \frac{1}{T}\sum_{t}[\ln \tilde{P}_{bsm}(\boldsymbol{\xi}_t) - \ln \tilde{P}_{bsm}(\boldsymbol{\xi}'_t)] \\
& - \lambda_1 \sum_{k,\tau,k'\in\mathcal{N}_k,m} |a_{k,\tau,m} - a_{k',\tau,m}| - \lambda_2 \sum_{k,\tau,m} |a_{k,\tau,m}| \\
& - \lambda_1 \sum_{k,\tau,k'\in\mathcal{N}_k,m} |b_{k,\tau,m} - b_{k',\tau,m}| - \lambda_2 \sum_{k,\tau,m} |b_{k,\tau,m}| \\
& - \lambda_1 \sum_{k,\tau,k'\in\mathcal{N}_k,m} |c_{k,\tau,m} - c_{k',\tau,m}| - \lambda_2 \sum_{k,\tau,m} |c_{k,\tau,m}| \\
& - \lambda_1 \sum_{k,\tau,k'\in\mathcal{N}_k,m} |d_{k,\tau,m} - d_{k',\tau,m}| - \lambda_2 \sum_{k,\tau,m} |d_{k,\tau,m}| \\
& - \lambda_1 \sum_{k,r,k'\in\mathcal{N}_k} |w_{k,r} - w_{k',r}| \\
& - \lambda_2 (\sum_{m,n} |w_{m,n}| - \sum_{n,o} |w_{n,o}| - \sum_{k,r} |w_{k,r}|)
\end{aligned}
$$

where λ_1, λ_2 are regularization parameters, set to 10^{-3} and 10^{-4} respectively in our experiments, $\mathcal{N}(k)$ here denotes all the variables representing point positions whose average distance to point k is less than 10% of the largest distance between a pair of points across all shapes. We perform projected gradient ascent to maximize the above objective function under the constraint that the parameters $a_{k,m}, b_{k,m}, c_{k,m}, d_{k,m}$ are all positive. The same parameters are initialized to random positive numbers according to a uniform distribution on

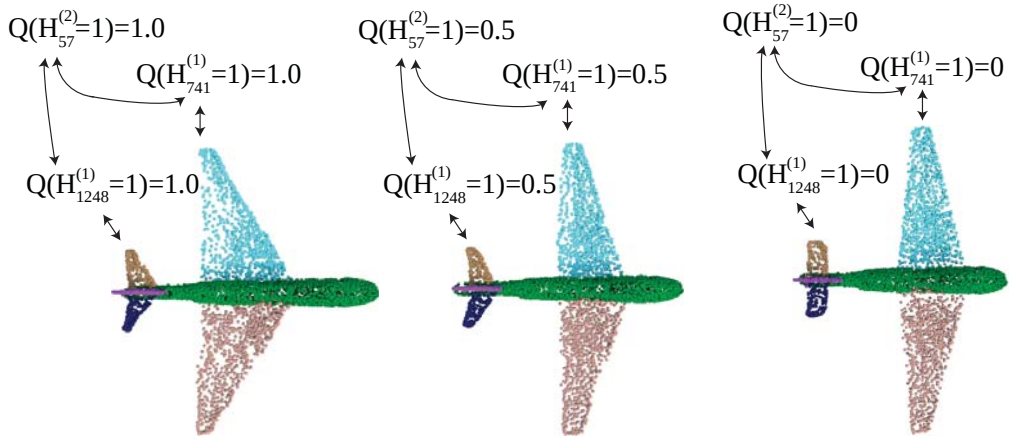

Figure 3.24: Certain latent variables in our learned model have meaningful interpretations. For example, given the leftmost delta-shaped wing arrangement, certain hidden variables of the first layer are activated (i.e., activation probability based on mean-field inference becomes approximately one), while given the rightmost straight wing arrangement, the same variables are deactivated. Similarly, other first layer variables are activated and deactivated for triangular and straight tailplanes respectively. The model also learns that there is a potential statistical correlation between straight wings and tailplanes through variables of the second layer. Initializing the synthesis procedure with interpolated probabilities (e.g., 50%) of these variables generate plausible intermediate configurations of wings and tailplanes (middle).

$[10^{-7}, 10^{-3}]$, while the rest of the weights are initialized according to a normal distribution with mean 0 and variance 0.1.

To avoid local minima during learning, we perform pre-training [105]: we first separately train the parameters between the surface points layer and the first hidden layer, then the parameters of the first and second hidden layer, and so on. Then in the end, our method jointly learns the parameters of the model across all layers. To train our model with contrastive divergence, we also found that it was necessary to apply our training procedure separately on the part of the model involving the interaction parameters between the existence variables E_k and the latent variables $\mathbf{G}$, then the parameters involved in the rest of the model are learned conditioned on the existence variables E_k [74]. Otherwise, the parameters of the model were abruptly diverging due to the three-way interaction of the existence variables, point position variables and the first-layer latent variables. For more details regarding learning and parameter update equations, we refer the reader to Appendix C. Figure 3.24 demonstrates a characteristic example of the information captured in la-

tent variables of our model after learning. In the results section, we discuss fine-grained classification experiments indicating the relationship of the uppermost layer variables with high-level shape attributes, e.g., shape types.

Number of latent variables. A significant factor in the design of the model is the number of latent variables $\mathbf{G}$ and $\mathbf{H}$. Using too few latent variables in the model results in missing important correlations in the data and causing large reconstruction errors during training. Using too many latent variables causes longer training times, and results in capturing redundant correlations. We practically experimented with various numbers of latent variables per layer, and checked performance differences in terms of correspondence accuracy (see joint shape analysis and synthesis paragraph). The best performance was achieved by selecting the number of latent variables of the first layer to be $1/10th$ of the total number of the point position variables $\mathbf{D}$ per part. The number of latent variables in the second layer was $1/5th$ of the number of latent variables in the first layer, and the number of latent variables in the third layer $\mathbf{H}^{(3)}$ was $1/2.5th$ the number of latent variables in the second layer. The number of latent variables $\mathbf{G}$ was set equal to the total number of latent variables $\mathbf{H}^{(1)}$.

Shape synthesis. To perform shape synthesis with the generative model, we first sample the binary variables of the top hidden layer, then we alternate twice between top-down and bottom-up mean-field inference in the model. We then compute the expectations of the surface point existences and location variables based on their inferred distributions. The result is a point cloud representing the surface of a new shape. Figure 3.23 shows representative sampled point clouds. The point clouds consists of 5000 points. Due to the approximate nature of learning and inference on our model, the samples do not lie necessarily on a perfect surface. Due to their low number, we cannot apply a direct surface reconstruction technique. Instead, we find parts in the training shapes whose corresponding points are closest to the sampled point positions based on their average Euclidean distances. Then we apply the embedded deformation method [125] to preserve the local surface detail

of the used mesh parts, and deform them smoothly and conservatively towards the sampled points. A deformation graph of 100 nodes is constructed per part, and nodes are deformed according to the nearest sampled points. We use the following parameters found in [125]: $w_{rot} = 3$, $w_{reg} = 15$, $w_{con} = 120$ and set the deformation node neighborhood size equal to 20. Figure 3.26 shows synthesized shapes based on the sampled point clouds and the closest training shape parts (in blue). As it can been, the model learns important constraints for generating plausible shapes, such as symmetries and functional part arrangements.

Joint shape analysis and synthesis. The BSM model can be used as a surface prior to improve shape correspondence. To do this, we combine the deformation model of the previous section and the BSM generative model into a single joint probability distribution, and find the most likely joint assignment to all variables:

$$\text{maximize} \;\; P_{crf}(\mathbf{Y}, \mathbf{U}, \mathbf{S}, \mathbf{D}|\mathbf{X}) \cdot P_{bsm}(\mathbf{D}, \mathbf{H}, \mathbf{G}, \mathbf{E}) \tag{3.5}$$

The most likely joint assignment to the above variables is found again with mean-field inference. First, we apply the Algorithm 1 to compute initial correspondences and segmentations, and train the BSM model based on the estimated correspondences and segmentations. Then we perform mean-field inference on the joint model to compute the most likely assignments to all variables based on their inferred approximating distribution modes, yielding improved point correspondences. We alternate between training the BSM model and mean-field inference 3 times in our experiments, after which we did not notice any considerable improvement.

3.3.3 Evaluation

We now describe the experimental validation of the BSM model for computing semantic point correspondences, fine-grained classification, and synthesis.

Correspondences. As mentioned above, the BSM model can be used as a surface prior to improve shape correspondences using MAP inference on the probabilistic model

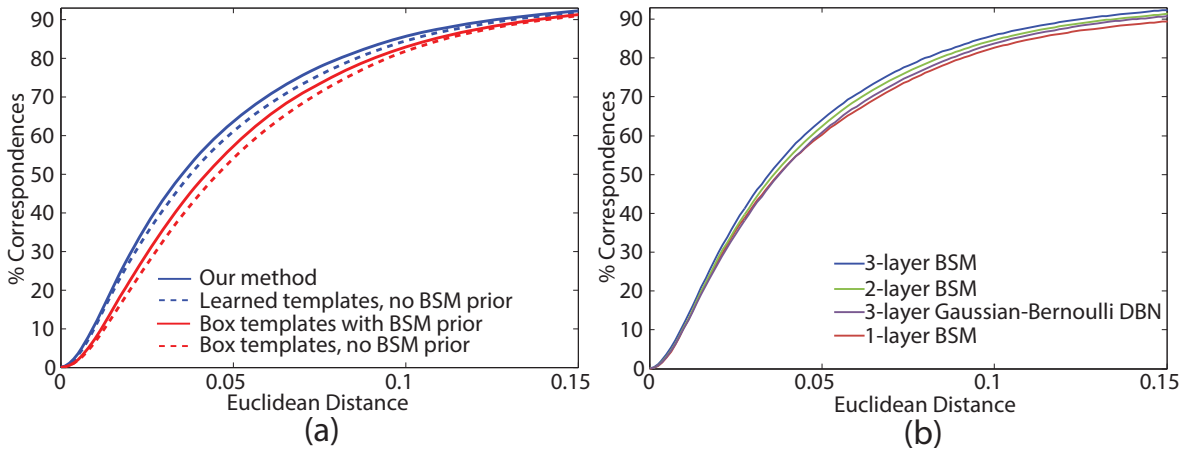

Figure 3.25: Correspondence accuracy of BSM in Kim et al.'s benchmark versus (a) previous approaches and CRF model, (b) using box templates and the BSM surface prior (c) versus alternative Deep Boltzmann Machine formulations.

of Equation 3.5. We evaluated the performance of our method on the BHCP benchmark provided by Kim et al. [57] based on the same methodology described in Section 3.2.3. In Figure 3.25a, we show the performance of the two versions of our correspondence method using the CRF model of Section 3.2 alone and using the CRF model together with the BSM surface prior. We note that using the BSM prior improves correspondence accuracy either in the case of box or learned part templates. We also evaluated the performance by using alternative formulations of the BSM surface prior, shown in Figure 3.25b. Our BSM version, discussed in the main text, achieves the best performance.

Fine-grained classification. The uppermost layer of the BSM model can be used to produce a compact, class-specific shape descriptor. We demonstrate its use in the context of fine-grained classification. For this purpose, we labeled the BHCP benchmark shapes with fine-grained class labels: we categorized airplanes into commercial jets, fighter jets, propeller aircraft and UAVs, chairs into benches, armchairs, side and swivel chairs, and bikes into bicycles, tricycles and motorbikes. We compute activation probabilities in the uppermost layer through mean-field inference given each input shape, and used those as descriptors. Using 10 training examples per class, and a single nearest neighbor classifica-

Figure 3.26: Synthesis of new shapes (left) based on the Beta Shape Machine samples (green box) and embedded deformation on input shape parts (right).

tion scheme based on L^1 norm distance, the rest of the shapes were classified with accuracy 92%, 94%, 96.5% for airplanes, chairs, and bikes respectively.

Shape synthesis. Figure 1.3(right) and 3.20(right) demonstrates synthesized chairs and airplanes using samples from the BSM model trained on these large collections. Our shape synthesis procedure makes use of the shape parts segmented by our method in these collections. However, not all shapes are segmented perfectly: even if the labeling accuracy of our method is high as demonstrated above, minor errors along segmentation boundaries (e.g., mislabeled individual faces crossing boundaries) cause visual artifacts when shapes are assembled from these segmented parts. Such errors are common in graph cuts. Corrections

would require re-meshing or other low-level mesh operations. We instead manually flagged 25% of airplanes and 40% of chairs with imperfect segmentation boundaries. These were excluded during the nearest neighbors procedure for selecting parts given the BSM samples. We still believe that the amount of human supervision for this process is much smaller compared to previous generative models [54] that required manually specified shape segmentations for at least half or the whole input collections. We also conducted a perceptual evaluation of our synthesized shapes with 31 volunteers recruited through Amazon Mechanical Turk. Our user study indicates that the shapes produced by our model were seen as plausible as the training shapes of the input collections.

Implementation and running times. Our method is implemented in C++ and is CPU-based. Learning the BSM model requires 45 hours for our largest dataset (3K chairs) with a E5-2697 v2 processor. Running times scale linearly with the dataset size.

3.3.4 User study

We conducted a perceptual evaluation of our synthesized shapes with volunteers recruited through Amazon Mechanical Turk. Each volunteer performed 40 pairwise comparisons in a web-based questionnaire. Each comparison was between images of two shapes from the chairs or airplanes domain. For each pair, one shape was coming from the training collection of one of the two domains, and the other shape was coming from our collection of synthesized shapes of the same domain. The two shapes were randomly sampled from their collections. They appeared in a random order on the web pages of the questionnaire.

The participants were asked to choose which of the two presented shapes was more plausible, or indicate whether they found both shapes to be equally plausible, or none of them to be plausible. Each questionnaire contained 20 unique comparisons and each comparison was repeated twice by flipping the order of the two shapes in the question. To diminish the risk of contaminating the results of the user study with unreliable respondents,

we excluded participants that gave two different answers to more than 6 of the 20 unique comparisons, or took less than 2 minutes in total to complete the questionnaire.

The number of reliable Mechanical Turk respondents after the above filtering was 31. A total of 1240 pairwise comparisons were gathered in total. The results are visualized in the following figure. The user study indicates that the shapes produced by our model were seen as plausible as the training shapes of the input collection.

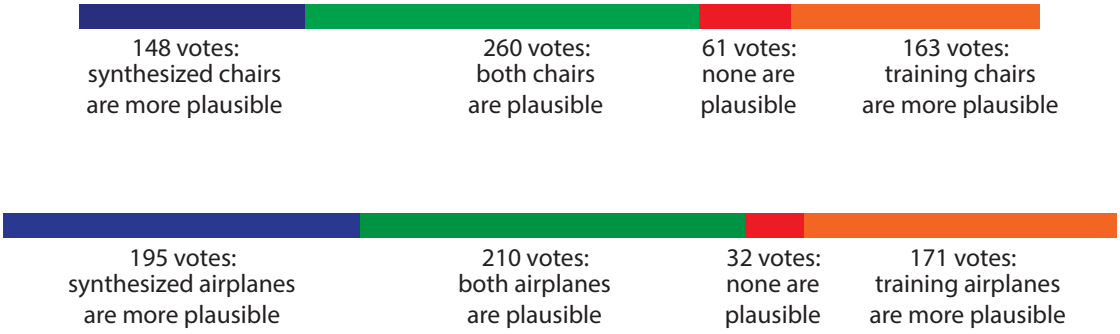

Figure 3.27: Results of our Amazon Mechanical Turk user study

3.3.5 Summary and Future Extensions

In this section, I described a method for joint shape analysis and synthesis in a shape collection: our method learns part templates, computes shape correspondence and part segmentations (as in previous section), and in addition, it also generates new shape surface variations, and yields shape descriptors for fine-grained classification. Our method represents an early attempt in this area, thus there are several limitations to our method and many exciting directions for future work. First, our method is greedy in nature. Our method relies on approximate inference for both the CRF deformation and the BSM generative model. Learning relies on approximate techniques. As a result, the sampled point clouds are not smooth and noiseless. We used conservative deformations of parts from the input collection to factor out the noise and preserve surface detail during shape synthesis. Assembling shapes from parts suffers from various limitations: adjacent parts are

not always connected in a plausible manner, segmentation artifacts affect the quality of the produced shapes, topology changes are not supported. Instead of re-using parts from the input collection, it would be more desirable to extend our generative model with layers that produce denser point clouds. In this case, the denser point clouds could be used as input to surface reconstruction techniques to create new shapes entirely from scratch. However, the computational cost for learning such generative model with dense output would be much higher. From this aspect, it would be interesting to explore more efficient learning techniques in the future. Deep learning architectures could be used to estimate initial segmentations. The learned shape descriptor could improve the shape grouping. Finally, the variability of the synthesized shapes seems somewhat limited. Fruitful directions include investigating deeper architectures, better sampling strategies, and matching templates with multiple symmetric parts if such exist in the input shapes.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

In this thesis, I have discussed deep learning algorithms that are capable of automatically inferring parametric representations of shape families, which could be used to generate new 3D shapes from high-level user specifications, such as freehand sketches. The goal is to provide users intuitive tools that enable rapid and easy creation of detailed shapes. With these modeling algorithms, I hope to significantly shorten the design cycle of 3D products and make it easy for users to create complex and plausible shapes.

This thesis only represented a first step towards building generative representations of 3D shapes. Concurrently to my work, other deep generative models have been proposed based on volumetric shape representations [138], view-based shape representations [4], and point sets [30] that do not rely on point correspondences. However, they all tend to build coarse shapes without much surface detail that artists would expect. Most importantly, in all these works, there is lack of interactive control in the produced shapes. It would be interesting to combine these new generative models with our pipeline proposed in Chapter 2 and control the output of these models interactively through sketches. Controlling shape synthesis through other input modalities, such as gestures or natural language wold be also an exciting direction. Finally, another highly unexplored area is how to couple these generative models of shapes with physical shape representations. The shapes created by current generative shape models are not guaranteed to be manufacturable or printable. Investigating deep architectures that learn to synthesize physical parameters of shapes would be a promising step towards this direction.

APPENDIX A

PUBLICATION LIST

Here is a list of my publications during my MS/PhD program

- Zhaoliang Lun, Changqing Zou, **Haibin Huang**, Evangelos Kalogerakis, Ping Tan, Marie-Paule Cani, Hao Zhang "Learning to Group Discrete Graphical Patterns ", conditionally accepted to SIGGRAPH ASIA 2017

- Xiaoguang Han, Zhen Li, **Haibin Huang**, Evangelos Kalogerakis, Yizhou Yu "High Resolution Shape Completion Using Deep Neural Networks for Global Structure and Local Geometry Inference ", IEEE International Conference on Computer Vision(ICCV) 2017

- **Haibin Huang**, Evangelos Kalogerakis, Siddhartha Chaudhuri, Duygu Ceylan, Vladimir G. Kim, M. Ersin Yumer"Learning Local Shape Descriptors with View-based Convolutional Neural Networks ", conditionally accepted to TOG 2017

- Amir Arsalan Soltani, **Haibin Huang**, Jiajun Wu, Tejas Kulkarni, Joshua Tenenbaum "Synthesizing 3D Shapes via Modeling Multi-View Depth Maps and Silhouettes with Deep Generative Networks ", IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017

- **Haibin Huang**, Evangelos Kalogerakis, M. Ersin Yumer , Radomír Měch "Shape Synthesis from Sketches via Procedural Models and Convolutional Networks ", IEEE Transactions on Visualization and Computer Graphics 2017

- **Haibin Huang**, Evangelos Kalogerakis, Benjamin Marlin "Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces ", Computer Graphics Forum, Vol. 34, No. 5, 2015 (Proceedings of SGP 2015)

- Chongyang Ma, **Haibin Huang**, Alla Sheffer, Evangelos Kalogerakis, Rui Wang "Analogy-Driven 3D Style Transfer ", Computer Graphics Forum, Vol 33, Issue 2, 175–184 (Eurographics 2014)

- Yahan Zhou, **Haibin Huang**, Li-Yi Wei and Rui Wang, "Point Sampling with General Noise Spectrum", ACM Trans Graph. 31(4) (SIGGRAPH 2012), pp.76:01-76:11

APPENDIX B

CNN IMPLEMENTATION DETAILS FOR SECTION 2.3

We provide here details about our CNN implementation and the transformations used in its convolutional and pooling layers.

Architecture implementation Each of our two sub-networks follow the structure of AlexNet [64]. In general, any deep convolutional neural network, reasonably pre-trained on image datasets, could be used instead. We summarize the details of AlexNet structure for completeness. The first convolutional layer processes the 227x227 input image with 96 filters of size 11x11. Each filter is applied to each image window with a separation (stride) of 4 pixels. In our case, the input image has a single intensity channel (instead of the three RGB channels used in computer vision pipelines). The second convolutional layer takes as input the max-pooled output of the first convolutional layer and processes it with 256 filters of size 5x5x48. The third convolutional layer processes the max-pooled output of the second convolutional layer with 384 filters of size 3x3x256. The fourth and fifth convolutional layers process the output of the third and fourth convolutional layer respectively with 384 filters of size 3x3x192. There are two fully connected layers contain 4096 processing functions (nodes) each. Finally, the regression layer contains as many regression functions as the number of the PM continuous parameters, and the classification layer contains as many softmax functions as the number of PM discrete parameters. The number of the PM discrete and continuous parameters depend on the rule set (statistics are described in Section 3.1.5). The architecture is implemented using the Caffe [50] library.

Convolutional layer formulation Mathematically, each convolution filter k in a layer l produces a feature map (i.e., a 2D array of values) $h_{k,l}$ based on the following transfor-

mation:

$$h_{k,l}[i,j] = f\Big(\sum_{u=1}^{N}\sum_{v=1}^{N}\sum_{m\in\mathcal{M}} w_{k,l}[u,v,m]\cdot h_{m,l-1}[i+u,j+v] + b_{k,l}\Big) \tag{B.1}$$

where i, j are array (pixel) indices of the output feature map h, $\mathcal{M}$ is a set of feature maps produced in the previous layer (with index $l-1$), m is an index for each such input feature map $h_{m,l-1}$ produced in the previous layer, NxN is the filter size, u and v are array indices for the filter. Each filter is three-dimensional, defined by NxNx$|\mathcal{M}|$ learned weights stored in the 3D array $w_{k,l}$ as well as a bias weight $b_{k,l}$. In the case of the first convolutional layer, the input is the image itself (a single intensity channel), thus its filters are two-dimensional (i.e., $|\mathcal{M}| = 1$ for the first convolutional layer). Following [64], the response of each filter is non-linearly transformed and normalized through a function f. Let $x = h_{k,l}[i,j]$ be a filter response at a particular pixel position i, j. The response is first non-linearly transformed through a rectifier activation function that prunes negative responses $f_1(x) = max(0,x)$, and a contrast normalization function that normalizes the rectified response according to the outputs $x_{k'}$ of other filters in the same layer and in the same pixel position: $f_2(x) = [x/(\alpha+\beta\sum_{k'\in\mathcal{K}} x_{k'}^2)]^\gamma$ [64]. The parameters α, β, γ, and the filters $\mathcal{K}$ used in contrast normalization are set according to the cross-validation procedure of [64] ($\alpha = 2.0$, $\beta = 10^{-4}$, $\gamma = 0.75$, $|\mathcal{K}| = 5$).

Pooling layer formulation The transformations used in max-pooling are expressed as follows:

$$h_{k,l}[i,j] = \max\Big\{h_{k,l-1}[u,v]\Big\}_{i<u<i+N, j<v<j+N}$$

where k is an index for both the input and output feature map, l is a layer index, i, j represent output pixel positions, and u, v represent pixel positions in the input feature map.

APPENDIX C

IMPLEMENTATION DETAILS OF MEAN-FIELD INFERENCE AND BSM MODEL OF SECTIONS 3.2 AND 3.3

C.1 Mean-field inference equations

According to the mean-field approximation theory [63], given a probability distribution P defined over a set of variables $X_1, X_2, ..., X_V$, we can approximate it with a simpler distribution Q, expressed as a product of individual distributions over each variable, such that the KL-divergence of P from Q is minimized:

$$KL(Q \mid\mid P) = \sum_{X_1} \sum_{X_2} ... \sum_{X_V} Q(X_1, X_2, ..., X_V) \cdot \ln \frac{Q(X_1, X_2, ..., X_V)}{P(X_1, X_2, ..., X_V)}$$

In the case of continuous variables, the above sums are replaced with integrals over their value space. Suppose that the original distribution P is defined as a product of factors:

$$P(X_1, X_2, ..., X_V) = \frac{1}{Z} \prod_{s=1...S} \phi_s(\mathbf{D}_s)$$

where $\mathbf{D}_s$ is a subset of the random variables (called scope) for each factor s in the distribution P, and Z is a normalization constant.

Minimizing the KL-divergence of P from Q yields the following mean-field updates for each variable X_v ($v = 1...V$):

$$Q(X_v) = \frac{1}{Z_v} \exp\left\{ \sum_s \sum_{\mathbf{D}_s - \{X_v\}} Q(\mathbf{D}_s - \{X_v\}) \ln \phi_s(\mathbf{D}_s) \right\}$$

where $Z_v = \sum_{X_v} Q(X_v)$ is a normalization constant for this distribution (the sum is replaced with the integral over the value space of X_v if this is a continuous variable), and $\mathbf{D}_s - \{X_v\}$ is the subset of the random variables for the factor s excluding the variable X_v.

Below we specialize the above update formula for each of our variable in our probabilistic model.

C.1.1 Deformation variables

The mean-field update for each deformation variables is the following:

$$Q(\mathbf{D}_{t,k}) \propto \exp\Big\{ -0.5 \sum_{p} Q(U_{t,p} = k)(\mathbf{D}_{t,k} - \mathbf{X}_{t,p})^T \Sigma_1^{-1} (\mathbf{D}_{t,k} - \mathbf{X}_{t,p}) \\ -0.5 \sum_{k' \in N(k)} (\mathbf{D}_{t,k} - \boldsymbol{\mu}_{t,k,k'})^T \Sigma_2^{-1} (\mathbf{D}_{t,k} - \boldsymbol{\mu}_{t,k,k'}) \Big\}$$

where $\mathcal{N}(k)$ includes all neighboring points k' of point k on the part template (see main text of the paper) and $\boldsymbol{\mu}_{t,k}$ is a 3D vector defined as follows:

$$\boldsymbol{\mu}_{t,k,k'} = \mathbb{E}_Q[\mathbf{D}_{t,k'}] + (\mathbb{E}_Q[\mathbf{Y}_k] - \mathbb{E}_Q[\mathbf{Y}_{k'}]))$$

We note that the above distribution is a product of Gaussians; when re-normalized, the distribution is equivalent to a Gaussian with the following expectation, or mean, which we use in other updates:

$$\mathbb{E}_Q[\mathbf{D}_{t,k}] = (\sum_{p} Q(U_{t,p} = k)\Sigma_1^{-1} + \sum_{k' \in N(k)} \Sigma_2^{-1})^{-1} \\ \cdot (\sum_{p} Q(U_{t,p} = k)\Sigma_1^{-1}\mathbf{X}_{t,p} + \sum_{k' \in N(k)} \Sigma_2^{-1}\boldsymbol{\mu}_{t,k,k'}) \tag{C.1}$$

The above formula indicates that the most likely deformed position of a point on a part template is a weighted average of its associated points on the input surface as well as its

neighbors. The weights are controlled by the covariance matrices Σ_1 and Σ_2 as well as the degree of association between the part template point and each input surface point, given by $Q(U_{t,p} = k)$. The covariance matrix of the above distribution is forced to be diagonal (see next section); its diagonal elements tend to increase when the input surface points have weak associations with the part template point, as indicated by the following formula:

$$Cov_Q[\mathbf{D}_{t,k}] = (\sum_p Q(U_{t,p} = k)\Sigma_1^{-1} + \sum_{k' \in N(k)} \Sigma_2^{-1})^{-1}$$

Computing the above expectation and covariance for each variable $\mathbf{D}_{t,k}$ involving summing over every surface point p on the input shape t. This is computationally too expensive. Practically in our implementation, we find the 100 nearest input surface points for each part template point k, and we also find the 20 nearest part template points for each input surface point p. For each template point k, we always keep indices to its 100 nearest surface points as well as the surface points whose nearest points include that template point k. Instead of summing over all the surface points of each input shape, for each template point k we sum over its abovementioned indices to surface point only. For the rest of the surface points, the distribution values $Q(U_{t,p} = k)$ are practically negligible and are skipped in the above summations.

C.1.2 Part template variables

For each part template point $\mathbf{Y}_k$, the mean-field update is given by:

$$Q(\mathbf{Y}_k) \propto \exp\left\{ -.5(\mathbf{Y}_k - \boldsymbol{\mu}_k)^T \Sigma_2^{-1} (\mathbf{Y}_k - \boldsymbol{\mu}_k) \right\}$$

where $\boldsymbol{\mu}_k$ is the mean, or expectation (3D vector), defined as follows:

$$\boldsymbol{\mu}_k = \mathbb{E}_Q[\mathbf{Y}_k] = \frac{1}{|\mathcal{N}(k)|} \sum_{k'} \left(\mathbb{E}_Q[\mathbf{Y}_{k'}] + \frac{1}{T} \sum_t (\mathbb{E}_Q[\mathbf{D}_{t,k}] - \mathbb{E}_Q[\mathbf{D}_{t,k'}]) \right)$$

and $\mathcal{N}(k)$ includes all neighboring points k' of point k on the part template, T is the number of input shapes. The covariance matrix for the above distribution is given by Σ_2.

C.1.3 Point correspondence variables

The mean-field update for the latent variables $U_{t,p}$ yields a categorical distribution computed as follows:

$$\begin{aligned} Q(U_{t,p} = k) \propto \exp \Big\{ & -0.5(\mathbb{E}_Q[\mathbf{D}_{t,k}] - \mathbf{X}_{t,p})^T \Sigma_1^{-1} (\mathbb{E}_Q[\mathbf{D}_{t,k}] - \mathbf{X}_{t,p}) \\ & - 0.5 Tr(\Sigma_1^{-1} \cdot Cov_Q[\mathbf{D}_{t,k}]) \\ & - 0.5(\mathbf{f}_k - \mathbf{f}_{t,p})^T \Sigma_3^{-1} (\mathbf{f}_k - \mathbf{f}_{t,p}) - \ln \epsilon \cdot Q(S_{t,p} = label(k)) \Big\} \end{aligned}$$

where $Tr(\Sigma_1^{-1} \cdot Cov_Q[\mathbf{D}_{t,k}])$ represents the matrix trace, ϵ is a small constant discussed in the main text of the paper. For computational efficiency reasons, we avoid computing the above distribution for all pairs of part template and input surface points. As in the case of the updates for the deformation variables, we keep indices to input surface point positions that are nearest neighbors to part template points and vice versa. We compute the above distributions only for pairs between these neighboring points. For the rest of the pairs, we set $Q(U_{t,p} = k) = 0$.

C.1.4 Segmentation variables

The mean-field update for the variables $S_{t,p}$ also yields a categorical distribution:

$$Q(S_{t,p}=l) \propto \exp\Big\{ \sum_{k} Q(U_{t,p}=k)[label(k)\neq l]\ln\epsilon + \sum_{p'\in N(p)}\sum_{l'\neq l} Q(S_{t,p'}=l')\ln(1.0-\Phi) + \sum_{p'\in N(p)} Q(S_{t,p'}=l)\ln(\Phi)\Big\}$$

where $N(p)$ is the neighborhood of the input surface point used for segmentation (see main text for more details), Phi evaluates feature differences between neighboring surface points (see main text for its definition). The binary indicator function $[label(k) \neq l]$ is 1 if the expression in brackets holds, otherwise it is 0.

C.2 Covariance matrix updates

The covariance matrices of our factors are updated as follows:

$$\Sigma_1 = \frac{1}{Z_1}\sum_{t,k,p\in\mathcal{N}(k)} Q(U_{t,p}=k)(\mathbb{E}_Q[\mathbf{D}_{t,k}]-\mathbf{X}_{t,p})(\mathbb{E}_Q[\mathbf{D}_{t,k}]-\mathbf{X}_{t,p})^T$$

$$\Sigma_2 = \frac{1}{Z_2}\sum_{t,k,k'\in N(k)} ((\mathbb{E}_Q[\mathbf{D}_{t,k}]-\mathbb{E}_Q[\mathbf{D}_{t,k'}])-(\mathbb{E}_Q[\mathbf{Y}_k]-\mathbb{E}_Q[\mathbf{Y}_{k'}]))\cdot \cdot(\mathbb{E}_Q[\mathbf{D}_{t,k}]-\mathbb{E}_Q[\mathbf{D}_{t,k'}]-(\mathbb{E}_Q[\mathbf{Y}_k]-\mathbb{E}_Q[\mathbf{Y}_{k'}]))^T$$

$$\Sigma_3 = \frac{1}{Z_3}\sum_{t,k,p\in\mathcal{N}(k)} Q(U_{t,p}=k)(\mathbf{f}_k-\mathbf{f}_{t,p})(\mathbf{f}_k-\mathbf{f}_{t,p})^T$$

$$\Sigma_5 = \frac{1}{Z_5} \sum_{t,p,p' \in N(p)} (\mathbf{f}_{t,p} - \mathbf{f}_{t,p'})(\mathbf{f}_{t,p} - \mathbf{f}_{t,p'})^T$$

where $Z_1 = Z_3 = \sum_{t,k,p \in \mathcal{N}(k)} Q(U_{t,p} = k)$, $Z_2 = \sum_{t,k,k' \in N(k)} 1$, $Z_5 = \sum_{t,p,p' \in N(p)} 1$. The computed covariance matrices are forced to be diagonal i.e., in the above computations only the diagonal elements of the covariance matrices are taken into account, while the rest of the elements are set to 0.

C.3 Contrastive divergence

Contrastive divergence iterates over the following three steps in our implementation: variational (mean-field) inference, stochastic approximation (or sampling), and parameter updates. We discuss the steps in detail below.

Variational inference step. Our deformation model yields expectations over deformed point positions of the part templates based on Equation C.1. For each deformed point, we find the surface point that is closest to its expected position. Let $D_{k,\tau}[t]$ the observed surface position of point k for an input shape t. The subscript τ takes values 1, 2, or 3 that correspond to the $x-$,$y-$,$z-$ coordinate of the point respectively. Let $E_k[t]$ represents the observed existence of a point k (binary variable) also inferred by our deformation model. Given all observed point positions $\mathbf{D}[t]$ and existences $\mathbf{E}[t]$ per shape t, we perform bottom-up mean-field inference on the binary hidden nodes according to the following equations in the following order:

$$Q(H_m^{(1)}=1|\mathbf{D}[t],\mathbf{E}[t])=\sigma\Bigg(w_{m,0}+\sum_{k\in\mathcal{N}_m,\tau}(a_{k,\tau,m}-c_{k,\tau,m})\ln(D_{k,\tau}[t])E_k[t]$$
$$+\sum_{k\in\mathcal{N}_m,\tau}(b_{k,\tau,m}-d_{k,\tau,m})\ln(1-D_{k,\tau}[t])E_k[t]$$
$$+\sum_n w_{m,n}Q(H_n^{(2)}=1|\mathbf{D}[t],\mathbf{E}[t])\Bigg)$$

$$Q(H_n^{(2)}=1|\mathbf{D}[t],\mathbf{E}[t])=\sigma\Bigg(w_{n,0}+\sum_m w_{m,n}Q(H_m^{(1)}=1|\mathbf{D}[t],\mathbf{E}[t])$$
$$+\sum_o w_{n,o}Q(H_o^{(3)}=1|\mathbf{D}[t],\mathbf{E}[t])\Bigg)$$

$$Q(H_o^{(3)}=1|\mathbf{D}[t],\mathbf{E}[t])=\sigma\Bigg(w_{o,0}+\sum_n w_{n,o}Q(H_n^{(2)}=1|\mathbf{D}[t],\mathbf{E}[t])\Bigg)$$

where $\sigma(\cdot)$ represents the sigmoid function, $\mathcal{N}_m$ is the set of the observed variables each hidden node (variable) $H_m^{(1)}$ is connected to. The mean-field updates for $H_m^{(1)}$ involve a weighted summation over the observed variables $\mathbf{D}[t]$ per part, which can be thought of as a convolutional scheme per part. We perform 3 mean-field iterations alternating the updates over the above hidden nodes. During the first iteration, we initialize $Q(H_n^{(2)}=1)=0$ and $Q(H_o^{(3)}=1)=0$ for each hidden node n and o.

Stochastic approximation. This step begins by sampling the binary hidden nodes of the top layer for each training shape t. Sampling is performed according to the inferred distributions $Q(H_o^{(3)}=1|\mathbf{D}[t],\mathbf{E}[t])$ of the previous step. Let $H_o^{(3)}[t']$ the resulting sampled $0/1$ values. Then we perform top-down mean-field inference on the binary hidden nodes of the other layers as well as the visible layer:

$$Q(H_n^{(2)} = 1|\mathbf{E}[t], \mathbf{H}^{(3)}[t']) = \sigma\Bigg(w_{n,0} + \sum_m w_{m,n} Q(H_m^{(1)} = 1|\mathbf{E}[t], \mathbf{H}^{(3)}[t']) + \sum_o w_{n,o} H_o^{(3)}[t'] \Bigg)$$

$$Q(H_m^{(1)} = 1|\mathbf{E}[t], \mathbf{H}^{(3)}[t']) = \sigma\Bigg(w_{m,0} + \sum_{k \in \mathcal{N}_m, \tau} (a_{k,\tau,m} - c_{k,\tau,m}) \ln(D_{k,\tau}[t']) E_k[t] + \sum_{k \in \mathcal{N}_m, \tau} (b_{k,\tau,m} - d_{k,\tau,m}) \ln(1 - D_{k,\tau}[t']) E_k[t] + \sum_n w_{m,n} Q(H_n^{(2)} = 1|\mathbf{E}[t], \mathbf{H}^{(3)}[t']) \Bigg)$$

$$Q(D_{k,\tau}|\mathbf{E}[t], \mathbf{H}^{(3)}[t']) \propto D_{k,\tau}^{(a_{k,\tau,0}-1)+\sum_{m \in \mathcal{N}_k} a_{k,\tau,m} Q(H_m^{(1)}=1|\mathbf{E}[t],\mathbf{H}^{(3)}[t'])+\sum_{m \in \mathcal{N}_k} c_{k,\tau,m}(1-Q(H_m^{(1)}=1|\mathbf{E}[t],\mathbf{H}^{(3)}[t']))} (1 - D_{k,\tau})^{(b_{k,\tau,0}-1)+\sum_{m \in \mathcal{N}_k} b_{k,\tau,m} Q(H_m^{(1)}=1|\mathbf{E}[t],\mathbf{H}^{(3)}[t'])+\sum_{m \in \mathcal{N}_k} d_{k,\tau,m}(1-Q(H_m^{(1)}=1|\mathbf{E}[t],\mathbf{H}^{(3)}[t']))}$$

where $D_{k,\tau}[t']$ in the above mean-field updates is set to be the expectation of the above Beta distribution and $\mathcal{N}_k$ is the set of the hidden variables each observed node (variable) $D_{k,\tau}$ is connected to. We note that sampling all the variables in the model caused contrastive divergence not to converge, thus we instead used expectations of the above distributions instead. As in the previous step, we performed 3 iterations alternating over the above mean-field updates. During the first iteration, we skip the terms involving $D_{k,\tau}[t']$ during the inference of the hidden nodes of the first layer. At the second and third iteration, we infer distributions for the hidden layers as follows:

$$Q(H_o^{(3)} = 1|\mathbf{D}[t'], \mathbf{E}[t]) = \sigma\Bigg(w_{o,0} + \sum_n w_{n,o} Q(H_n^{(2)} = 1|\mathbf{D}[t'], \mathbf{E}[t]) \Bigg)$$

$$Q(H_n^{(2)} = 1|\mathbf{D}[t'], \mathbf{E}[t]) = \sigma\Bigg(w_{n,0} + \sum_m w_{m,n} Q(H_m^{(1)} = 1|\mathbf{D}[t'], \mathbf{E}[t]) + \sum_o w_{n,o} Q(H_o^{(3)} = 1|\mathbf{D}[t'], \mathbf{E}[t])\Bigg)$$

$$Q(H_m^{(1)} = 1|\mathbf{D}[t'], \mathbf{E}[t]]) = \sigma\Bigg(w_{m,0} + \sum_{k\in\mathcal{N}_m,\tau} (a_{k,\tau,m} - c_{k,\tau,m}) \ln(D_{k,\tau}[t']) E_k[t] + \sum_{k\in\mathcal{N}_m,\tau} (b_{k,\tau,m} - d_{k,\tau,m}) \ln(1 - D_{k,\tau}[t']) E_k[t] + \sum_n w_{m,n} Q(H_n^{(2)} = 1|\mathbf{D}[t'], \mathbf{E}[t])\Bigg)$$

Parameter updates. The parameters of the model are updated with project gradient ascent according to the expectations over the final distributions computed in the previous two steps and the observed data. We list the parameter updates below. We note that $sgn(\cdot)$ used below denotes the sign function, ν is the iteration number (or epoch), η is the learning rate, μ is the momentum rate. The learning rate is set to 0.001 initially, and is multiplied by a factor 0.9 when at the previous epoch the reconstruction error $\sum_{t,k,\tau} |D_{k,\tau}[t]E_k[t] - D_{k,\tau}[t']E_k[t]|$ increases, and it is multiplied by a factor 1.1 when the reconstruction error decreases. The momentum rate is progressively increased from 0.5 towards 1.0 asymptotically during training.

$$a_{k,\tau,0} = max(a_{k,\tau,0} + \Delta a_{k,\tau,0}[\nu], 0) \quad \text{, where}$$

$$\Delta a_{k,\tau,0}[\nu] = \mu \cdot \Delta a_{k,\tau,0}[\nu - 1] + \eta \frac{1}{T} \sum_t \Bigg(\ln(D_{k,\tau}[t]) - \ln(D_{k,\tau}[t']) \Bigg) - \eta\lambda_1 \sum_{k'\in\mathcal{N}_k} sgn(a_{k,\tau,0} - a_{k',\tau,0}) - \eta\lambda_2 sgn(a_{k,\tau,0})$$

$$b_{k,\tau,0} = max(b_{k,\tau,0} + \Delta b_{k,\tau,0}[\nu], 0) \quad \text{, where}$$

$$\Delta b_{k,\tau,0}[\nu] = \mu \cdot \Delta b_{k,\tau,0}[\nu - 1] + \eta \frac{1}{T} \sum_{t} \Bigg(\ln(1 - D_{k,\tau}[t]) - \ln(1 - D_{k,\tau}[t']) \Bigg)$$
$$- \eta\lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(b_{k,\tau,0} - b_{k',\tau,0}) - \eta\lambda_2 sgn(b_{k,\tau,0})$$

$$a_{k,\tau,m} = max(a_{k,\tau,m} + \Delta a_{k,\tau,m}[\nu], 0) \quad \text{, where}$$

$$\Delta a_{k,\tau,m}[\nu] = \mu \cdot \Delta a_{k,\tau,m}[\nu - 1] + \eta \frac{1}{T} \sum_{t} \Bigg(\ln(D_{k,\tau}[t]) Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t])$$
$$- \ln(D_{k,\tau}[t']) Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Bigg)$$
$$- \eta\lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(a_{k,\tau,m} - a_{k',\tau,m}) - \eta\lambda_2 sgn(a_{k,\tau,m})$$

$$b_{k,\tau,m} = max(b_{k,\tau,m} + \Delta b_{k,\tau,m}[\nu], 0) \quad \text{, where}$$

$$\Delta b_{k,\tau,m}[\nu] = \mu \cdot \Delta b_{k,\tau,m}[\nu-1] + \eta \frac{1}{T} \sum_{t} \Bigg(\ln(1 - D_{k,\tau}[t]) Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - \ln(1 - D_{k,\tau}[t']) Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Bigg) - \eta \lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(b_{k,\tau,m} - b_{k',\tau,m}) - \eta \lambda_2 sgn(b_{k,\tau,m})$$

$$c_{k,\tau,m} = max(c_{k,\tau,m} + \Delta c_{k,\tau,m}[\nu], 0) \quad \text{, where}$$

$$\Delta c_{k,\tau,m}[\nu] = \mu \cdot \Delta c_{k,\tau,m}[\nu-1] + \eta \frac{1}{T} \sum_{t} \Bigg(\ln(D_{k,\tau}[t])(1 - Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - \ln(D_{k,\tau}[t'])(1 - Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Bigg) - \eta \lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(c_{k,\tau,m} - c_{k',\tau,m}) - \eta \lambda_2 sgn(c_{k,\tau,m})$$

$$d_{k,\tau,m} = max(d_{k,\tau,m} + \Delta d_{k,\tau,m}[\nu], 0) \quad \text{, where}$$

$$\Delta d_{k,\tau,m}[\nu] = \mu \cdot \Delta d_{k,\tau,m}[\nu-1] + \eta \frac{1}{T} \sum_{t} \Bigg(\ln(1 - D_{k,\tau}[t])(1 - Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - \ln(1 - D_{k,\tau}[t'])(1 - Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Bigg) - \eta \lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(d_{k,\tau,m} - d_{k',\tau,m}) - \eta \lambda_2 sgn(d_{k,\tau,m})$$

$$w_{m,0} = w_{m,0} + \Delta w_{m,0}[\nu] \quad \text{, where}$$

$$\begin{aligned}\Delta w_{m,0}[\nu] = \mu \cdot \Delta w_{m,0}[\nu - 1] + \eta \frac{1}{T} \sum_{t} \Big(& Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) \\ & - Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Big) \\ & - \eta \lambda_2 sgn(w_{m,0})\end{aligned}$$

$$w_{n,0} = w_{n,0} + \Delta w_{n,0}[\nu] \quad \text{, where}$$

$$\begin{aligned}\Delta w_{n,0}[\nu] = \mu \cdot \Delta w_{n,0}[\nu - 1] + \eta \frac{1}{T} \sum_{t} \Big(& Q(H_n^{(2)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) \\ & - Q(H_n^{(2)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Big) \\ & - \eta \lambda_2 sgn(w_{n,0})\end{aligned}$$

$$w_{o,0} = w_{o,0} + \Delta w_{o,0}[\nu] \quad \text{, where}$$

$$\Delta w_{o,0}[\nu] = \mu \cdot \Delta w_{o,0}[\nu - 1] + \eta \frac{1}{T} \sum_t \Big(Q(H_o^{(3)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - Q(H_o^{(3)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Big) - \eta \lambda_2 sgn(w_{o,0})$$

$$w_{m,n} = w_{m,n} + \Delta w_{m,n}[\nu] \quad \text{, where}$$

$$\Delta w_{m,n}[\nu] = \mu \cdot \Delta w_{m,n}[\nu - 1] + \eta \frac{1}{T} \sum_t \Big(Q(H_m^{(1)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) Q(H_n^{(2)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - Q(H_m^{(1)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) Q(H_n^{(2)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Big) - \eta \lambda_2 sgn(w_{m,n})$$

$$w_{n,o} = w_{n,o} + \Delta w_{n,o}[\nu] \quad \text{, where}$$

$$\Delta w_{n,o}[\nu] = \mu \cdot \Delta w_{n,o}[\nu - 1] + \eta \frac{1}{T} \sum_t \Big(Q(H_n^{(2)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) Q(H_o^{(3)} = 1 | \mathbf{D}[t], \mathbf{E}[t]) - Q(H_n^{(2)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) Q(H_o^{(3)} = 1 | \mathbf{D}[t'], \mathbf{E}[t]) \Big) - \eta \lambda_2 sgn(w_{n,o})$$

C.3.1 Parameter updates for the structure part of the BSM

To learn the parameters $w_{k,0}$, $w_{k,r}$, $w_{r,0}$ involving the variables $\mathbf{E}$ and $\mathbf{G}$ of the BSM part modeling the shape structure, we similarly perform contrastive divergence with the following steps:

Inference. Given the observed point existences $\mathbf{E}[t]$, we infer the following distribution over the latent variables $\mathbf{G}$ (we note that this is exact inference):

$$Q(G_r = 1|\mathbf{E}[t]) = \sigma(w_{r,0} + \sum_k w_{k,r} E_k[t])$$

Sampling. We sample the binary latent variables $\mathbf{G}$ according to the inferred distribution $Q(G_r = 1|\mathbf{E}[t])$. Let $G_r[t']$ the resulting sampled $0/1$ values. We perform inference for the existence variables as follows:

$$Q(E_k = 1|\mathbf{G}[t']) = \sigma(w_{k,0} + \sum_r w_{k,r} G_r[t'])$$

and repeat for the latent variables:

$$Q(G_r = 1|\mathbf{G}[t']) = \sigma(w_{r,0} + \sum_k w_{k,r} Q(E_k = 1|\mathbf{G}[t']))$$

Parameter updates The parameters of the structure part of the BSM are updated as follows:

$$w_{k,0} = w_{k,0} + \Delta w_{k,0}[\nu] \quad \text{, where}$$

$$\Delta w_{k,0}[\nu] = \mu \cdot \Delta w_{k,0}[\nu-1] + \eta \frac{1}{T} \sum_t \Big(E_k[t] - Q(E_k = 1 | \mathbf{G}[t']) \Big)$$
$$- \eta \lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(w_{k,0} - w_{k',0}) - \eta \lambda_2 sgn(w_{k,0})$$

$$w_{r,0} = w_{r,0} + \Delta w_{r,0}[\nu] \quad \text{, where}$$

$$\Delta w_{r,0}[\nu] = \mu \cdot \Delta w_{r,0}[\nu-1] + \eta \frac{1}{T} \sum_t \Big(Q(G_r = 1 | \mathbf{E}[t]) - Q(G_r = 1 | \mathbf{G}[t']) \Big)$$
$$- \eta \lambda_2 sgn(w_{r,0})$$

$$w_{k,r} = w_{k,r} + \Delta w_{k,r}[\nu] \quad \text{, where}$$

$$\Delta w_{k,r}[\nu] = \mu \cdot \Delta w_{k,r}[\nu-1] + \eta \frac{1}{T} \sum_t \Big(E_k[t] Q(G_r = 1 | \mathbf{E}[t])$$
$$- Q(E_k = 1 | \mathbf{G}[t']) Q(G_r = 1 | \mathbf{G}[t']) \Big)$$
$$- \eta \lambda_1 \sum_{k' \in \mathcal{N}_k} sgn(w_{k,r} - w_{k',r}) - \eta \lambda_2 sgn(w_{k,r})$$

BIBLIOGRAPHY

[1] Allen, Brett, Curless, Brian, and Popović, Zoran. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graphics 22*, 3 (2003).

[2] Anguelov, Dragomir, Srinivasan, Praveen, Koller, Daphne, Thrun, Sebastian, Rodgers, Jim, and Davis, James. SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24*, 3 (2005).

[3] Ankerst, Mihael, Kastenmüller, Gabi, Kriegel, Hans-Peter, and Seidl, Thomas. 3D shape histograms for similarity search and classification in spatial databases. In *Proc. of Inter. Symposium on Advances in Spatial Databases* (1999), pp. 207–226.

[4] Arsalan Soltani, Amir, Huang, Haibin, Wu, Jiajun, Kulkarni, Tejas D., and Tenenbaum, Joshua B. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[5] Asafi, Shmuel, Goren, Avi, and Cohen-Or, Daniel. Weak convex decomposition by lines-of-sight. *Comp. Graph. Forum 32*, 5 (2013).

[6] Au, Oscar Kin-Chung, Tai, Chiew-Lan, Chu, Hung-Kuo, Cohen-Or, Daniel, and Lee, Tong-Yee. Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3 (2008).

[7] Aubry, M., Schlickewei, U., and Cremers, D. The wave kernel signature: A quantum mechanical approach to shape analysis. In *2011 IEEE International Conference on Computer Vision Workshops* (2011).

[8] Averkiou, Melinos, Kim, Vladimir G., Zheng, Youyi, and Mitra, Niloy J. ShapeSynth: Parameterizing Model Collections for Coupled Shape Exploration and Synthesis. *Comp. Graph. Forum 33*, 2 (2014).

[9] Bae, Seok-Hyung, Balakrishnan, Ravin, and Singh, Karan. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proc. ACM UIST* (2008).

[10] Bao, Sid Yingze, Chandraker, Manmohan, Lin, Yuanqing, and Savaresey, Silvio. Dense object reconstruction with semantic priors. In *Proc. CVPR* (2013).

[11] Belongie, S., Malik, J., and Puzicha, J. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 4 (2002).

[12] Bishop, Christopher M. *Pattern recognition and machine learning*. Springer, 2006.

[13] Blanz, Volker, and Vetter, Thomas. A morphable model for the synthesis of 3D faces. In *Proc. SIGGRAPH* (1999).

[14] Bo, Liefeng, Ren, Xiaofeng, and Fox, Dieter. Learning hierarchical sparse features for rgb-(d) object recognition.

[15] Bogo, Federica, Romero, Javier, Loper, Matthew, and Black, Michael J. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2014).

[16] Boscaini, D., Masci, J., Melzi, S., Bronstein, M. M., Castellani, U., and Vandergheynst, P. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *SGP* (2015), SGP '15, pp. 13–23.

[17] Boscaini, Davide, Masci, Jonathan, Rodolà, Emanuele, and Bronstein, Michael M. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS* (2016).

[18] Bromley, Jane, Guyon, I., Lecun, Yann, Sackinger, Eduard, and Shah, R. *Signature verification using a Siamese time delay neural network*. 1993.

[19] Bronstein, Alexander M., Bronstein, Michael M., Guibas, Leonidas J., and Ovsjanikov, Maks. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM TOG 30* (2011).

[20] Chang, Angel X., Funkhouser, Thomas A., Guibas, Leonidas J., Hanrahan, Pat, Huang, Qi-Xing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, Xiao, Jianxiong, Yi, Li, and Yu, Fisher. Shapenet: An information-rich 3D model repository. *CoRR abs/1512.03012* (2015).

[21] Chang, Chih-Chung, and Lin, Chih-Jen. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2* (2011).

[22] Chen, Ding-Yun, Tian, Xiao-Pei, Shen, Yu-Te, and Ouhyoung, Ming. On Visual Similarity Based 3D Model Retrieval. *CGF* (2003).

[23] Chen, Yu, Kim, T.K., and Cipolla, Roberto. Silhouette-based object phenotype recognition using 3d shape priors. In *Proc. CVPR* (2011).

[24] Cimpoi, Mircea, Maji, Subhrajyoti, Kokkinos, Iasonas, Mohamed, Salina, and Vedaldi, Andrea. Describing textures in the wild. In *Proc. CVPR* (2014).

[25] Dame, Amaury, Prisacariu, Victor, Ren, Carl, and Reid, Ian. Dense reconstruction using 3D object shape priors. In *Proc. CVPR* (2013).

[26] DeCarlo, Doug, Finkelstein, Adam, Rusinkiewicz, Szymon, and Santella, Anthony. Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3 (2003).

[27] Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv:1310.1531* (2013).

[28] Eitz, Mathias, Hays, James, and Alexa, Marc. How do humans sketch objects? *ACM Trans. Graph. 31*, 4 (2012).

[29] Eitz, Mathias, Richter, Ronald, Boubekeur, Tamy, Hildebrand, Kristian, and Alexa, Marc. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (2012).

[30] Fan, Haoqiang, Su, Hao, and Guibas, Leonidas J. A point set generation network for 3d object reconstruction from a single image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[31] Fish, Noa, Averkiou, Melinos, van Kaick, Oliver, Sorkine-Hornung, Olga, Cohen-Or, Daniel, and Mitra, Niloy J. Meta-representation of shape families. *ACM Trans. Graph. 33*, 4 (2014).

[32] Frey, Brendan J., and Dueck, Delbert. Clustering by passing messages between data points. *Science 315* (2007).

[33] Fu, Hongbo, Cohen-Or, Daniel, Dror, Gideon, and Sheffer, Alla. Upright Orientation of Man-made Objects. *ACM Trans. Graph. 27*, 3 (2008).

[34] Funkhouser, Thomas, Min, Patrick, Kazhdan, Michael, Chen, Joyce, Halderman, Alex, Dobkin, David, and Jacobs, David. A search engine for 3d models. *ACM Trans. Graph. 22*, 1 (2003).

[35] Gal, Ran, and Cohen-Or, Daniel. Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph. 25*, 1 (2006).

[36] Germer, T., and Schwarz, M. Procedural arrangement of furniture for real-time walkthroughs. *Comp. Graph. Forum 28*, 8 (2009).

[37] Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jagannath. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR* (2014).

[38] Givoni, Inmar E., Chung, Clement, and Frey, Brendan J. Hierarchical affinity propagation. In *UAI* (2011).

[39] Guo, Kan, Zou, Dongqing, and Chen, Xiaowu. 3D mesh labeling via deep convolutional neural networks. *ACM TOG 35* (2015).

[40] Hadsell, Raia, Chopra, Sumit, and LeCun, Yann. Dimensionality reduction by learning an invariant mapping. In *Proc. CVPR* (2006).

[41] Han, Xufeng, Leung, T., Jia, Y., Sukthankar, R., and Berg, A. C. Matchnet: Unifying feature and metric learning for patch-based matching. In *IEEE CVPR* (2015).

[42] Hou, Suyu, and Ramani, Karthik. Sketch-based 3d engineering part class browsing and retrieval. In *Proc. SBIM* (2006).

[43] Hu, Ruizhen, Fan, Lubin, and Liu, Ligang. Co-segmentation of 3d shapes via subspace clustering. *Comp. Graph. Forum 31*, 5 (2012).

[44] Huang, Qi-Xing, Su, Hao, and Guibas, Leonidas. Fine-grained semi-supervised labeling of large shape collections. *ACM Trans. Graph. 32*, 6 (2013).

[45] Huang, Qi-Xing, Su, Hao, and Guibas, Leonidas. Fine-grained semi-supervised labeling of large shape collections. *ACM Trans. Graph. 32*, 6 (2013).

[46] Huang, Qi-Xing, Zhang, Guo-Xin, Gao, Lin, Hu, Shi-Min, Butscher, Adrian, and Guibas, Leonidas. An optimization approach for extracting and encoding consistent maps in a shape collection. *ACM Trans. Graph. 31*, 6 (2012).

[47] Huang, Qixing, and Guibas, Leonidas. Consistent shape maps via semidefinite programming. *Comp. Graph. Forum 32*, 5 (2013).

[48] Huang, Qixing, Koltun, Vladlen, and Guibas, Leonidas. Joint shape segmentation with linear programming. *ACM Trans. Graph. 30*, 6 (2011).

[49] Huang, Qixing, Wang, Fan, and Guibas, Leonidas. Functional map networks for analyzing and exploring large shape collections. *ACM Trans. Graph. 33*, 4 (2014).

[50] Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093* (2014).

[51] Johnson, Andrew E., and Hebert, Martial. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on pattern analysis and machine intelligence 21*, 5 (1999), 433–449.

[52] Kae, Andrew, Sohn, Kihyuk, Lee, Honglak, and Learned-Miller, Erik. Augmenting crfs with boltzmann machine shape priors for image labeling. In *Proc. CVPR* (2013).

[53] Kalogerakis, Evangelos, Averkiou, Melinos, Maji, Subhransu, and Chaudhuri, Siddhartha. 3D shape segmentation with projective convolutional networks. In *Proc. CVPR* (2017).

[54] Kalogerakis, Evangelos, Chaudhuri, Siddhartha, Koller, Daphne, and Koltun, Vladlen. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4 (2012).

[55] Kalogerakis, Evangelos, Hertzmann, Aaron, and Singh, Karan. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 102.

[56] Kalogerakis, Evangelos, Hertzmann, Aaron, and Singh, Karan. Learning 3D Mesh Segmentation and Labeling. *ACM Trans. Graphics 29*, 3 (2010).

[57] Kazhdan, Michael, Funkhouser, Thomas, and Rusinkiewicz, Szymon. Symmetry descriptors and 3D shape matching. In *SGP* (2004).

[58] Kim, Vladimir G., Chaudhuri, Siddhartha, Guibas, Leonidas, and Funkhouser, Thomas. Shape2Pose: Human-Centric Shape Analysis. *Transactions on Graphics (Proc. of SIGGRAPH) 33*, 4 (2014).

[59] Kim, Vladimir G., Li, Wilmot, Mitra, Niloy J., Chaudhuri, Siddhartha, DiVerdi, Stephen, and Funkhouser, Thomas. Learning part-based templates from large collections of 3d shapes. *ACM Trans. Graph. 32*, 4 (2013).

[60] Kim, Vladimir G, Li, Wilmot, Mitra, Niloy J, Chaudhuri, Siddhartha, DiVerdi, Stephen, and Funkhouser, Thomas. Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 70.

[61] Kim, Vladimir G., Li, Wilmot, Mitra, Niloy J., DiVerdi, Stephen, and Funkhouser, Thomas. Exploring collections of 3d models using fuzzy correspondences. *ACM Trans. Graph. 31*, 4 (2012).

[62] Kingma, Diederik P., and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014).

[63] Koller, Daphne, and Friedman, Nir. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[64] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS* (2012).

[65] Laga, Hamid, Mortara, Michela, and Spagnuolo, Michela. Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes. *ACM Trans. Graph. 32*, 5 (2013).

[66] Lai, Kevin, Bo, Liefeng, and Fox, Dieter. Unsupervised feature learning for 3D scene labeling. In *IEEE ICRA* (2014).

[67] Lavoue;, Guillaume. Combination of bag-of-words descriptors for robust partial shape retrieval. *Vis. Comput. 28*, 9 (2012).

[68] Lienhard, S., Specht, M., Neubert, B., Pauly, M., and Müller, P. Thumbnail galleries for procedural models. *Comp. Graph. Forum 33*, 2 (2014).

[69] Lindenmayer, Aristid. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *J. Theoretical biology* (1968).

[70] Lintermann, Bernd, and Deussen, Oliver. Interactive modeling of plants. *IEEE Computer Graphics and Applications* (1999).

[71] Lipp, Markus, Wonka, Peter, and Wimmer, Michael. Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph.. 27*, 3 (2008).

[72] Litman, Roee, Bronstein, Alex, Bronstein, Michael, and Castellani, Umberto. Supervised learning of bag-of-features shape descriptors using sparse coding. *Comput. Graph. Forum 33*, 5 (2014).

[73] Liu, Yi, Zha, Hongbin, and Qin, Hong. Shape topics: A compact representation and new algorithms for 3d partial shape retrieval. In *Proc. CVPR* (2006).

[74] Marlin, Benjamin M. *Missing Data Problems in Machine Learning*. PhD thesis, University of Toronto, 2008.

[75] Masci, Jonathan, Boscaini, Davide, Bronstein, Michael, and Vandergheynst, Pierre. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2015), pp. 37–45.

[76] Maturana, Daniel, and Scherer, Sebastian. 3D convolutional neural networks for landing zone detection from lidar. In *ICRA* (March 2015).

[77] McCrae, James, and Singh, Karan. Sketch-based path design. In *Proc. Graphics Interface* (2009).

[78] Mech, Radomír, and Miller, Gavin. The deco framework for interactive procedural modeling. *J. Computer Graphics Techniques* (2012).

[79] Měch, Radomír, and Prusinkiewicz, Przemyslaw. Visual models of plants interacting with their environment. In *Proc. SIGGRAPH* (1996).

[80] Merrell, Paul, Schkufza, Eric, and Koltun, Vladlen. Computer-generated residential building layouts. In *ACM Trans. Graph.* (2010), vol. 29.

[81] Mitra, Niloy, Wand, Michael, Zhang, Hao (Richard), Cohen-Or, Daniel, Kim, Vladimir, and Huang, Qi-Xing. Structure-aware shape processing. In *SIGGRAPH Asia 2013 Courses* (2013).

[82] Mohamed, A., Dahl, G. E., and Hinton, G. Acoustic modeling using deep belief networks. *Trans. Audio, Speech and Lang. Proc. 20*, 1 (2012).

[83] Monti, Federico, Boscaini, Davide, Masci, Jonathan, Rodola, Emanuele, Svoboda, Jan, and Bronstein., Michael M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR* (2017).

[84] Müller, Pascal, Wonka, Peter, Haegler, Simon, Ulmer, Andreas, and Van Gool, Luc. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (2006).

[85] Nishida, Gen, Garcia-Dorado, Ignacio, G. Aliaga, Daniel, Benes, Bedrich, and Bousseau, Adrien. Interactive sketching of urban procedural models. *ACM Trans. Graph., to appear* (2016).

[86] Novotni, Marcin, and Klein, Reinhard. 3d zernike descriptors for content based shape retrieval. In *Proc. SMI* (2003).

[87] Ohbuchi, Ryutarou, and Furuya, Takahiko. Distance metric learning and feature combination for shape-based 3d model retrieval. In *Proc. 3DOR* (2010).

[88] Osada, Robert, Funkhouser, Thomas, Chazelle, Bernard, and Dobkin, David. Shape distributions. *ACM Trans. Graph. 21*, 4 (2002).

[89] Ovsjanikov, Maks, Li, Wilmot, Guibas, Leonidas, and Mitra, Niloy J. Exploration of continuous variability in collections of 3D shapes. *ACM Trans. Graph. 30*, 4 (2011).

[90] Ovsjanikov, Maks, Li, Wilmot, Guibas, Leonidas, and Mitra, Niloy J. Exploration of continuous variability in collections of 3D shapes. *ACM Trans. Graphics 30*, 4 (2011).

[91] Parish, Yoav IH, and Müller, Pascal. Procedural modeling of cities. In *Proc. SIGGRAPH* (2001).

[92] Prisacariu, Victor Adrian, and Reid, Ian. Shared shape spaces. In *Proc. ICCV* (2011).

[93] Prusinkiewicz, Przemyslaw. Graphical applications of l-systems. In *Proc. Graphics interface* (1986).

[94] Prusinkiewicz, Przemyslaw, James, Mark, and Měch, Radomír. Synthetic topiary. In *Proc. SIGGRAPH* (1994).

[95] Pu, Jiantao, Lou, Kuiyang, and Ramani, Karthik. A 2d sketch-based user interface for 3d cad model retrieval. *Computer-Aided Design and Applications 2*, 6 (2005).

[96] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. Volumetric and multi-view cnns for object classification on 3D data. In *IEEE CVPR* (2016), pp. 5648–5656.

[97] Qi, Charles R., Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proc. CVPR* (2017).

[98] Ranzato, Marc Aurelio, Susskind, Joshua, Mnih, Volodymyr, and Hinton, Geoffrey. On deep generative models with applications to recognition. In *Proc. IEEE CVPR* (2011).

[99] Razavian, Ali S, Azizpour, Hossein, Sullivan, Josephine, and Carlsson, Stefan. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proc. CVPR Workshops* (2014).

[100] Ritchie, Daniel, Mildenhall, Ben, Goodman, Noah D, and Hanrahan, Pat. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Trans. Graph. 34*, 4 (2015).

[101] Rodola, E., Bulo, S., Windheuser, T., Vestner, M., and Cremers, D. Dense non-rigid shape correspondence using random forests. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014).

[102] Rodola, E., Cosmo, L., Litany, O., Bronstein, M. M., Bronstein, A. M., Audebert, N., Hamza, A. Ben, Boulch, A., Castellani, U., Do, M. N., Duong, A.-D., Furuya, T., Gasparetto, A., Hong, Y., Kim, J., Saux, B. Le, Litman, R., Masoumi, M., Minello, G., Nguyen, H.-D., Nguyen, V.-T., Ohbuchi, R., Pham, V.-K., Phan, T. V., Rezaei, M., Torsello, A., Tran, M.-T., Tran, Q.-T., Truong, B., Wan, L., and Zou, C. Deformable Shape Retrieval with Missing Parts. In *Eurographics Workshop on 3D Object Retrieval* (2017).

[103] Roux, Nicolas Le, Heess, Nicolas, Shotton, Jamie, and Winn, John M. Learning a generative model of images by factoring appearance and shape. *Neural Computation 23*, 3 (2011).

[104] Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* (2015).

[105] Salakhutdinov, Ruslan, and Hinton, Geoffrey. An efficient learning procedure for deep boltzmann machines. *Neural Computation 24*, 8 (2012).

[106] Sandhu, Romeil, Dambreville, Samuel, Yezzi, Anthony, and Tannenbaum, Allen. A nonrigid kernel-based framework for 2d-3d pose estimation and 2d image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 33*, 6 (2011).

[107] Saupe, Dietmar, and Vranic, Dejan V. 3d model retrieval with spherical harmonics and moments. In *Symposium on Pattern Recognition* (2001), pp. 392–397.

[108] Savva, M., Yu, F., Su, Hao, Aono, M., Chen, B., Cohen-Or, D., Deng, W., Su, Hang, Bai, S., Bai, X., Fish, N., Han, J., Kalogerakis, E., Learned-Miller, E. G., Li, Y., Liao, M., Maji, S., Tatsuma, A., Wang, Y., Zhang, N., and Zhou, Z. Large-scale 3d shape retrieval from shapenet core55. In *Proc. 3DOR* (2016).

[109] Schneider, Rosália G., and Tuytelaars, Tinne. Sketch classification and classification-driven analysis using fisher vectors. *ACM Trans. Graph. 33*, 6 (2014).

[110] Schneider, Rosália G., and Tuytelaars, Tinne. Sketch classification and classification-driven analysis using fisher vectors. *ACM Trans. Graph. 33*, 6 (2014).

[111] Schwarz, Michael, and Wonka, Peter. Procedural design of exterior lighting for buildings with complex constraints. *ACM Trans. Graph. 33*, 5 (2014).

[112] Shapira, L., Shalom, S., Shamir, A., Cohen-Or, D., and Zhang, H. Contextual part analogies in 3D objects. *Int. J. Comput. Vision 89*, 2-3 (2010).

[113] Sidi, Oana, van Kaick, Oliver, Kleiman, Yanir, Zhang, Hao, and Cohen-Or, Daniel. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Trans. Graphics 30*, 6 (2011).

[114] Simo-Serra, Edgar, Trulls, Eduard, Ferraz, Luis, Kokkinos, Iasonas, Fua, Pascal, and Moreno-Noguer, Francesc. Discriminative learning of deep convolutional feature point descriptors. In *IEEE ICCV* (2015), ICCV '15.

[115] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[116] Sinha, Ayan, Bai, Jing, and Ramani, Karthik. Deep learning 3D shape surfaces using geometry images. In *IEEE ECCV* (2016).

[117] Smelik, Ruben M., Tutenel, Tim, Bidarra, Rafael, and Benes, Bedrich. A survey on procedural modelling for virtual worlds. *Comp. Graph. Forum 33*, 6 (2014).

[118] Socher, Richard, Huval, Brody, Bhat, Bharath, Manning, Christopher D., and Ng, Andrew Y. Convolutional-recursive deep learning for 3D object classification. In *NIPS* (2012), NIPS'12, pp. 656–664.

[119] Song, Shuran, and Xiao, Jianxiong. Deep Sliding Shapes for amodal 3D object detection in RGB-D images. In *IEEE CVPR* (2016).

[120] Sorkine, Olga, and Alexa, Marc. As-rigid-as-possible surface modeling. In *Proc. SGP* (2007).

[121] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *J. Machine Learning Research 15*, 1 (2014).

[122] Stava, Ondrej, Pirk, Sören, Kratt, Julian, Chen, Baoquan, Měch, R, Deussen, Oliver, and Benes, Bedrich. Inverse procedural modelling of trees. *Comp. Graph. Forum 33*, 6 (2014).

[123] Stiny, George, and Gips, James. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress* (1971).

[124] Su, Hang, Maji, Subhransu, Kalogerakis, Evangelos, and Learned-Miller, Erik G. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV* (2015).

[125] Sumner, Robert W., Schmid, Johannes, and Pauly, Mark. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (2007).

[126] Sutton, Charles, and Mccallum, Andrew. Piecewise training of undirected models. In *Proc. UAI* (2005).

[127] Talton, Jerry O., Gibson, Daniel, Yang, Lingfeng, Hanrahan, Pat, and Koltun, Vladlen. Exploratory modeling with collaborative design spaces. *ACM Trans. Graph. 28*, 5 (2009).

[128] Talton, Jerry O., Lou, Yu, Lesser, Steve, Duke, Jared, Měch, Radomír, and Koltun, Vladlen. Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2 (2011).

[129] Tenenbaum, J.B., Silva, V.De, and Langford, J.C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000).

[130] Theologou, Panagiotis, Pratikakis, Ioannis, and Theoharis, Theoharis. A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation. *CVIU* (2015).

[131] Tombari, Federico, Salti, Samuele, and Di Stefano, Luigi. Unique signatures of histograms for local surface description. In *Proc. ECCV* (2010).

[132] van Kaick, Oliver, Zhang, Hao, Hamarneh, Ghassan, and Cohen-Or, Daniel. A survey on shape correspondence. *Comp. Graph. Forum 30*, 6 (2011).

[133] Vanegas, Carlos A, Garcia-Dorado, Ignacio, Aliaga, Daniel G, Benes, Bedrich, and Waddell, Paul. Inverse design of urban procedural models. *ACM Trans. Graph. 31*, 6 (2012).

[134] Wang, Fang, Kang, Le, and Li, Yi. Sketch-based 3d shape retrieval using convolutional neural networks. In *Proc. CVPR* (2015).

[135] Wang, Yunhai, Asafi, Shmulik, van Kaick, Oliver, Zhang, Hao, Cohen-Or, Daniel, and Chen, Baoquan. Active co-analysis of a set of shapes. *ACM Trans. Graph. 31*, 6 (2012).

[136] Wei, Lingyu, Huang, Qixing, Ceylan, Duygu, Vouga, Etienne, and Li, Hao. Dense human body correspondences using convolutional networks. In *IEEE CVPR* (2016).

[137] White, Chris. King kong: the building of 1933 new york city. In *Proc. SIGGRAPH* (2006).

[138] Wu, Jiajun, Zhang, Chengkai, Xue, Tianfan, Freeman, William T, and Tenenbaum, Joshua B. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (2016), pp. 82–90.

[139] Wu, Zhirong, Song, Shuran, Khosla, Aditya, Yu, Fisher, Zhang, Linguang, Tang, Xiaoou, and Xiao, Jianxiong. 3D shapenets: A deep representation for volumetric shapes. In *IEEE CVPR* (2015), pp. 1912–1920.

[140] Wu, Zhirong, Song, Shuran, Khosla, Aditya, Zhang, Linguang, Tang, Xiaoou, and Xiao, Jianxiong. 3d shapenets: A deep representation for volumetric shape modeling. In *Proc. CVPR* (2015).

[141] Xian, Yongqin, Schiele, Bernt, and Akata, Zeynep. Zero-shot learning - the good, the bad and the ugly. *CoRR* (2017).

[142] Xie, Jin, Fang, Yi, Zhu, Fan, and Wong, Edward. Deepshape: Deep learned shape descriptor for 3d shape matching and retrieval. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[143] Xie, Zhige, Xu, Kai, Liu, Ligang, and Xiong, Yueshan. 3d shape segmentation and labeling via extreme learning machine. *Comp. Graph. Forum 33*, 5 (2014).

[144] Xu, Kai, Kim, Vladimir G., Huang, Qixing, Mitra, Niloy, and Kalogerakis, Evangelos. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses* (2016), SA '16, ACM.

[145] Xu, Kai, Zhang, Hao, Cohen-Or, Daniel, and Chen, Baoquan. Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Trans. Graph. 31*, 4 (2012).

[146] Xu, Weiwei, Shi, Zhouxu, Xu, Mingliang, Zhou, Kun, Wang, Jingdong, Zhou, Bin, Wang, Jinrong, and Yuan, Zhenming. Transductive 3d shape segmentation using sparse reconstruction. *Comp. Graph. Forum 33*, 5 (2014).

[147] Yi, Kwang Moo, Trulls, Eduard, Lepetit, Vincent, and Fua, Pascal. LIFT: Learned Invariant Feature Transform. In *IEEE ECCV* (2016).

[148] Yi, Li, Kim, Vladimir G., Ceylan, Duygu, Shen, I-Chao, Yan, Mengyan, Su, Hao, Lu, Cewu, Huang, Qixing, Sheffer, Alla, and Guibas, Leonidas. A scalable active framework for region annotation in 3D shape collections. *SIGGRAPH Asia* (2016).

[149] Yi, Li, Su, Hao, Guo, Xingwen, and Guibas, Leonidas. Synchronized spectral cnn for 3d shape segmentation. In *Proc. CVPR* (2017).

[150] Yosinski, Jason, Clune, Jeff, Bengio, Yoshua, and Lipson, Hod. How transferable are features in deep neural networks? In *Proc. NIPS* (2014).

[151] Yumer, M.E., and Kara, L.B. Co-constrained handles for deformation in shape collections. *ACM Trans. Graph. 32*, 6 (2014).

[152] Yumer, Mehmet Ersin, Asente, Paul, Mech, Radomír, and Kara, Levent Burak. Procedural modeling using autoencoder networks. In *Proc. ACM UIST* (2015).

[153] Zeng, Andy, Song, Shuran, Nießner, Matthias, Fisher, Matthew, Xiao, Jianxiong, and Funkhouser, Thomas. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. *arXiv preprint arXiv:1603.08182* (2016).

[154] Zhang, Eugene, Mischaikow, Konstantin, and Turk, Greg. Feature-based surface parameterization and texture mapping. *ACM TOG 24* (2005).

[155] Zheng, Youyi, Liu, Han, Dorsey, Julie, and Mitra, Niloy J. Smartcanvas: Context-inferred interpretation of sketches for preparatory design studies. *Comp. Graph. Forum, to appear* (2016).