October 2021

3D Shape Understanding and Generation

Matheus Gadelha
University of Massachusetts Amherst

Recommended Citation

Gadelha, Matheus, "3D Shape Understanding and Generation" (2021). *Doctoral Dissertations*. 2318.
https://doi.org/10.7275/24419869 https://scholarworks.umass.edu/dissertations_2/2318

3D SHAPE UNDERSTANDING AND GENERATION

A Dissertation Presented

by

MATHEUS GADELHA

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 2021

College of Information and Computer Sciences

3D SHAPE UNDERSTANDING AND GENERATION

A Dissertation Presented

by

MATHEUS GADELHA

Approved as to style and content by:

Rui Wang, Co-chair

Subhransu Maji, Co-chair

Evangelos Kalogerakis, Member

Duygu Ceylan, Member

James Allan, Chair
College of Information and Computer Sciences

DEDICATION

Para o Persistente...

ACKNOWLEDGMENTS

Being a Ph.D. student was, by far, the best job I have ever had. This happened not only due to being able to work on problems that I found interesting and challenging, but mainly to interacting with great people. There are many people that I am thankful for in that regard.

My collaborators in industry: Tamy Boubekeur, Nathan Carr, Radomir Mech, Giorgio Gori and, specially, my internship manager and committee member, Duygu Ceylan. Thanks Adobe for allowing me to work with all these amazing people and for generously funding part of my research. Thanks UMass Amherst and NSF for funding my research too. I am also thankful to Tal Hassner and Stefano Soatto for making my first interaction with industry research so pleasant at Amazon. Also, to the people at Google: Abhijit Kundu, Andrea Tagliasacchi, Kyle Genova, Tom Funkhouser, Alireza Fathi and Caroline Pantofaru – thank you for navigating a remote collaboration during a weird pandemic year. Hopefully we will actually meet someday!

My amazing lab mates: Tsung-Yu Lin, Aruni RoyChowdhury, Huaizu Jiang, Gopal Sharma, Hang Su, Pia Bideau, SouYoung Jin, Chenyun Wu, Zezhou Cheng, Jong-Chyi Su, Ashish Singh, Li Yang and Archan Ray – thank you for the conversations, meetings, ideas, dinners and companionship during all these years. I feel incredibly lucky to have had the chance to meet you and go through this journey together.

Great professors at UMass: Thanks Erik Learned-Miller for his joyful and insightful advice, collaboration and, for promoting fun and lively paper discussions during lab meetings; thank you Vangelis Kalogerakis for being a great collaborator and for a great 3D Modelling and Simulation class that got me irreversibly hooked on the topic, and; finally, I am forever grateful to my advisors Subhransu Maji and Rui Wang. I will dearly miss our Wednesday morning meetings and will fondly remember them as one of best parts of my Ph.D. Thank you, Rui, for teaching me the importance of attending to every single detail and for diligently helping me polish my early paper submissions – I learned a lot from those sessions and I will always try to match your aesthetic sense. Thank you, Subhransu, for all the ideas, discussions, in-promptu meetings and debugging sessions. Your fearless confidence in tackling hard problems will always be an inspiration. Thank you both for being great advisors!

Even though being a Ph.D. student was a fantastic experience, many things happen outside work that made life quite hard sometimes. In all those crucial moments, friends and family were paramount for keeping me (reasonably) sane. Thanks to Haniel Barbosa and Hugo Melo for sharing all the hardships of being brazilian academics in computer science. Our life is a bit harder than it should be and just a little bit of that is our fault. Thanks to Filipe Nascimento, Ana Terra, Marcelino Góes, Luiz Rogério and José Neto for just being great friends through all these years. Thank you, Clarinha. You will forever be missed.

Finally, I am thankful to my family for all their support. Mom, dad and Luiza: I am sorry that I've put myself in a path that led me so far away and I am sorry we had to go through such hard times while being apart. I love you and I miss you.

To my wife, Amanda, thank you for always being there for me. Your kindness and wit are guiding lights in the darkest hours. Thank you for sharing your life with me. You are my best friend, greatest teacher and most beloved adventure companion. I love you dearly, darling. Thank you!

ABSTRACT

3D SHAPE UNDERSTANDING AND GENERATION

SEPTEMBER 2021

MATHEUS GADELHA

B.Sc., UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

M.Sc., UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Rui Wang and Professor Subhransu Maji

In recent years, Machine Learning techniques have revolutionized solutions to longstanding image-based problems, like image classification, generation, semantic segmentation, object detection and many others. However, if we want to be able to build agents that can successfully interact with the real world, those techniques need to be capable of reasoning about the world as it truly is: a tridimensional space. There are two main challenges while handling 3D information in machine learning models. First, it is not clear what is the best 3D representation. For images, convolutional neural networks (CNNs) operating on raster images yield the best results in virtually all image-based benchmarks. For 3D data, the best combination of model and representation is still an open question. Second, 3D data is not available on the same scale as images – taking pictures is a common procedure in our daily lives, whereas capturing 3D content is an activity usually restricted to specialized professionals. This thesis is focused on addressing both of these issues. Which model and representation should we use for generating and recognizing 3D data? What are efficient ways of learning 3D representations from a few examples? Is it possible to leverage image data to build models capable of reasoning about the world in 3D?

Our research findings show that it is possible to build models that efficiently generate 3D shapes as irregularly structured representations. Those models require significantly less memory while generating higher quality shapes than the ones based on voxels and multi-view representations. We start by developing techniques to generate shapes represented as point clouds. This class of models leads to high quality reconstructions and better unsupervised feature learning. However, since point clouds are not amenable to editing and human manipulation, we also present models capable of generating shapes as sets of shape handles – simpler primitives that summarize complex 3D shapes and were specifically designed for high-level tasks and user interaction. Despite their effectiveness, those approaches require some form of 3D supervision, which is scarce. We present multiple alternatives to this problem. First, we investigate how approximate convex decomposition techniques can be used as self-supervision to improve recognition models when only a limited number of labels are available. Second, we study how neural network architectures induce shape priors that can be used in multiple reconstruction tasks – using both volumetric and manifold representations. In this regime, reconstruction is performed from a single example – either a sparse point cloud or multiple silhouettes. Finally, we demonstrate how to train generative models of 3D shapes without using any 3D supervision by combining differentiable rendering techniques and Generative Adversarial Networks.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

The ability of reasoning in a tridimensional space is of utmost importance for any agent in our physical world. Very early in the evolutionary process, living beings developed mechanisms for sensing the world in 3D. Birds, mammals, reptiles and even insects: they all developed some type of *stereopsis* – the ability to perceive 3D from multiple views. Since evolution showed us that living beings benefit from 3D reasoning, it makes sense that man-made agents should be endowed with similar capabilities. If we want to build machines that can interact with the world around us, grasp objects, avoid obstacles and reason about the space in general, we need to be able build models that allow those machines to analyze and generate 3D data. This thesis is focused on developing techniques that allow us to build those models in a variety of scenarios, with a specific focus on deep learning techniques.

We are mainly concerned about two important issues: *representation* and *data*. Regarding representations, differently from images, it is not clear what is the best way to represent 3D data in deep learning models. The first part of this thesis (Chapters 1, 2 and 3) focuses on irregularly structured representations. We show how we can build models capable of generating 3D data represented as sets of point clouds (Chapters 1 and 2) and shape handles (Chapter 3). Point cloud models have a smaller memory footprint than volumetric and multiview counterparts while reconstructing more accurate shapes; models based on sets of shape handles have a different goal – they are designed be amenable to shape manipulation tasks, like editing and completion. The second part of this thesis concerns dealing with the lack of 3D data as supervision and understanding what is the role that neural network architectures play in inducing shape priors. Chapter 4 describes how to utilize approximate convex decomposition (ACD) as self-supervised learning task to improve discriminative models of point clouds trained with limited amount of labels. We show that features learned by computing ACD yield significant improvement in few-shot segmentation and unsupervised shape classification benchmarks. Chapters 5 and 6 explore the prior induced by neural networks when generating 3D data. Chapter 5 studies the case where networks are used to represent manifolds. We analytically characterize this prior by analyzing the networks' limiting behavior as a Gaussian Process and show that it yields impressive results in surface reconstruction tasks. On the other hand, Chapter 6 is focused on reconstructing shapes using volumetric representations while learning directly from images. We introduce a series of differentiable projection operators and show applications to shape reconstruction from silhouettes, depth images and computational tomography. Chapter 7 builds upon some of these operators to tackle a more challenging problems: learn generative models of 3D data when no 3D or viewpoint information is available. More precisely, we present a class of generative adversarial networks, named PrGANs, that is capable of generating 3D shapes from a collection of *unlabeled* images.

Learning from Irregularly Structured Data

Tridimensional occupancy grids are a natural choice for representing 3D data in deep neural networks. They are a straightforward extension to raster images and convolutional layers can be seamlessly applied to this type of data. Another way to represent 3D data is by simply utilizing multiple 2D images of a 3D object. We refer to this as multi-view representation. This type of representation can also be easily integrated with convolutional layers and even offers the extra advantage of being able to leverage image features pre-trained from massive image datasets.

However, generating 3D shapes poses a more challenging situation. While generating 3D data, we are primarily concerned about generating surfaces, which are

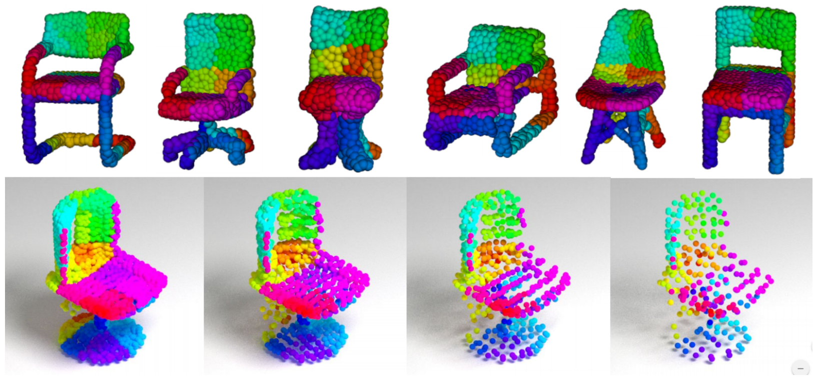


Figure 1: Point clouds sorted according to spatial partitioning structures induce reasonable point correspondence (indicated by similar colors). The same structure can also be used to compute multiple point cloud resolutions. We build upon these observations to design multi-resolution convolutional operators for point cloud data.

inherently sparse in the 3D space. This leads to a big drawback for occupancy grids: models using them require huge amounts of memory, being prohibitively large when generating high-resolution shapes. For multi-view representations, there are two main issues: first, these representations are restricted to representing only the visible portion of the surface – interior parts are not represented. Second, it is not clear how to enforce consistency between different views, which leads to a reduced quality in the generated shapes. Nevertheless, these models are still very memory intensive and the literature has generating multiple categories of objects.

A reasonable alternative to those 3D representations is utilizing point clouds. Point clouds are a very compact surface representation – every point in the point must be part of the surface. They also naturally support extra surface attributes, like color and normals, and are directly captured by a variety of 3D sensors. The biggest challenge while using point clouds in deep networks lies in its unstructured nature. Since they are sets of points, point cloud representations need to be invariant to permutations. Moreover, differently form multi-view representations and occupancy grids, it is not clear what is the best way to use convolutional layers in point cloud data.

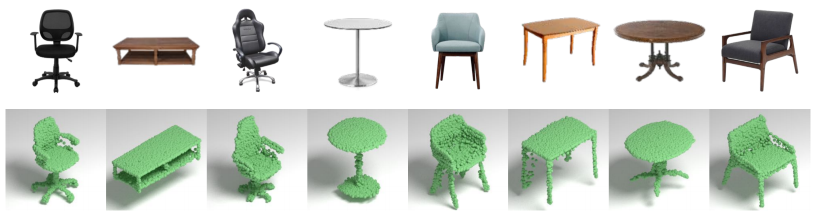


Figure 2: Single-view reconstruction using MRTNets.

Our attempt in creating generative models for point clouds was bootstrapped by using spatial-partitioning data-structures to assign an approximate correspondence between points of different point clouds [51]. The motivation is simple: if one can induce such correspondence, point clouds can be treated as structured data. In practical terms, we compute a *kd*-tree for every point cloud and sort the points according to a level-order traversal in the leaves of the tree. This sorting induces a reasonable correspondence between points, as shown in Figure 1. Using this correspondence, we compute a linear low-dimensional shape representation that were used to train the first Generative Adversarial Networks (GANs) for point cloud generation [51] (**Chapter 1**). Later, we noticed that the spatial partitioning induces a local neighborhood that can be successfully used to define convolutional operations and to represent multiple point cloud resolutions [55] (**Chapter 2**). We called these models Multi-Resolution Tree Networks (MRTNets) and applied them to a variety of discriminative and generative tasks, like shape classification, part segmentation, single-view reconstruction and VAEs. Some of the results are presented in Figure 2. The models have a small memory footprint when compared against multi-view and occupancy grids counterparts while yielding state-of-the-art results for point cloud classification, single-view reconstruction and unsupervised feature learning benchmarks.

However, manipulating shapes represented as point clouds is a complicated task. Suppose one wants to edit the wings of an airplane and make them a bit larger. If the airplane is represented as a set of points that means manually selecting and displacing a big number of elements, which is a very laborious, borderline unfeasible task. To this end, multiple techniques have been developed to summarize 3D shapes a set of simpler proxy shapes that are amenable to manipulations. We refer to those as *shape handles*. Thus, we build upon our previous point cloud work and develop generative models capable of creating shapes represented as sets of shape handles [50] (**Chapter 3**). We then show how those models can be trained an utilized in applications like shape editing, creation, parsing and interpolation.

Learning with Limited 3D Supervision and Shape Priors

Gathering 3D data is a laborious task. Popular 3D shape benchmarks are orders of magnitude smaller than their image counterparts. In this context, building models and training strategies that rely in less training data is specially important. In this thesis, we tackle this problem in multiple manners.

We start by investigating discriminative models for 3D data trained with a limited amount of labels. Simply gathering 3D data is by itself a problem, but labeling such data is equally problematic. While many methods have been developed to improve the efficiency of labeling tasks for 3D shapes, it is still highly desirable to develop label-efficient models that can leverage data from vast unlabelled shape repositories. To this end, we propose to utilize Approximate Convex Decomposition (ACD) as a self-supervised task for training neural networks [54] (**Chapter 4**). We show how point cloud architectures can learn ACD by posing it as a metric learning problem trained using automatically computed labels from raw shape representations (meshes, volumes or point clouds). Our experiments indicate that multiple neural networks architectures trained in this fashion achieve state of the art results in few shot part segmentation and unsupervised shape classification benchmarks.

Another way to tackle the lack of 3D data available is to investigate the role of neural network architectures as priors for shape generation. For images, recent work [188] has showed that convolutional architectures induce natural image priors that allow them to be used for multiple reconstruction tasks without requiring any training data. We investigate an analogous behavior for two types of 3D representations in different contexts. We start by describing how some popular neural network architectures for shape generation can be posed as manifold parametrizations and induce useful priors for manifold reconstruction (**Chapter 5**). We further analyze the limiting behavior of those models as Gaussian Processes (GPs) and analytically characterize such prior by deriving its kernel. We also develop regularization techniques that further improve shape reconstructions in setups with and without training data. In the following chapter, we use convolutional architectures to generate volumetric shape representations coupled with differentiable projection operators to reconstruct shapes from a set of images [56] (**Chapter 6**). We show how volumetric priors induced by those convolutional architectures can be used in applications like shape reconstruction from silhouettes, depth maps, and computational tomography.

Finally, we investigate how to utilize image supervision to train 3D generative models. As mentioned before, image data is considerably more prevalent than 3D. For example, the most used shape classification benchmark, ModelNet40, contains about 10 thousand shapes, whereas the most popular image classification benchmark, ImageNet, contains about 14 million images. Nevertheless, images usually contain real world entities which are inherently 3D. In other words, a lot of 3D information is encoded in images and being able to leverage that information to learn to generate 3D shapes is key to build models that can overcome the lack of 3D training data. We study this issue within a very challenging problem setup. Consider a set of silhouette images like the ones in Figure 3. Those images represent silhouettes of various objects from the same category. If we have viewpoint annotation and object identification (i.e. which images correspond to the same object) this problem can be easily solved using visual hull, which corresponds to the setup analyzed in Chapter 6. We can make the problem harder by assuming that no viewpoint annotation is available. In that case, we can probably achieve a reasonable result by relying in Structure from

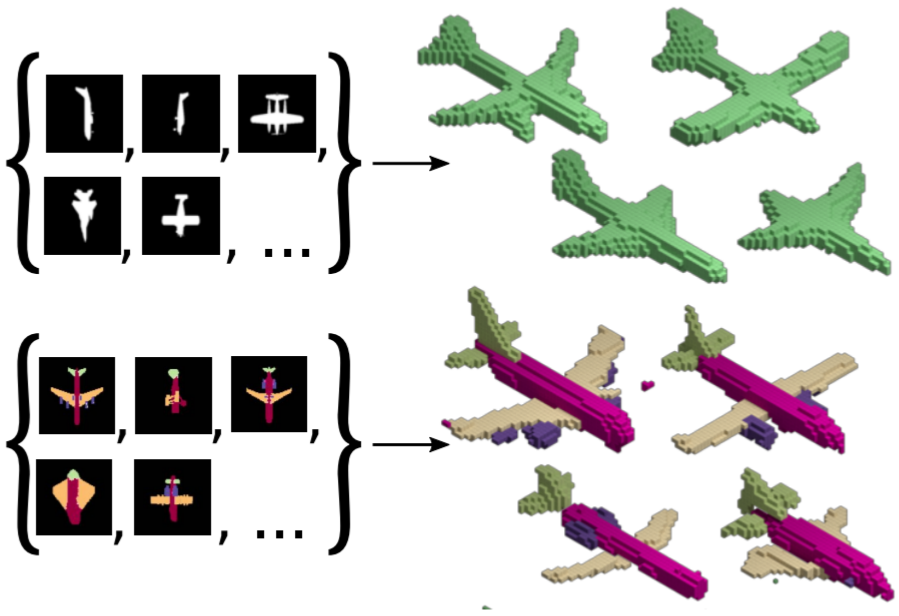


Figure 3: PrGAN is capable of learning generative models of 3D data without using any 3D supervision. The core of the approach is the utilization of differentiable projection operators that turn 3D representation into images of silhouettes and segmentation masks.

Motion (SfM) techniques. The most difficult setup occurs when we neither have object identification nor viewpoint annotation. In this scenario, one can only rely on non-rigid SfM, which require a strong prior over the generated shapes. What happens when we have no information regarding 3D shapes? Can we still do something about it?

Our solution consists of utilizing deep generative models coupled with some of the differentiable projection operators described in Chapter 4. The intuition is simple: given a dataset with images, we want to generate 3D shapes that, when projected into the image plane, will look like they came from the dataset. More precisely, we want to match the distribution of images in the dataset to the images created by projecting the generated 3D shapes. Fortunately, there is a class of deep learning models which is remarkably good in mimicking target image distributions: generative adversarial networks (GANs). Thus, we augment the GAN generator with a differentiable shape projection module which turns 3D shapes into silhouette images. The result is a 3D generative model that is trained without ever seeing any 3D data, only silhouettes of

3D objects (**Chapter 7**). We name this model Projective GAN (PrGAN) [52, 53]. Additionally, we extended the projection operators from Chapter 6 to enable learning from extra image annotations allowing training PrGANs from part-segmented images [53].

CHAPTER 1

SHAPE GENERATION USING SPATIALLY PARTITIONED POINT CLOUDS

The choice of representation is a critical for learning a good generative model of 3D shapes. Voxel-based representations that discretize the geometric occupancy into a fixed resolution 3D grid offers compelling advantages since convolutional operations can be applied. However, they scale poorly with the resolution of the grid and are also wasteful since the geometry of most 3D shapes lies on their surfaces, resulting in a voxel grid that's mostly empty, especially at high resolutions. Surface-based representations such as triangle meshes and point clouds are more efficient for capturing surface geometry, but these representations are inherently unstructured – because there is no natural ordering of the points, they are better expressed as an unordered *set*. Consequently, unlike *ordered* representations, they are cannot be easily generated using existing deep convolutional architectures. The exception is when the points are in perfect correspondence across shapes, in which case a linear shape basis can be effective (*e.g.*, for faces or human bodies). However estimating accurate global correspondences is difficult and even poorly defined for categories such as chairs that have complex and varying geometry. Thus generating 3D shapes as point clouds remains a challenge.

We propose a new method for learning a generative model for 3D shapes represented as point clouds. Figure 1.1 illustrates our network architecture. The key idea is to use a space-partitioning data structure, such as a *kd-tree*, to approximately order the points. Unlike a voxel-grid occupancy representation, the kd-tree representation

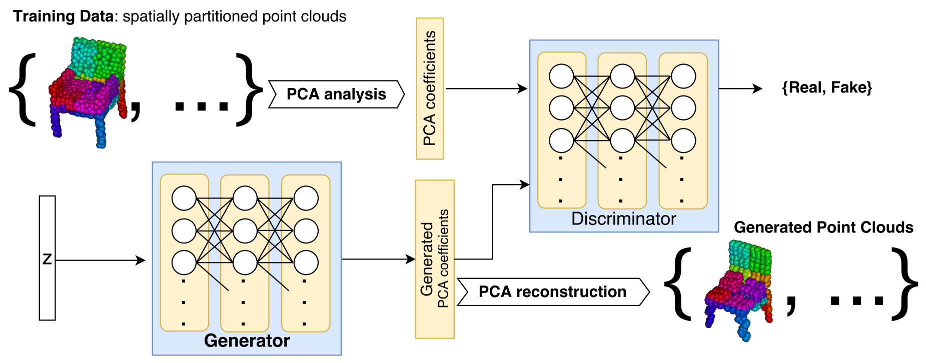


Figure 1.1: Our network architecture for generating 3D shapes using spatially partitioned point clouds. We perform PCA analysis on the training data to drive a shape basis and associated shape coefficients. We then train a GAN to learn the multi-modal distribution over the coefficients. The generated coefficients are combined with the shape basis to produce the output point clouds.

scales linearly with the number of points on the surface and can adapt to the geometry of the model. Moreover one can easily incorporate other point attributes such as surface normal, color, and texture coordinates into this representation, making it possible to generate new shapes that automatically include these information. We learn a shape basis over the ordered point clouds using low-rank factorization of the shape coordinates. If the alignments induced by the kd-tree sorting was perfect, the distribution of the coefficients would be simple. Indeed this is the assumption behind generative models such as Probabilistic PCA [180] that models the distributions of coefficients using independent Gaussians. However, imperfect alignment can lead to a multi-modal and heavy-tailed distribution over the coefficients. To address this issue, we propose to leverage the expressive power of neural networks and employ a Generative Adversarial Network (GAN) [63] to learn the distribution over the shape coefficients. Unlike other non-parametric distributions such as a mixture of Gaussians, the GAN linearizes the distribution of shapes and allows interpolation between them using arithmetic operations. At the same time our method remains light-weight and scalable, since most shape categories can be well represented with a hundred basis coefficients.

We compare the proposed generative model to a 3D-GAN approach of Wu *et al.* [204] that learns a convolutional architecture over a voxel-representation of 3D shapes. In addition we compare to a Probabilistic PCA (PPCA) baseline using the same point-cloud representation. Experiments on several categories in the ShapeNet dataset show that the proposed approach outperforms PPCA and 3D-GAN, quantitatively and qualitatively. Compared to the 3D-GANs our models are an order-of-magnitude faster and smaller. We then present several experiments evaluating the role of the kd-tree on the quality of the generated shapes. We also show that a 1D-convolutional GAN trained on the ordered list of point coordinates produces samples of reasonable quality, suggesting that the kd-tree ordering plays a key role.

1.1 Related Work

Generative models for 3D shapes. Wu *et al.*in [204] proposed a generative model of 3D shapes represented by voxels, using a variant of GAN adapted to 3D convolutions. Two other works are also related. Rezende *et al.* [145] show results for 3D shape completion for simple shapes when views are provided, but require the viewpoints to be known and the generative models are trained on 3D data. Yan *et al.*in [211] learn a mapping from an image to 3D using multiple projections of the 3D shape from known viewpoints (similar to a visual-hull technique). However, these models operate on a voxel representation of 3D shape, which is difficult to scale up to higher resolution. The network also contains a large number of parameters, which are difficult and take a long time to train. Our method uses spatially partitioned point cloud to represent each shape. It is considerably more lightweight and easy to scale up. In addition, by using a linear shape basis, our network is small hence much easier and faster to train. Through experiments we show that the benefits of this lightweight approach come with no loss of quality compared to previous work. Several recent techniques [147, 174] have explored multi-resolution voxel representations such as *octrees* [117] to improve their memory footprint at the expense of additional book keeping. But it remains unclear if 3D-GANs can generate high-resolution sparse outputs.

Learning a 3D shape bases using point-to-point correspondence. Another line of work aims to learn a shape basis from data assuming a global alignment of point clouds across models. Blanz and Vetter in [13] popularized the 3D morphable models for faces which are learned by a PCA analysis of the point clouds across a set of faces with known correspondences. The same idea has also been applied to human bodies [4], and other deformable categories [88]. However, establishing the point-to-point correspondence between 3D shapes is a challenging problem. Techniques are based on global rigid or non-rigid pairwise alignment (*e.g.*, [12, 18, 29]), learning feature descriptors for matching (*e.g.*, techniques in this survey [189]), or fitting a parametric model to each instance (*e.g.*, [23, 139]). Some techniques improve pairwise correspondence by enforcing cycle-consistency across instances [80]. However, none of these techniques provide consistent global correspondences for shapes with varying and complex structures (*e.g.*, chairs and airplanes). Our method uses spatial sorting based on a kd-tree structure. It is a fast and lightweight approximation to the correspondence problem. However, unlike alignment-based approaches, one drawback of the kd-tree sorting is that it's not robust to rotations of the model instances. This is also a drawback of the voxel-based representations. The ShapeNet dataset [24] used in our experiments already contains objects that are consistently oriented, but otherwise one can apply automatic techniques (*e.g.*, [170]) for viewpoint estimation to achieve this.

1.2 Method

This section explains our method. To begin, we sample each training 3D shape using Poisson Disk sampling [15] to generate a consistent number of evenly distributed points per shape. We typically sample each shape with 1K points, and this can

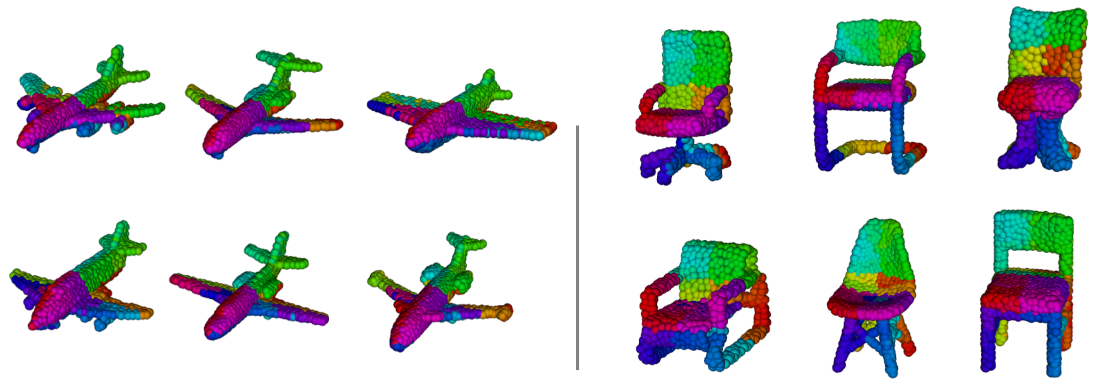


Figure 1.2: Visualization of spatially partitioned points for six training shapes from each category. Every point is colored by its index in the sorted order. This shows that the kd-tree sorting leads to reasonably good correspondences between points across all shapes.

be easily increased or decreased based on actual need. We then build a kd-tree data structure for each point cloud to spatially partition the points and order them consistently across all shapes. Next, we compute the PCA bases using all the point data. Finally, we train a GAN on the shape coefficients to learn the multi-modal distribution of these coefficients and use it to generate new shapes.

Spatially partitioned point cloud. We use $\{P_i^s\}$ to represent a point cloud where i is the point index and s is the shape index. By default the point data P includes the x, y, z coordinates of a point, but can include additional attributes such as normal and color etc. We assume each point cloud is centered at the origin and the bounding box is normalized so that the longest dimension spans [-0.5, 0.5]. For each point cloud we build a kd-tree by the following procedure: we start by sorting the entire point cloud along the x-axis, and split it in half, resulting in a left subset and a right subset; we then recursively split each of the two subsets, but this time along the y-axis; then along z-axis, and so on. Basically it's a recursive splitting process where the splitting axis alternates between x, y, and z. The splitting axes can also be chosen in other ways (such as using the longest dimension at each split) to optimize the kd-tree building, but it needs to be consistent across all point clouds.

The kd-tree building naturally sorts the point cloud spatially, and is consistent across all shapes. For example, if we pick the first point from each sorted point cloud, they all have the same spatial relationship to the rest of the points. As a result, this establishes reasonably good correspondences among the point clouds. Figure 1.2 shows an illustration.

Computing PCA bases. We use PCA analysis to derive a linear shape basis on the spatially partitioned point clouds. To begin, we construct a matrix P that consists of the concatenated x, y, z coordinates of each point cloud and all shapes in a given category. The dimensionality of the matrix is $3\,N \times S$ where N is the number of points in each shape, and S is the number of shapes. We then perform a PCA on the matrix: $P = U\Sigma V$, resulting in a linear shape basis U. Thanks to point sorting using kd-tree, a small basis set is sufficient to well represent all shapes in a category. We use B to represent the size of the shape basis, and by default choose $B = 100$, which has worked well for all ShapeNet categories we experimented with. The choice of B can be observed from the rapid dropping of singular values Σ following the PCA analysis. Without a good spatial sorting method, it would require a significantly larger basis set to accurately represent all shapes.

To include other point attributes, such as normal, we can concatenate these attributes with the x, y, z coordinates. For example, a matrix that consists of both point and normal data would be $6\,N \times S$ in size. We suitable increase the basis size (e.g. by choosing $B = 200$) to accommodate the additional data. The rest of the PCA analysis is performed the same way.

Learning shape coefficients using GAN. Our method employs a GAN to learn the distribution over the shape coefficients. Following the PCA analysis step, the matrix V captures the coefficients for all training shapes, i.e. the projections of each point cloud onto the PCA basis. It provides a compact and yet accurate approximation of the 3D shapes. Therefore our generative model only needs to learn to generate the shape coefficients. Since the dimensionality of the shape basis ($B = 100$) is much smaller (than the number of points on each shape), we can train a GAN to learn the distribution of coeffcients using a simple and lightweight architecture. In our setup, the random encoding z is a 100-D vector. The generator and discriminator are both fully connected neural networks consisting of 4 layers each, with 100 nodes in each layer. Each layer is followed by a batch normalization step. Following the guidelines of previous architectures [204], our discriminator uses a LeakyReLU activation while our generator uses regular ReLU.

The discriminator is trained by minimizing the vanilla GAN loss described as follows:

$$\mathcal{L}_d = \mathbb{E}_{x \sim \mathcal{T}}[\log(D(x))] + \mathbb{E}_{z \sim U}[\log(1 - D(G(z)))]. \tag{1.1}$$

where x represents the shape coefficients, D is the discriminator, G is the generator, U represents an uniform distribution of real numbers in $(-1, 1)$, and $\mathcal{T}$ is the training data. In our experiments, we noticed that using the traditional loss for the generator leads to a highly unstable training where the generated data converges to a single mode (which loses diversity). To overcome this issue, we employ an approach similar to the one proposed in [151]. Specifically, let $f(x)$ be the intermediate activations of the discriminator given an input x. Our generator will try to generate samples that match some statistics of the activations of the real data, namely mean and covariance. Thus, the generator loss is defined as follows:

$$\mathcal{L}_g = \|\mathbb{E}_{x \sim \mathcal{T}}[f(x)] - \mathbb{E}_{z \sim U}[f(G(z))]\|_2^2 + \|cov_{x \sim \mathcal{T}}[f(x)] - cov_{z \sim U}[f(G(z))]\|_2^2 \tag{1.2}$$

where *cov* is the vectorized covariance matrix of the activations. Using this loss results in a much more stable learning procedure. During all our experiments the single mode problem never occurred, even when training the GAN for thousands of epochs. We use the Adam optimizer [93] with a learning rate of 10^{-4} for the discriminator and

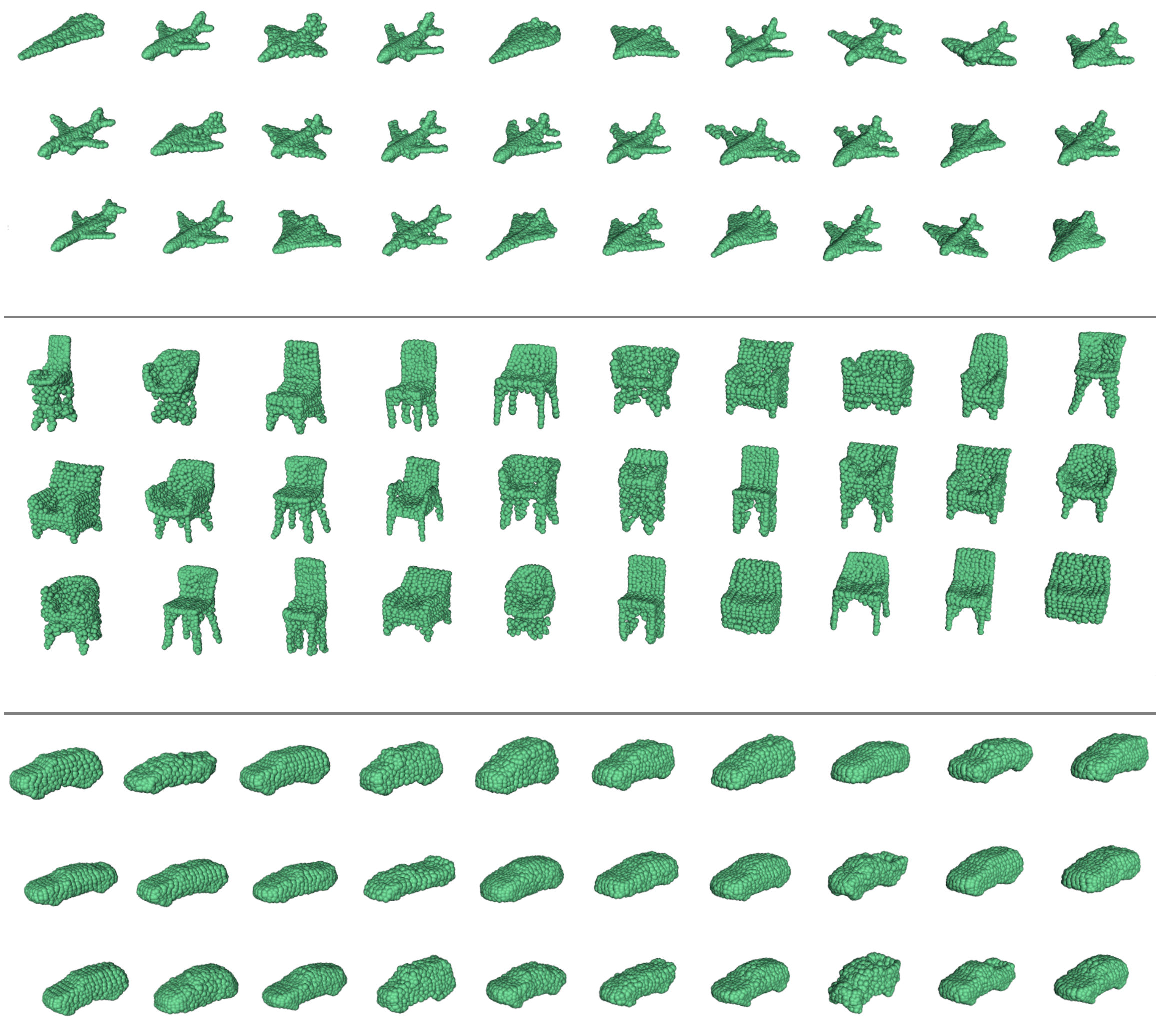


Figure 1.3: A gallery showing results of using our method to generate points clouds for three categories: airplane, chair, and car. We use our method to train a GAN for each category separately. The training is generally very fast and completes within a few minutes. The results shown here are generated by randomly sampling the encoding z of the GAN.

0.0025 for the generator. Similarly to [204], we only train the discriminator if its accuracy is below 80%.

Optimizing point ordering. While sorting using the kd-tree creates good initial correspondences between points, the point ordering can be further optimized by iteratively reducing the PCA reconstruction error through the following procedure. For shape's point cloud $\{P_i^s\}$ (where s is the shape index and i is the point index), we perform random swapping K times. Specifically, we first randomly select a pair of

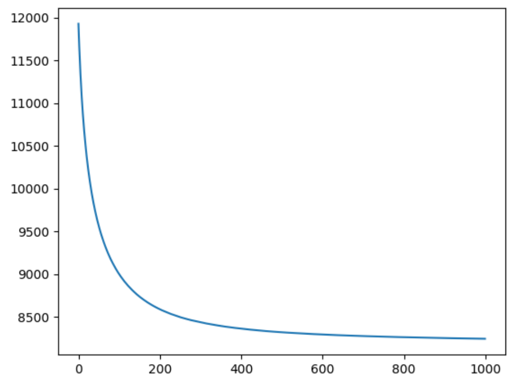


Figure 1.4: Decay of PCA reconstruction error following $I = 1000$ iterations of the point optimiation procedure. The vertical axis represents the PCA reconstruction error and the horizontal axis represents the number of iterations.

points $\langle P_i^s, P_j^s \rangle$ and make them candidates for swapping. If the resutling PCA reconstruction error is reduced, we swap the two points. This is repeated K times. The reconstruction error of a vectorized point cloud P^s using a basis U is computed as follows:

$$\mathcal{L}_{rec}(P^s, U) = \left\| (P^s - \mu)^T U^T U + \mu - P^s \right\|_2^2, \tag{1.3}$$

where $\mu = \frac{1}{|\mathcal{D}|} \sum_{s \in \mathcal{D}} P^s$. After every shape is processed, we then re-compute a new PCA basis using the optimized point ordering. Finally, the whole procedure is repeated I iterations. In our experiments, we have chosen to use $K = 10^4, I = 10^3$. Figure 1.4 shows the decay of reconstruction error during the optimization procedure. The shapes used in this figure are chair models from the ShapeNet dataset. Experiments show that the point optimization improves the results both qualitatively and quantitatively.

1.3 Experiments

Training data. To generate training data, we use several shape categories from the ShapeNet dataset [24], including chairs, airplanes, cars etc. We sample each shape with 1K Poisson disk sample points using the algorithm described in [15]. Poisson

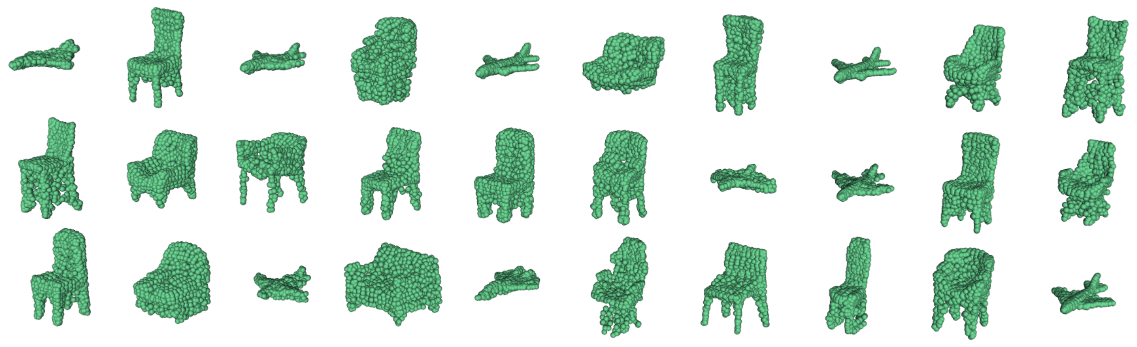


Figure 1.5: Results for a mixed category (chair + airplane) showing the ability of our method to capture multi-modal distributions over mixed-category shapes.

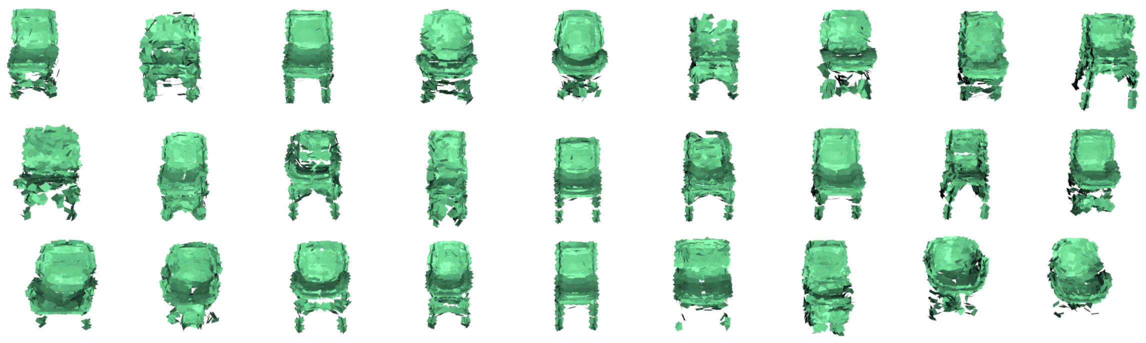


Figure 1.6: Chairs generated with normal. For visualization we shade each point as a square patch centered at the point and oriented by the normal. This shows the ability of our method to generate not only x, y, z coordinates but also incorporate associated point attributes such as normal.

disk samples evenly disperse the points over the surface, which turns out to work better at preserving geometric details than using white noise samples. We can easily increase the number of sample points to 4K or 8K and beyond. Unlike voxel-based representations, our method is lightweight, and increasing the sample size only leads to moderate increases in computation resources and time.

Qualitative evaluation. Figure 1.3 shows a gallery of results generated using our method for each of the three categories: airplane, chair, and car. The results are generated by randomly sampling the encoding z and demonstrate a variety of shapes within each category. The training is very fast and generally completes within a few minutes. This is an order of magnitude faster than training deep neural networks built

Dataset	GAN(10)	GAN(50)	GAN(100)	SGAN (100)	PPCA (100)
Chairs	2.57	2.53	2.37	**2.19**	2.88
Airplanes	1.96	1.93	1.94	**1.48**	2.29
Cars	1.45	1.42	1.44	**1.25**	1.59
Tables	2.88	2.68	2.66	**2.34**	3.18

Table 1.1: Distance (Eq.1.4) between the generated samples and training samples for different generative models. The numbers in parentheses indicate the number of PCA coefficients used for each column. SGAN is the GAN trained using the sorted data. The GAN approach outperforms the PPCA baseline by a considerable margin even without thesorting procedure.

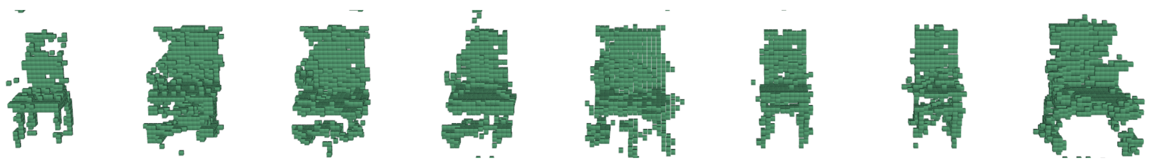


Figure 1.7: 3D-GAN result for the chair category. The models are generated by following [204].

upon voxel representations. Figure 1.5 shows additional results for a mixed category that combines shapes from the chair and airplane datasets. For this mixed category we used $B = 300$ basis. The results show the ability of our method to capture the multi-modal distributions over mixed-category shapes.

Generating multiple point attributes. Our method can generate points with multiple attributes, such as surface normal, color, by simply appending these attributes to the (x, y, z) coordinates. The overall procedure remains the same except the shape basis is learned over the joint space of positions and normals etc. Figure 1.6 shows chair results generated with normal. The ability to incorporate point attributes is an additional advantage over voxel-based representations (which do not explicitly represent surface information of shapes).

Quantitative evaluation. We compare variations of our model to a PPCA baseline [180]. The PPCA model performs a linear factor analysis of the data using: $y \sim \mathbf{W}x + \mu + \sigma$. The matrix $\mathbf{W}$ is a basis, the latent variables $x \sim N(0, I)$, noise

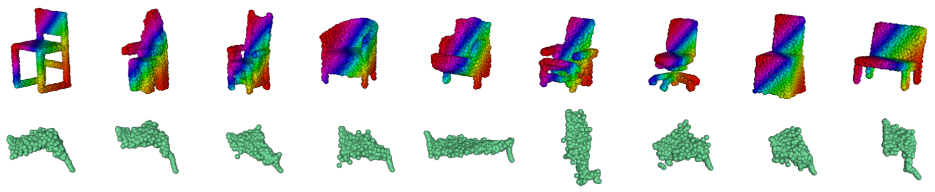


Figure 1.8: Sorting point clouds using $x + y + z$ values. Top row shows a visualization of the training data using this sorting strategy. Bottom row shows the generated shapes for the chair category. They are visually of poor quality compared to kd-tree sorting.

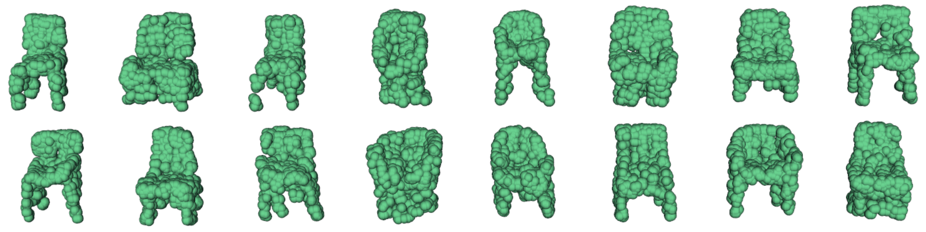


Figure 1.9: Samples from an alternative GAN architecture using 1D convolutions. Trained using the the point clouds directly.

$\sigma \sim N(0, \sigma^2 I)$ and the μ is the data mean. In other words, PPCA learns an independent Gaussian distribution over the coefficients x, whereas our approach employs a GAN. We compare PPCA results with variations of our model by changing the number of basis and examining its influence on the quality of the results. The metric used in the evaluation is defined as follows. Let $\mathcal{T}$ and $\mathcal{S}$ be the set of training and generated samples, respectively. We define our distance measure $d(\mathcal{T}, \mathcal{S})$ using a variant of the Chamfer distance, as follows:

$$d(\mathcal{T}, \mathcal{S}) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \min_{s \in \mathcal{S}} \|t - s\|_2 + \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \min_{t \in \mathcal{T}} \|t - s\|_2 \tag{1.4}$$

The results can be seen in Table 1.1. Our approach that uses a GAN to model the distribution of coefficients consistently outperforms the PPCA baseline, which models the distribution as a Gaussian. For the chairs and tables categories the difference between the PPCA and GAN is large, suggesting that the distribution of the coefficients is highly multi-modal. The results by varying the number of bases are also shown in the Table 1.1. Increasing the number of basis beyond a hundred did not improve our results further.

Visual comparison to 3D-GAN. To compare our results with the 3D-GAN model [204], we followed their description to implement our own version of the algorithm as there is no publicly available implementation that can be trained from scratch. Figure 1.7 shows the 3D-GAN results for the chair category. As in [204], the training data is generated by voxelizing each shape to 64^3 resolution, and we employ the same hyper-parameters for our GAN model as theirs. Our results, which can be found in Figure 1.3, compare favorably to 3D-GAN. In addition, our network is significantly smaller and faster to train.

The role of the *kd-tree*. The kd-tree induces a shape-dependent but consistent ordering of points across shapes. Moreover the ordering is locality preserving, i.e., two points that are close in the underlying 3D shape are also likely to be close in the list after kd-tree ordering. We believe that this property is critical for the estimating a good basis for the shape representation. In order to verify this hypothesis we consider an alternative scheme where the points are ordered according to their $x + y + z$ value. Although consistent across shapes this ordering does not preserve locality of the points and indeed yields poor results as seen in Figure 1.8. However, other data structures that preserve locality such as locality-sensitive hashing [61] and random-projection trees [40] are possible alternatives to kd-trees.

We also experimented an scheme for generating shapes where *1D convolutions* on the ordered points are used for both the generative and discriminative models in a GAN framework. Instead of learning a linear shape basis with has wide support over all the points, the 1D-GAN architecture only has local support. Since the ordering is locality sensitive, one might expect that convolutional filters with small support are sufficient for generation and discrimination. This approach can also be robust to a partial reordering of the list due to variations in the shape structures. Moreover, the 1D-GAN can be directly learned on the ordered point list without having to first learn a bases, and is even more compact than the GAN+PCA basis approach. The architecture used for this experiments has the same number of layers with our standard approach. The major difference is in the fact that we use 1D convolutional layers instead of fully connected ones. The generator layers have a filter size of 25 and the first one has 32 filters. The following layers double the number filters of the previous layer. The discriminator is the mirrored version of the generator. Figure 1.9 shows the results obtained using the 1D-GAN for the chair category. Remarkably, the generated shapes are plausible, but are ultimately of worse quality than our GAN+PCA approach. Both these experiments suggest that the kd-tree plays a important role for our method.

Shape interpolation. Similar to image-based GAN and 3D-GAN, we can perform shape interpolation by linearly interpolating in the encoding space z. Specifically, we can pick two encodings z_1, z_2, linearly interpolate them, and use our generative model to compute the resulting point cloud. The interpolation results are shown in Figure 1.10. As observed, the interpolated shapes are plausible and exhibit non-linearity that cannot be achieved by directly interpolating the shape coefficients.

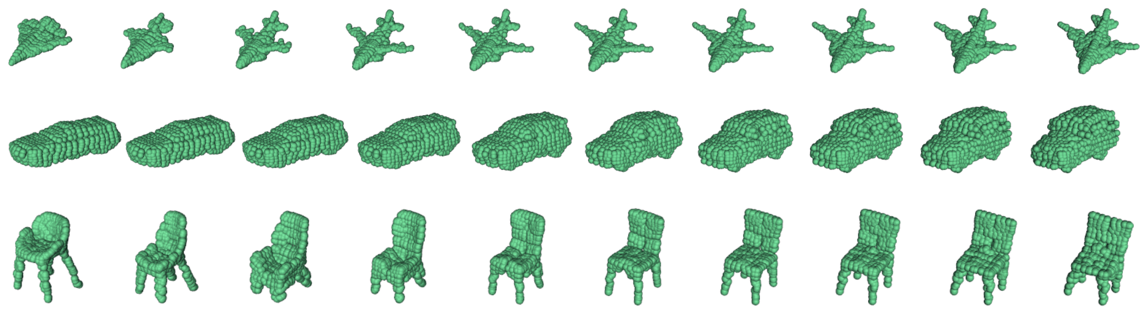


Figure 1.10: Interpolation of the encodings z between a start shape and an end shape for each of the three categories shown here: airplane, car, and chair.

1.4 Conclusion

We showed that conventional CNN architectures can be used to generate 3D shapes as point clouds once they are ordered using kd-trees. We found that a hundred linear basis are generally sufficient to model a category of diverse shapes such as chairs. By employing GANs to model the multi-modal distribution of the basis coefficients we showed that our method outperforms the PPCA baseline approach. The ordering of points produced by the kd-tree also allows reasonable shape generation using 1D-GANs. Our approach is of comparable quality but considerably more lightweight than 3D voxel-based shape generators. Moreover it allows the incorporation of multiple point attributes such normals and color in a seamless manner. In the next chapter, we further investigate the role of space-partitioning data structures on 3D shape classification and segmentation tasks. We also explore incorporating permutation invariant losses in conjunction with multi-grid architectures for unconditional shape generation and reconstruction from single RGB images.

CHAPTER 2

MULTIRESOLUTION TREE NETWORKS FOR 3D POINT CLOUD PROCESSING

One of the challenges in 3D shape processing concerns the question of representation. Shapes are typically represented as triangle meshes or point clouds in computer graphics applications due to their simplicity and light-weight nature. At the same time an increasing number of robotic and remote-sensing applications are deploying sensors that directly collect point-cloud representations of the environment. Hence architectures that efficiently operate on point clouds are becoming increasingly desirable.

On the other hand the vast majority of computer vision techniques rely on grid-based representation of 3D shapes for analyzing and generating them. These include multiview representations that render a shape from a collection of views [142,166,168] or voxel-based representations [17, 79, 116, 204, 206] that discretize point occupancy onto a 3D grid. Such representations allow the use of convolution and pooling operations for efficient processing. However, voxel-representations scale poorly with resolution and are inefficient for modeling surface details. Even multiscale or sparse variants [70, 106, 147] incur relatively high processing cost. Image-based representations, while more efficient, are not effective at modeling shapes with concave or filled interiors due to self occlusions. Moreover, generating shapes as a collection of views requires subsequent reasoning about geometric consistency to infer the 3D shape, which can be challenging.

The main contribution of our work is a multiresolution tree network capable of both recognizing and generating 3D shapes directly as point clouds. An overview of the network and how it can be applied to different applications are shown in Figure 2.1. Our approach represents a 3D shape as a set of locality-preserving 1D ordered list of points at multiple resolution levels. We can obtain such a ordering by using space-partitioning trees such as kd-tree or rp-tree. Feed-forward processing on the underlying tree can be implemented as 1D convolutions and pooling on the list. However, as our experiments show, processing the list alone is not sufficient since the 1D ordering distorts the underlying 3D structure of the shape. To ameliorate this problem we employ a multi-grid network architecture [92] where the representation at a particular resolution influences feed-forward processing at adjacent resolutions. This preserves the locality of point in the underlying 3D shape, improves information flow across scales, enables the network to learn a coarse-to-fine representation, and results in faster convergence during training. Our network outperforms existing point-based networks [96, 141, 169] that operate on position (xyz) information of points. Specifically, it obtains **91.7%** accuracy on the ModelNet40 classification task, while remaining efficient. It also outperforms similar graph networks that do not maintain multiresolution representations.

Our multiresolution decoders can be used for directly generating point clouds. This allows us to incorporate order-invariant loss functions, such as Chamfer distance, over point clouds during training. Moreover it can can be plugged in with existing image-based encoders for image-to-shape inference tasks. Our method is able to both preserve the overall shape structure as well as fine details. On the task of single-image shape inference using the ShapeNet dataset, our approach outperforms existing voxel-based [35], view-based [108], and point-based [49] techniques.

Finally, the combined encoder-decoder network can be used for unsupervised learning of shape representations in a variational autoencoder (VAE) framework. The features extracted from the encoder of our VAE (trained on the unlabeled ShapeNet

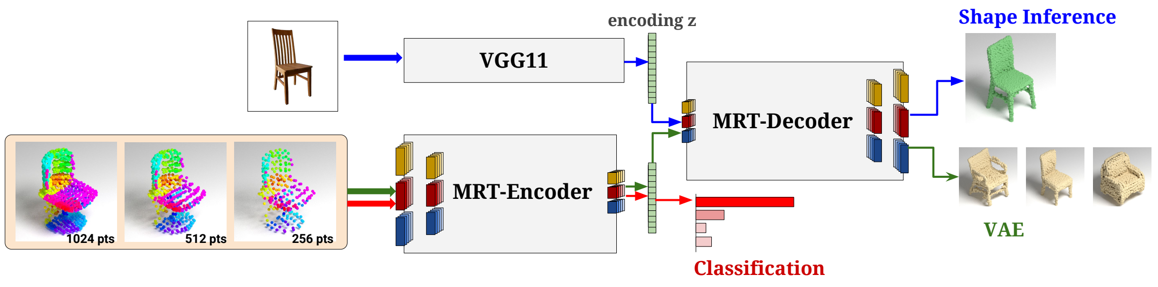


Figure 2.1: Overview of MRTNet. On the left, the MRT-Encoder takes as input a 1D ordered list of points and represents it at multiple resolutions. Points are colored by their indices in the list. On the right, the MRT-Decoder directly outputs a point cloud. Our network can be used for several shape processing tasks, including classification (red), image-to-shape inference (blue), and unsupervised shape learning (green). Refer to Fig. 2.2 for details on the encoder and decoder.

dataset) leads to better shape classification results (**86.4%** accuracy on ModelNet40) compared to those extracted from other unsupervised networks [204].

2.1 Related Work

A number of approaches have studied 3D shape recognition and generation using uniform 3D voxel grids [17, 35, 79, 116, 204, 206]. However, uniform grids have poor scalability and require large memory footprint, hence existing networks built upon them often operate on a relatively low-resolution grid. Several recent works address this issue through multiscale and sparse representations [66,70,147,167,174,194] at the expense of additional book keeping. Still, voxel-based methods generally incur high processing cost, and are not well suited for modeling fine surface details. Moreover, it's not clear how to incorporate certain geometric attributes, like surface normals, into voxel representation, since these attributes do not exist in the interior of the shape.

Multiview methods [86,111,142,166,168] represent a 3D shape as images rendered from different viewpoints. These methods use efficient convolutional and pooling operations and leverage deep networks pretrained on large labeled image datasets. However, they are not optimal for general shapes with complex interior structures due to self occlusions. Nonetheless since most models on existing shape repositories are described well by their exterior surface, view-based approaches have been adapted for shape classification and segmentation tasks. Recently they have also been used for generation where a set of depth and normal maps from different viewpoints are inferred using image-based networks, and have been successfully used for image to shape generation tasks [108, 111]. However such approaches requires subsequent processing to resolve view inconsistencies and outliers which is a challenging task.

Previous work has also studied extensions of ConvNets to mesh surfaces such as spectral CNNs [19, 217], geodesic CNNs [115], or anisotropic CNNs [14]. They have shown success for local correspondence and matching tasks. However, some of these methods are constrained on manifold surfaces, and generally it's unclear how well they perform on shape generation tasks. A recent work in [162] generalized the convolution operator from regular grid to arbitrary graphs while avoiding the spectral domain, allowing graphs of varying size and connectivity.

Another branch of recent works focused on processing shapes represented as point clouds. One example is PointNet [141, 169], that directly consumes point clouds as input. The main idea is to first process each point identically and independently, then leverage a symmetric function (max pooling) to aggregate information from all points. The use of max pooling preserves the permutation invariance of a point set, and the approach is quite effective for shape classification and segmentation tasks. Similarly, KD-net [96] operates directly on point cloud shapes. It spatially partitions a point cloud using a kd-tree, and imposes a feed-forward network on top of the tree structure. This approach is scalable, memory efficient, achieves competitive performance on shape recognition tasks. While successful as encoders, it hasn't been shown how these networks can be employed as decoders for shape generation tasks.

Generating shapes as a collection of points without intermediate modeling of view-based depth maps has been relatively unexplored in the literature. The difficulty stems from the lack of scalable approaches for generating sets. Two recent works are in this direction. Fan et al. [49] train a neural network to generate point clouds from a single image by minimizing Earth Mover's Distance (EMD) or Chamfer distance (CD) between the generated points and the model points. These distances are order invariant and hence can operate directly on point sets. This approach uses a two-branched decoder, one branch is built with 2D transposed convolutions and the other one is composed by fully connected layers. On the other hand, our approach uses a simpler and shallower decoder built as a composition of 1D deconvolutions that operate at multiple scales. This representation improves information flow across scales, which leads to higher quality generated shapes. Moreover, we use permutation invariant losses along with regularization of latent variables to build a model similar to a variational autoencoder [94] that can be used to sample shapes from gaussian noise. Another work in [51] learns a distribution over shape coefficients using a learned basis for a given category using a generative adversarial network [63]. However, in this approach, the underlying generative model assumes a linear shape basis, which produces less detailed surfaces. The improved scalability of our method allows generating shapes with more points and more accurate geometric details in comparison to previous work.

Our tree network builds on the ideas of multiscale [73, 109], mutligrid [92] and dilated [218] or atrous filters [28,46] effective for a number of image recognition tasks. They allow larger receptive fields during convolutions with a modest increase in the number of parameters. In particular Ke et al. [92] showed that communication across multiresolutions of an image throughout the network leads to improved convergence and better accuracy on a variety of tasks. Our approach provides an efficient way of building multigrid-like representations for 3D point clouds.

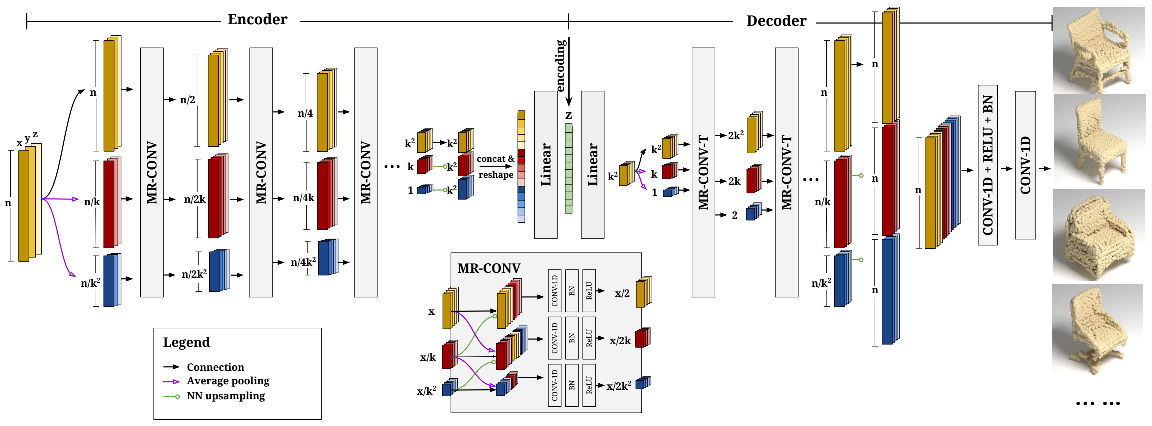


Figure 2.2: Our multiresolution tree network (MRTNet) for processing 3D point clouds. We represent a 3D shape as a 1D list of spatially sorted point cloud. The network represents each layer at three scales (indicated by yellow, red, and blue), the scale ratio is k between each two. The last two convolution layers have kernel size 1 and stride 1. MR-CONV refers to multi-resolution convolution (zoom-in to the inset for details); and MR-CONV-T refers to MR-CONV with transposed convolution. Our network is flexible and can be used for several shape processing tasks.

2.2 Method

Figure 2.2 shows the complete architecture of our multiresolution tree network (MRTNet) that includes both the encoder and decoder. We represent 3D shapes as a point cloud of a fixed size $N = 2^D$ (e.g. $N = 1K$). We center the point cloud at the origin and normalize its bounding box; then spatially sort it using a space-partitioning tree. The input to the network are thus a 1D list ($N \times 3$ tensor) containing the xyz coordinates of the points. The network leverages 1D convolution and represents each layer at three scales, with a ratio of k between each two. MR-CONV refers to multi-resolution convolution, and MR-CONV-T refers to MR-CONV with transposed convolution. The encoding $\mathbf{z}$ is a 512-D vector. Our network architecture is flexible and can be used for several shape processing tasks. For shape **classification**, we use only the multiresolution encoder but adding a fully connected layer after the encoding $\mathbf{z}$ to output a 40-D vector representing the ModelNet40 shape classes. For **single-image shape inference**, we employ a pretrained VGG-11 image encoder [163], combined with our multiresolution decoder to directly an output point cloud shape as a $N \times 3$ tensor. For **unsupervised learning of point clouds**, we use both the multiresolution encoder and decoder, forming a variational autoencoder.

Spatial sorting. As a point cloud is unordered to begin with, we use a space-partitioning tree such as KD-tree to order the points. To start, we sort the entire point set along the x-axis, then split it in half, resulting in equal-sized left and right subsets; we then recursively split each subset, this time along the y-axis; then along z-axis; then back along the x-axis and so on. Basically it's a recursive process to build a full tree where the splitting axes alternate between x, y, z at each level of the tree. The order of leaf nodes naturally becomes the order of the points. There are several variants on the splitting strategy. If at each split we choose an axis among x, y, z with probability proportional to the span of the subset along that axis, it builds a probabilistic KD-tree as described in [96]. If we choose axes from a random set of directions, it builds an RP-tree [40]. Note that after the ordering is obtained, the underlying details of the how the splits were taken are discarded. This is fundamentally different from [96] where the network computations are conditioned on the splitting directions.

The purpose of spatial sorting is to build a hierarchical and locality-preserving order of the points. Thus functions computed based on the local 3D neighborhood at a point can be approximated using convolutions and pooling operations on the 1D structure. However, any ordering of points is distortion inducing and in particular long-range relationships are not preserved well. Maintaining multiple resolutions of the data allows us to preserve locality at different scales. Since the partitioning is constructed hierarchically this can be efficiently implemented using pooling operations described next.

Multiresolution convolution. With the spatially sorted point set, we can build a network using 1D convolution and pooling operations. The convolution leverages the spatial locality of points after sorting, and the pooling leverages the intrinsic binary tree structure of the points.

With a conventional CNN, each convolutional operation has a restricted receptive field and is not able to leverage both global and local information effectively. We resolve this issue by maintaining three different resolutions of the same point cloud using a mutligrid architecture [92]. Different resolutions are computed directly through pooling and upsampling operations. Specifically, we use average pooling with kernel size and stride of k, where k is a power of 2. This configuration allows pooling/downsampling the point cloud while preserving its hierarchical tree structure. Figure 2.1 (left) shows an example point cloud at three different resolutions computed by pooling with $k = 2$. For upsampling, we use nearest neighbor (NN) upsampling with a factor of k.

Once we can pool and upsample the point clouds, we are able to combine global and local information in the convolutional operations by using the MR-CONV block in the inset of Fig. 2.2. The multiresolution block operates in the following way. We maintain the point cloud representations at three resolutions $\mathbf{f}_{(0)}$, $\mathbf{f}_{(1)}$, $\mathbf{f}_{(2)}$, where the scale ratio between each two is (as mentioned above) k. The MR-CONV block receives all three as input, and each resolution will be upsampled and/or pooled and concatenated with each other, creating three new representations $\mathbf{f}'_{(0)}$, $\mathbf{f}'_{(1)}$, $\mathbf{f}'_{(2)}$:

$$\mathbf{f}'_{(0)} = \mathbf{f}_{(0)} \oplus up(\mathbf{f}_{(1)}); \quad \mathbf{f}'_{(1)} = pool(\mathbf{f}_{(0)}) \oplus \mathbf{f}_{(1)} \oplus up(\mathbf{f}_{(2)}); \quad \mathbf{f}'_{(2)} = pool(\mathbf{f}_{(1)}) \oplus \mathbf{f}_{(2)}.$$

where $\oplus$ is the concatenation operation, *up* and *pool* are the upsampling and average pooling operations. Each new representation $\mathbf{f}'$ then goes through a sequence of operations: 1D convolution (kernel size=2 and stride=2), batch normalization and ReLU activation. Note that due to the stride 2, each output is half the size of its associated input. In our generative model and shape inference model we use $k = 4$, while for classification we use $k = 8$.

Shape classification model. For classification, we use our multiresolution encoder in Figure 2.2, and add a fully connected layer after encoding $\mathbf{z}$ that outputs a 40-D vector representing the ModelNet40 classification. Specifically, we train the network on the ModelNet40 [206] dataset, which contains 12,311 objects covering 40 different categories. It is split into 9,843 shapes for training and 2,468 shapes for testing. For each object, we sample 1K points on the surface using Poisson Disk sampling [15] to evenly disperse the points. Each sampled point cloud is then spatially sorted using the probabilistic kd-tree [96]. Specifically, at each split of the tree we choose a random split axis according to the following PDF:

$$P(\mathbf{n} = \mathbf{e}_i | \mathbf{x}) = \frac{\exp\{span_i(\mathbf{x})\}}{\sum_{j=1}^{d} \exp\{span_j(\mathbf{x})\}}$$

where $\mathbf{x}$ is the subset of points to be split, $\mathbf{n}$ is the split axis chosen from the canonical axes $\mathbf{e}_i$ (i.e. x,y,z in 3D), and $span_i(\mathbf{x})$ returns the span of $\mathbf{x}$ along each axis $\mathbf{e}_i$.

The network parameters are as follows: the first MR-CONV layer has 16 filters and the following layers double the amount of filter of the previous one, unless the previous layer has 1024 filters. In that case, the next layer also has 1024 filters. The network is trained by minimizing a cross-entropy loss using an Adam optimizer with learning rate 10^{-3} and $\beta = 0.9$. The learning rate is decayed by dividing it by 2 every 5 training epochs. We employ scale augmentation at training and test time by applying anisotropic scaling factors drawn from $\mathcal{N}(1, 0.25)$. At test time, for each point cloud we apply the sampled scale factors and build the probabilistic kd-tree 16 times as described above, thus obtaining 16 different versions and orderings of the point set. Our final classification is the mean output of those versions. The test-time average has very little impact on the computation time (a discussion is included in Sec. 2.3.4).

Single-image shape inference. Our multiresolution decoder can be used to perform image-to-shape inference. Specifically, we use a pretrained VGG-11 image encoder [163] combined with our decoder in Figure 2.2. Our decoder is set to generates 4K points. The entire network is trained using the dataset and splits provided by [35], which contains 24 renderings from different views for 43783 shapes from ShapeNet divided in 13 different categories. We sample each ShapeNet mesh at 4K points and use them for supervision. Given a rendered image, the task is to predict the complete point cloud (4K points) representing the object in the image. The decoder in Figure 2.2 has the following number of filters per layer: 512-512-256-256-128-64-64-64. As in Figure 2.2, the two additional convolutional layers at the end have kernel size 1 and stride 1: the first one has 128 filters and the second one outputs the final 4K point set.

There are many possible choices for the reconstruction loss function. One straightforward choice would be to use the ordering induced by the spatial partitioning and compute the L_2 loss between the output and ground-truth point clouds. However, L_2 loss turns out to work very poorly. We chose to use the Chamfer distance between two point clouds ($\mathbf{x}$ and $\mathbf{y}$), defined as:

$$Ch(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathbf{x}|} \sum_{x \in \mathbf{x}} \min_{y \in \mathbf{y}} \|x - y\|_2 + \frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} \min_{x \in \mathbf{x}} \|x - y\|_2$$

The Chamfer distance is invariant to permutations of points, making it suitable to measure dissimilarities between unstructured point clouds. The model is trained using an Adam optimizer with learning rate 10^{-3} and $\beta = 0.9$. Learning rate is divided by two at each two epochs.

Unsupervised learning of point clouds. By combining the multiresolution encoder and decoder together, we can perform unsupervised learning of 3D shapes. The entire network, called MR-VAE, builds upon a variational autoencoder (VAE) [94] framework. The encoder Q receives as an input a point cloud $\mathbf{x}$ and outputs an encoding $\mathbf{z} \in \mathbb{R}^{512}$. The decoder D tries to replicate the point cloud $\mathbf{x}$ from $\mathbf{z}$. Both encoder and decoder are built using a sequence of MR-CONV blocks as in Fig. 2.2. Similar to above, we use Chamfer distance as the reconstruction loss function. Besides this, we also need a regularization term that forces the distribution of the encoding $\mathbf{z}$ to be as close as possible to the Gaussian $\mathcal{N}(0, I)$. Differently from the original VAE model, we found that we can get more stable training if we try to match the first two moments (mean and variance) of $\mathbf{z}$ to $\mathcal{N}(0, I)$. Mathematically, this regularization term is defined as:

$$\mathcal{L}_{reg} = \|cov(Q(\mathbf{x}) + \delta) - I\|_2 + E[Q(\mathbf{x}) + \delta]$$

where *cov* is the covariance matrix, Q is the encoder, $\|\cdot\|_2$ is the Frobenius norm and $E[\cdot]$ is the expected value. δ is a random value sampled from $\mathcal{N}(0, cI)$ and $c = 0.01$. Thus, our generative model is trained by minimizing the following loss function:

$$\mathcal{L} = Ch(\mathbf{x}, D(Q(\mathbf{x}))) + \lambda L_{reg}$$

We set $\lambda = 0.1$. The model is trained using an Adam optimizer with learning rate 10^{-4} and $\beta = 0.9$. The encoder follows the classification model and the decoder follows the one used in the shape inference model, both described previously.

Shape part segmentation. MRTNet can also be applied for shape part segmentation tasks. For details please refer to the supplemental material.

2.3 Experimental Results and Discussions

This section presents experimental results. We implemented MRTNet using PyTorch.

Method	Accuracy
View-based methods	
MVCNN [168]	90.1
MVCNN-MultiRes [142]	91.4
Point-based methods (w/o normals)	
KDNet (1K pts) [96]	90.6
PointNet (1K pts) [169]	89.2
PointNet++ (1K pts) [141]	90.7
MRTNet (1K pts)	**91.2**
MRTNet (4K pts)	**91.7**
KDNet (32K pts) [96]	**91.8**
Point-based methods (with normals)	
PointNet++ (5K pts) [141]	91.9
Voxel-based methods	
OctNet [147]	86.5
O-CNN [194]	90.6

Table 2.1: Comparisons with classification models. Among point-based methods that use xyz data only, ours is the best in the 1K points group; and our 4K result is comparable with KDNet at 32K points.

2.3.1 Shape classification

To demonstrate the effectiveness of the multiresolution encoder, we trained a baseline model that follows the same classification model but replacing multiresolution convolutions with single-scale 1D convolutions. Also, we apply the same test-time data augmentation and compute the test-time average as described in the Section 2.2.

Classification benchmark results are in Table 2.1. As shown in the table, MRTNet achieves the best results among all **point-based** methods that use xyz data only. In particular, ours is the best in the 1K points group. We also experimented with sampling shapes using 4K points, and the result is comparable with KDNet at 32K points – in this case, KDNet uses $8\times$ more points (hence $8\times$ more memory) than ours, and is only 0.1% better. PointNet++ [141] with 5K points and normals is 0.2% better than ours.

Method	Accuracy
Full model (MRTNet, 4K pts)	91.7
Filters/4	91.7
Single res.	89.3
Single res., no aug. (rp-tree)	87.4
Single res., no aug. (kd-tree)	86.2

Table 2.2: MRTNet ablation studies on shape classification. Filters/4 reduces the number of filters in each layer by 4. The last three rows are the single resolution model.

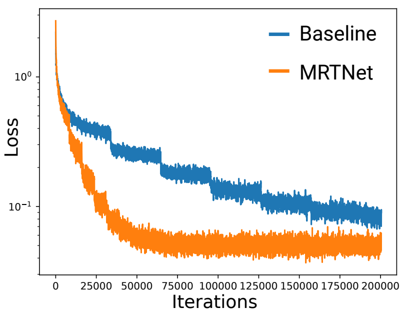


Table 2.2 shows ablation study results with variants of our approach. Particularly, the multiresolution version is more than 2% better than the baseline model (i.e. single resolution), while using the same number of parameters (the Filters/4 version). Besides, MRTNet converges must faster than the baseline model, as we can see in the cross entropy loss decay plots to the right. This shows that the multiresolution architecture leads to higher quality/accuracy and is memory efficient.

Our single resolution baseline is akin to KDNet except it doesn't condition the convolutions on the splitting axes. It results in 1.3% less classification accuracy compared to KDNet (1K pts). This suggests that conditioning on the splitting axes during convolutions improves the accuracy. However, this comes at the cost of extra book keeping and at least three times more parameters. MRTNet achieves greater benefits with lesser overhead. Similar to the KDNet, our methods also benefit from data augmentation and can be used with both kd-trees and rp-trees.

2.3.2 Single-image shape inference

We compare our single-image shape inference results with volumetric [35], view-based [108] and point-based [49] approaches using the evaluation metric by [108]. Given a source point cloud $\mathbf{x}$ and a target point cloud $\mathbf{y}$, we compute the average eu-

Category	3D-R2N2 [35]			Fan et al. [49]	Lin et al. [108]	MRTNet
	1 view	3 views	5 views	(1 view)	(1 view)	(1 view)
airplane	3.207 / 2.879	2.521 / 2.468	2.399 / 2.391	1.301 / 1.488	1.294 / 1.541	**0.976** / **0.920**
bench	3.350 / 3.697	2.465 / 2.746	2.323 / 2.603	1.814 / 1.983	1.757 / 1.487	**1.438** / **1.326**
cabinet	1.636 / 2.817	1.445 / 2.626	**1.420** / 2.619	2.463 / 2.444	1.814 / **1.072**	1.774 / 1.602
car	1.808 / 3.238	1.685 / 3.151	1.664 / 3.146	1.800 / 2.053	1.446 / **1.061**	**1.395** / 1.303
chair	2.759 / 4.207	1.960 / 3.238	1.854 / 3.080	1.887 / 2.355	1.886 / 2.041	**1.650** / **1.603**
display	3.235 / 4.283	2.262 / 3.151	2.088 / 2.953	1.919 / 2.334	2.142 / **1.440**	**1.815** / 1.901
lamp	8.400 / 9.722	6.001 / 7.755	5.698 / 7.331	2.347 / 2.212	2.635 / 4.459	**1.944** / **2.089**
speaker	2.652 / 4.335	2.577 / 4.302	2.487 / 4.203	3.215 / 2.788	2.371 / **1.706**	**2.165** / 2.121
rifle	4.798 / 2.996	4.307 / 2.546	4.193 / 2.447	1.316 / 1.358	1.289 / 1.510	**1.029** / **1.028**
sofa	2.725 / 3.628	2.371 / 3.252	2.306 / 3.196	2.592 / 2.784	1.917 / **1.423**	**1.768** / 1.756
table	3.118 / 4.208	2.268 / 3.277	2.128 / 3.134	1.874 / 2.229	1.689 / 1.620	**1.570** / **1.405**
telephone	2.202 / 3.314	1.969 / 2.834	1.874 / 2.734	1.516 / 1.989	1.939 / **1.198**	**1.346** / 1.332
watercraft	3.592 / 4.007	3.299 / 3.698	3.210 / 3.614	1.715 / 1.877	1.813 / 1.550	**1.394** / **1.490**
mean	3.345 / 4.102	2.702 / 3.465	2.588 / 3.342	1.982 / 2.146	1.846 / 1.701	**1.559** / **1.529**

Table 2.3: Single-image shape inference results. The training data consists of 13 categories of shapes provided by [35]. The numbers shown are [pred→GT / GT→pred] errors, scaled by 100. The mean is computed across all 13 categories. Our MRTNet produces 4K points for each shape.

Fully Connected	Single Res.	MRTNet
1.824 / 2.297	1.708 / 1.831	**1.559** / **1.529**

Table 2.4: Ablation studies for the image to shape decoder. The numbers shown are [pred→GT / GT→pred] errors, scaled by 100. The values are the mean computed across all 13 categories.

clidean distance from each point in $\mathbf{x}$ to its closest in $\mathbf{y}$. We refer to this as pred→GT (prediction to groundtruth) error. It indicates how dissimilar the predicted shape is from the ground-truth. The GT→pred error is computed similarly by swapping $\mathbf{x}$ and $\mathbf{y}$, and it measures coverage (i.e. how complete the ground-truth surface was covered by the prediction). For the voxel based model [35], we used the same procedure as [108], where point clouds are formed by creating one point in the center of each surface voxel. Surface voxels are extracted by subtracting the prediction by its eroded version and rescale them such that the tightest 3D bounding boxes of the prediction and the ground-truth CAD models have the same volume.

Table 2.3 shows our results. Our solution outperforms competing methods in 12 out of 13 categories on the pred→GT error, and in 6 categories on GT→pred error. Note that we are consistently better than the point-based methods such as [49] in both metrics; and we are consistently better than [108] in the pred→GT metric. Furthermore, our method wins by a considerable margin in terms of the mean per category

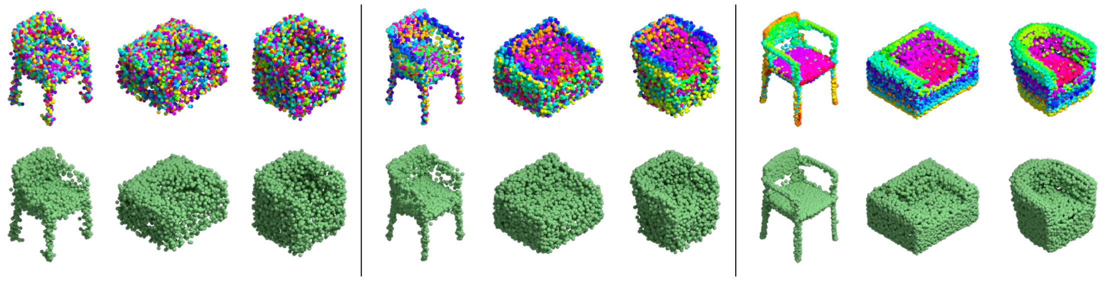


Figure 2.3: Shapes generated by 1) the fully connected baseline; 2) the single-resolution baseline; and 3) MRTNet. Colors in the first row indicate the index of a point in the output point list.

on both metrics. It is important to highlight that the multi-view based method [108] produces tens of thousands of points and many of them are not in the right positions, which penalizes their pred→GT metric, but that helps to improve their GT→pred. Moreover, as mentioned in [108], their method has difficulties capturing thin structures (e.g. lamps) whereas ours is able to capture them relatively well. For example, our GT→pred error for the **lamp** category (which contains many thin geometric structures) is more than two times smaller than the error by [108], indicating that MRTNet is more successful at capturing thin structures in the shapes.

Ablation studies. In order to quantify the effectiveness of the multiresolution decoder, we compared our method with two different baselines: a fully connected decoder and a single-resolution decoder. The fully connected decoder consists of 3 linear layers with 4096 hidden neurons, each layer followed by batch normalization and ReLU activation units. On top of that, we add a final layer that outputs 4096×3 values corresponding to the final point cloud, followed by a hyperbolic tangent activation function. The single resolution decoder follows the same architecture of the MRT decoder but replacing multiresolution convolutions with single-scale 1D convolutions. Results are shown in Table 2.4. Note that both baselines are quite competitive. The single-resolution decoder is comparable to the result of [108], while the fully connected one achieves similar mean errors to [49]. Still, they fall noticeably behind MRTNet.

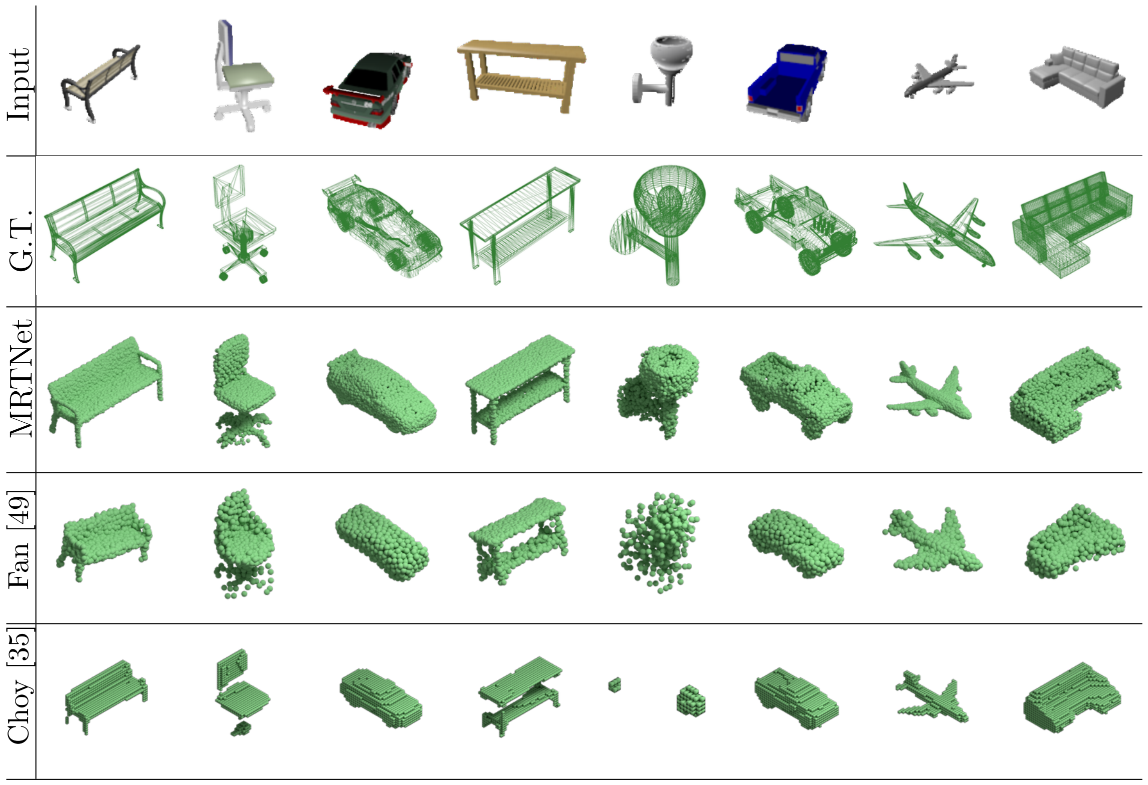


Figure 2.4: Qualitative results for single-image shape inference. From top to bottom: input images, ground truth 3D shapes, results of MRTNet, Fan et al. [49], and Choy et al. [35].

In Figure 2.3 we visualize the structures of the output point clouds generated by the three methods. The point clouds generated by MRTNet present strong spatial coherence: points that are spatially nearby in 3D are also likely to be nearby in the 1D list. This coherence is present to some degree in the single-resolution outputs (note the dark blue points in the chair's arms), but is almost completely absent in the results by the fully connected decoder. This is expected, since fully connected layers do not leverage the spatial correlation of their inputs. Operating at multiple scales enables MRTNet to enforce a stronger spatial coherence, allowing it to more efficiently synthesize detailed point clouds with coherent geometric structures.

Qualitative results. In Figure 2.4 we present qualitative results of our method and comparisons to two prior works. The input images have 3 color channels and dimensions 224×224. In Figure 2.5 we show results of our method applied on

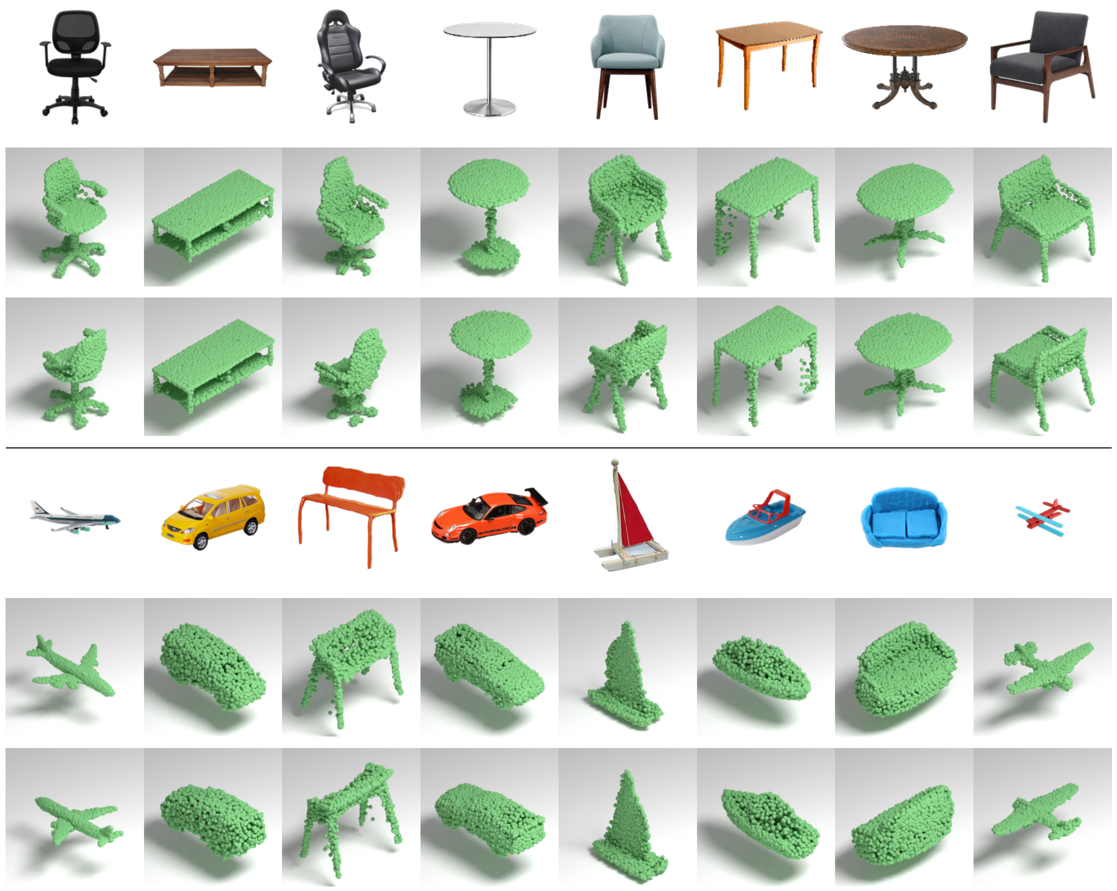


Figure 2.5: Shapes generated by applying MRTNet on Inernet photos of furnitures and toys. MRTNet is trained on the 13 categories of ShapeNet database (Table 2.3) . Note how the network is capable of generating detailed shapes from real photos, even though it is trained only on rendered images using simple shading models. For each output shape we show two different views.

photographs downloaded from the Internet. To apply our method, we manually removed the background from the photos using [1], which generally took less than half a minute per photo. As seen from the results, MRTNet is able to capture the structure and interesting geometric details of the objects (e.g. wheels of the office chairs), even though the input images are considerably different from the rendered ones used in training.

2.3.3 Unsupervised Learning of Point Clouds

For unsupervised learning of point clouds, we train our MR-VAE using the ShapeNet dataset [24]. By default we compute $N = 4K$ points for each shape using Poisson

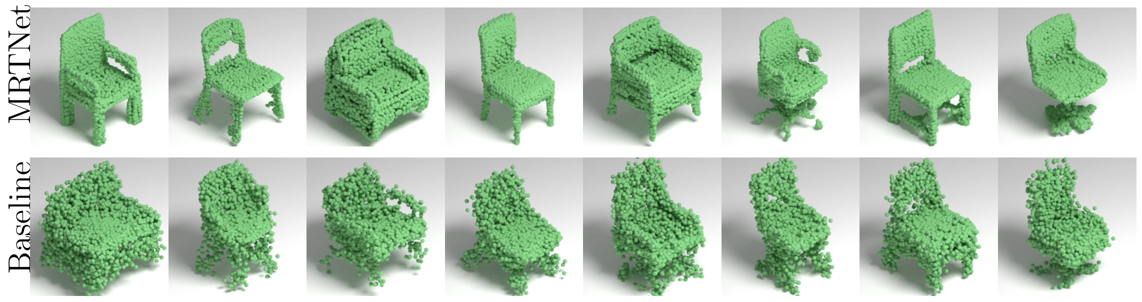


Figure 2.6: Qualitative comparisons of MR-VAE with a single-resolution baseline model. Results are generated by randomly sampling the encoding $\mathbf{z}$. MR-VAE is able to preserve shape details much better than the baseline model, and produces less noisy outputs.

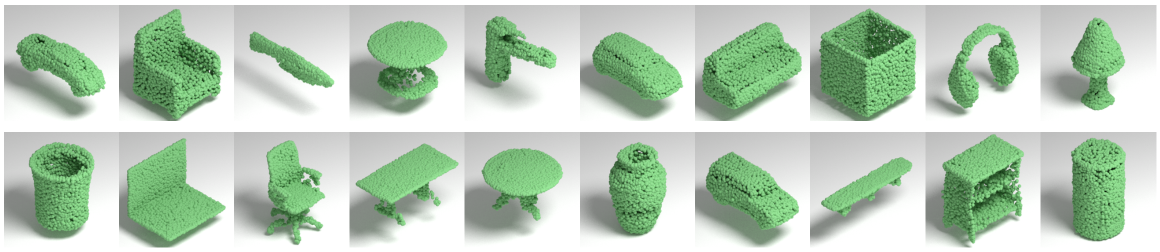


Figure 2.7: Test set shapes reconstructed by MR-VAE trained on all categories of ShapeNet (using 80%/20% training/test split). MR-VAE is able to reconstruct high-quality diverse shapes.

Disk sampling [15] to evenly disperse the points. Each point set is then spatially sorted using a kd-tree. Here we use the vanilla kd-tree where the splitting axes alternate between x, y, z at each level of the tree. The spatially sorted points are used as input to train the MR-VAE network (Section 2.2). Similar to before, we also train a baseline model that follows the same network but replacing multiresolution convolutions with single-scale 1D convolutions in both encoder and decoder. As Figure 2.6 shows, the shapes generated by the MR-VAE trained on chairs are of considerably higher quality than those generated by the baseline model.

We also performed multiple-category shape generation by training MR-VAE on 80% of the objects from ShapeNet dataset. The remaining models belong to our test split. Reconstructions of objects in the test split are included in Figure 2.7. Even when trained with a greater variety of shapes, the MR-VAE is able to reconstruct

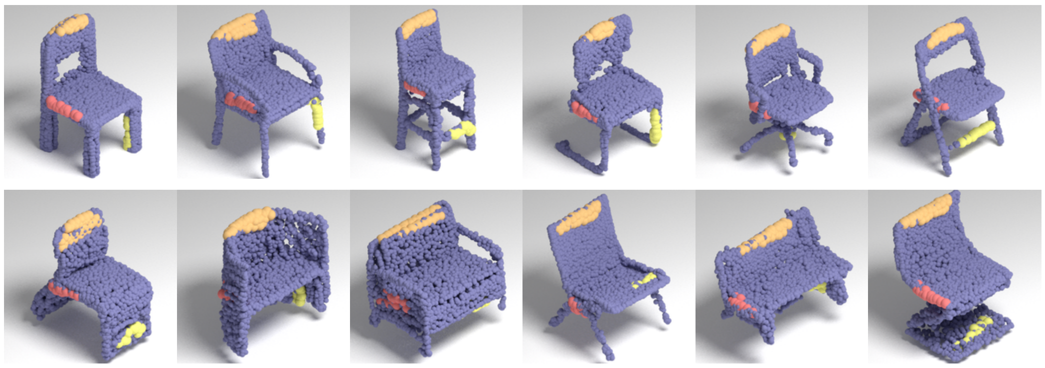


Figure 2.8: Point correspondences among different shapes generated by MR-VAE. We picked three index ranges (indicated by three colors) from one example chair, and then color coded points in every shape that fall into these three ranges. The images clearly show that the network learned to generate shapes with consistent point ordering.

high quality shapes from its embedding. This demonstrates that MR-VAE is suitable for various inference tasks such as shape completion or point cloud reconstructions.

Point ordering in the generated shapes. A useful way to analyze shape generation is to see if the generated points have any consistent ordering across different shapes. This is an interesting question because as described previously, our MR-VAE is trained using Chamfer Distance, a metric that's invariant to permutations of points. While the input to the network is all spatially sorted, the output is not restricted to any particular order and can in theory assume any arbitrary order. In practice, similar to the image-to-shape model, we observe that there is a consistent ordering of points in the generated shapes, as shown in Figure C.4. Specifically, we picked three index ranges from one example chair, one at the top, one on the side, and one close to the bottom, then we color coded points in each shape that fall into these three index ranges. In the figure we can see clearly that they fall into approximately corresponding regions on each chair shape.

Shape interpolation. Another common test is shape interpolation: pick two encodings (either randomly sampled, or generated by the encoder for two input shapes),

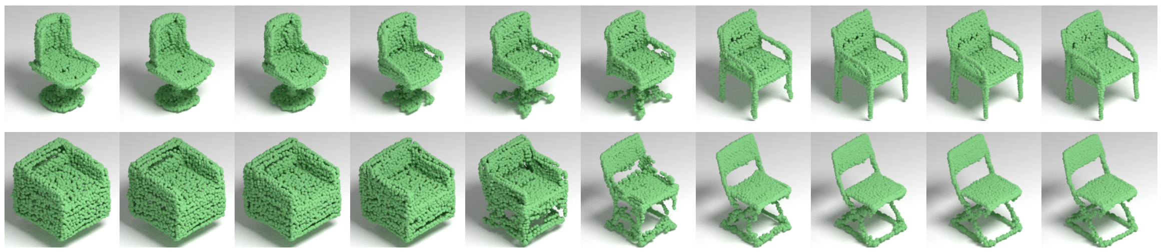


Figure 2.9: Shape interpolation results. For each example, we obtain the encodings $\mathbf{z}$ of the starting shape and ending shape, then linearly interpolate the encodings and use the decoder to generate output shapes from the interpolated $\mathbf{z}$. Results show plausible interpolated shapes.

Method	Accuracy
SPH [90]	68.2
LFD [27]	75.5
T-L Network [62]	74.4
VConv-DAE [157]	75.5
3D-GAN [204]	83.3
MRTNet-VAE (Ours)	**86.4**

Table 2.5: Unsupervised representation learning. The MR-VAE model is trained with all ShapeNet objects, and its features are used to classify ModelNet40 [206] objects. This protocol is the same used by the other competing methods. Our classifier is a single linear layer, where the input is a set of features gathered from the first three layers of the MR-VAE encoder.

linearly interpolate them and use the decoder to generate the output shape. Figure 2.9 shows two sets of interpolation results of chairs from the ShapeNet dataset.

Unsupervised classification. A typical way of assessing the quality of representations learned in a unsupervised setting is to use them as features for classification. To do so, we take the MR-VAE model trained with all ShapeNet objects, and use its features to classify ModelNet40 [206] objects. Our classifier is a single linear layer, where the input is a set of features gathered from the first three layers of the MR-VAE encoder. The features are constructed this way: we apply a pooling operation of size 128, 64 and 32 respectively on these three layers; then at each layer upsample the two smaller resolutions of features to the higher resolution such that all three resolutions have the same size. Finally, we concatenate all those features and pass them through a linear layer to get the final classification. It is important to notice that we did not perform any fine-tuning: the only learned parameters are those from the single linear layer. We used an Adam optimizer with learning rate 10^{-3} and $\beta = 0.9$. The learning rate is decayed by dividing it by 2 every 5 epochs. Using this approach, we obtained an accuracy of 86.34% on the ModelNet40 classification benchmark, as shown in Table 2.5. This result is considerably higher compared to similar features extracted from unsupervised learning in other autoencoders. This shows that the representations learned by our MR-VAE is more effective at capturing and linearizing the latent space of training shapes. Visualizations of the latent representation learned by MR-VAE can be found in the supplemental material for this chapter.

2.3.4 Discussions

Robustness to transformations. Kd-trees are naturally invariant to point jittering as long as it's small enough so as to not alter the shape topology. Our approach is invariant to translations and uniform scaling as the models are re-centered at the origin and resized to fit in the unit cube. On the other hand, kd-trees are not invariant to rotations. This can be mitigated by using practices like pooling over different rotations (e.g. MVCNN) or branches that perform pose prediction and transformation (e.g. PointNet). However, we notice that simply having unaligned training data was enough to account for rotations in the classification task, and the ModelNet40 dataset contains plenty of unaligned shapes. Moreover, since the KDNet [96] also employs a kd-tree spatial data structure, the discussions there about transformations also apply to our method.

Computation time. Building a kd-tree of N points takes $O(N \log N)$ time, where $N = 2^{10}$ for 1K points. While PointNet does not require this step, it's also more than 2.0% worse in the classification task. The time to run a forward pass for classification is as follows: PointNet takes 25.3ms, while MRTNet takes 8.0ms on a TITAN

GTX1080, both with batch size of 8. Kd-tree building is also much faster than rendering a shape multiple times like in MVCNN [168] or voxelizing it [147]. Using 16 different test-time augmentations does not have significant impact in computational time, as the 16 versions are classified in the same batch. This number of test-time augmentations is comparable to other approaches, e.g. 10 in [96], 80 in [168], and 12 in [194] and [169].

2.4 Conclusion

In conclusion, we introduced multiresolution tree networks (MRTNet) for point cloud processing. They are flexible and can be used for shape classification, generation, and inference tasks. Our key idea is to represent a shape as a set of locality-preserving 1D ordered list of points at multiple resolutions, allowing efficient 1D convolution and pooling operations. The representation improves information flow across scales, enabling the network to perform coarse-to-fine analysis, leading to faster convergence during training and higher quality for shape generation.

In future work, we would like to incorporate additional point attributes, such as normal and color, into the network, to further improve accuracy of shape recognition and allow the shape generator to produce these attributes. We would also like to apply and extend the method to process spatio-temporal shape analysis, such as on animated shapes or other temporarlly changing 3D point data.

CHAPTER 3

LEARNING GENERATIVE MODELS OF SHAPE HANDLES

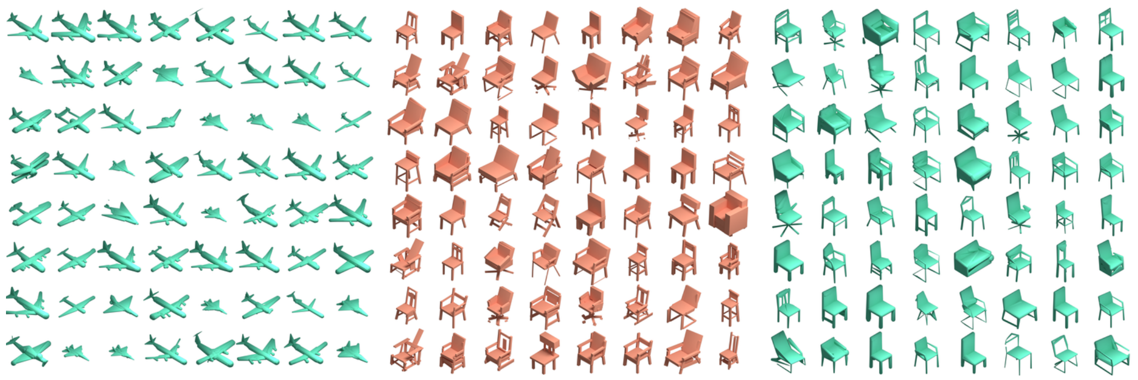


Figure 3.1: Gallery of 3D shapes generated as sets of handles *(zoom for details)*. We propose a class of generative models for synthesizing sets of handles – lightweight proxies that can be easily utilized for high-level tasks such as shape editing, parsing, animation etc. Our model can generate sets with different cardinality and is flexible to work with various types of handles, such as sphere-mesh handles [178] (first and third figures) and cuboids (middle figure).

Dramatic improvements in quality of image generation have become a key driving force behind many novel image editing applications. Yet, similar approaches are lacking for editing and generating 3D shapes. There are two related challenges. First, learning generative models for 3D data is challenging, as unlike images, high-quality 3D data is hard to obtain and the data is high dimensional and often unstructured. Second, regardless of whether good generative models are available, manipulating and editing 3D shapes in interactive applications is harder to users than editing images. For this reason, the geometry processing community has developed techniques for representing 3D data as a small collection of simpler *proxy shapes* [7,26,37,83,107,118, 207]. In this paper, we refer to these light-weight proxies as *shape handles* due to their ability to be easily manipulated by users. These representations have been widely used in tasks that require interaction and high-level reasoning in 3D environments, such as shape editing [57, 178], animation [179], grasping [120], and tracking [181].

We propose a generative models of shape handles. Our method adopts a two-branch network architecture to generate shapes with varying number of handles, where one branch focuses on generating handles while the other predicts the existence of each handle (Section 3.2.2). Furthermore, we propose a novel similarity measure based on distance fields to compare shape handle pairs. This measure can be easily adapted to accommodate various type of handles, such as cuboids and sphere-meshes [178] (Section 3.2.1). Finally, in contrast to previous work [136, 186] which focuses on unsupervised methods, we leverage recent works in collecting 3D annotations [122] as well as shape summarization techniques [178] to provide supervision to our approach. Experiments show that our method significantly outperforms previous methods on shape parsing and generation tasks. Using self-supervised training data generated by [178], our approach produces shapes that are twice as accurate as competing approaches in terms of intersection-over-union (IoU) metric. By employing human annotated data, our model can be further improved, achieving even higher accuracy than using self-supervised training data. Moreover, as shape handles provide a compact representation, our generative networks are compact (less than 10MB). Despite the small memory footprint, our method generates a diverse set of high quality 3D shapes, as seen in Figure 3.1.

Finally, our method is built towards generating shapes using representations that are amenable to manipulation by users. In contrast to point clouds and other 3D representations such as occupancy grids, handles are intuitive to modify and naturally suitable for editing and animation tasks. The latent space of shape handles induced by the learned generative model can be leveraged to support shape editing, completion, and interpolation tasks, as depicted in Figure 3.2.

3.1 Related work

Deep generative models of 3D shapes. Multiple 3D shape representations have been used in the context of deep generative models. 3D voxel grids [35, 52] are a natural extension to image-based architectures, but suffer from high memory footprint requirements. Sparse occupancy grids [70, 146, 174, 194] alleviate this issue using a hierarchical grid, but they are still not able to generate detailed shapes and they require additional bookkeeping. Multi-view representations [111, 166], point clouds [2,49,51,55], mesh deformations [87,193] and implicit functions [31,58,119,135] provide alternatives that are compact and capable of generating detailed shapes. However, these approaches are focused on reconstructing accurate 3D shapes and are not amenable to tasks like editing. Our goal is different: we explore generative models to produce sets of handles – summarized shape representations that can be easily manipulated by users.

Two closely related methods to ours are Tulsiani et al. [186] and Paschalidou et al. [136] where they propose models to generate shapes as a collection of primitives *without supervision*. In contrast, we are focused on creating models capable of utilizing shape decompositions provided by external agents; either a human annotator or a shape summarization technique. We demonstrate that, by using the extra information provided by annotations or well known geometric processing techniques, our method is capable of generating more accurate shapes while keeping the representation interpretable and intuitive for easy editing. Other approaches focused on learning shape structures through stronger supervision [104, 121, 130], requiring not only handle description, but also relationships between them, e.g. support, symmetry, adjacency, hierarchy, etc. In contrast, our method models shapes as sets and we show that inter-handle relationships can be learned directly from data, so that the latent space induced by our model can be used to guide shape editing, completion, and interpolation tasks. Furthermore, we present a general framework that can be easily adapted to different types of handles, not only a single parametric family, like cuboids [104, 121, 186] or superquadrics [136].

Methods for shape decomposition. Shape decomposition has been extensively studied by the geometry processing literature. The task is to approximate a complex shape as a set of simpler, lower-dimensional parts that are amenable for editing. We refer to these parts as *shape handles*. Early cognitive studies have shown that humans tend to reason about 3D shapes as a union of convex components [75]. Multiple approaches have explored decomposing shapes in this manner [85, 107, 225]. However, those approaches are likely to generate too many parts, making them difficult to manipulate. This problem was addressed by later shape approximation methods such as cages [207], 3D curves [57, 64, 118] and sphere-meshes [178], which are shown very useful in shape editing and other high-level tasks. Our method is flexible to work with various types of shape handles, and in particular we show experiments using cuboids as well as sphere-meshes.

Several closely related methods to ours approximate complex shapes using primitives such as cylinders [223] or cuboids [207]. These approximations are easy to interpret and manipulate by humans. However, most existing methods rely solely on geometric cues for computing primitives, which can lead to counter-intuitive decompositions. In contrast, our method takes supervision from semantic information provided by human annotators or shape summarization techniques, and therefore our results more accurately match human intuition.

3.2 Method

Consider a dataset $\mathcal{D} = \{S_i\}_{i=1}^{n}$ containing n sets of shape handles. Each set of handles S_i represents a 3D shape and consists of multiple handle descriptors. Our goal is to train a model f_θ capable of generating sets similar to the ones in $\mathcal{D}$, i.e., using them as supervision. More precisely, given an input x_i associated with a set

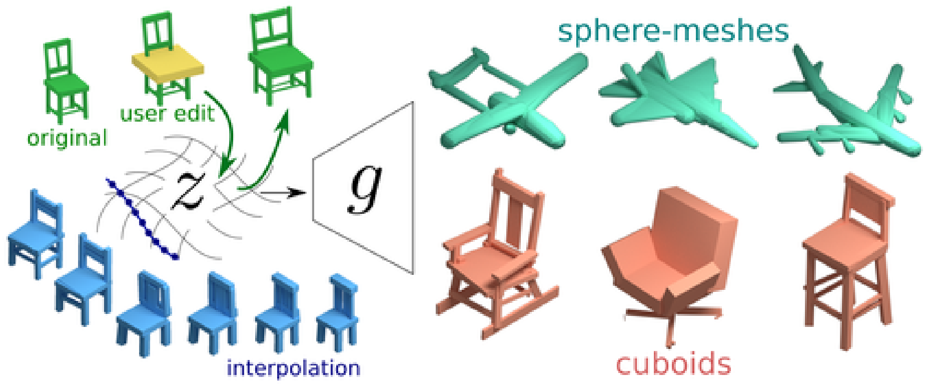


Figure 3.2: Overview. We propose a method to train a generative model g for sets of shape handles. Once trained, the latent representation z can also be used in applications like shape editing and interpolation.

of handles S_i, our goal is to estimate the parameters θ such that $f_\theta(x_i) \approx S_i$. The input x_i can be an image, a point cloud, an occupancy grid, or even the set of handles itself. When $x_i = S_i$, f_θ corresponds to an autoencoder. If we add a regularization term to the bottleneck of f_θ, we have a Variational Auto-Encoder (VAE), which we use for applications like shape editing, completion and interpolation (Section 3.3.4). However, we need to use a loss function capable of measuring the similarity between two sets of handles, *i.e.*the reconstruction component of a VAE. Ideally, this loss function would be versatile – we should be able to use it to generate different types of handles with minimal modifications. Moreover, our model needs to be capable of generating sets with different cardinalities, since the sets S_i do not always have the same size – in practice, the size of the sets used as supervision can vary a lot and our network must accommodate this need.

In this section, we describe how to create a model satisfying these constraints. First, we describe how to compute similarities between handles. Our method is flexible and only relies on the ability to efficiently compute the distance from an arbitrary point in space to the handle's surface. We then demonstrate how to use this framework with two types of handles: cuboids and sphere-meshes. Finally, we describe how to build models capable of generating sets with varying sizes, by employing a separate network branch to predict the existence of shape handles.

3.2.1 Similarity between shape handles

Consider two sets of shape handles of the same type: $A = \{a_j\}_{j=1}^{|A|}$ and $B = \{b_k\}_{k=1}^{|B|}$, where a_j and b_k are parameters that describe each handle. For example, if the handle type is cuboid, a_j (or b_k) would include the cuboid dimensions, rotation and translation in space. One way to compute similarity between sets is through Chamfer distance. Let the asymmetric Chamfer distance between the two sets of handles A and B be defined as:

$$Ch(A, B) = \frac{1}{|A|} \sum_{a \in A} \min_{b \in B} D(a, b) \tag{3.1}$$

where $D(a, b)$ is a function that computes the similarity between two handles with parameters a and b. There are multiple choices for $D(a, b)$. One straightforward choice is to define D as the ℓ_p-norm of the vector $a - b$. However, this is a poor choice as the parameters are not homogeneous. For example, parameters that describe rotations should not contribute to the similarity metric in the same way as those describing translations. Furthermore, there may be multiple configurations that describe the same shape – e.g., vertices that are in different orders may describe the same triangle; a cube can be rotated and translated in multiple ways and end up occupying the same region in space.

We address these problems by proposing a novel distance metric $D(a, b)$ which measures the similarity of the distance field functions of the two handles. Specifically, let $\mathcal{P}$ be a set of points in the 3D space and let $\mu(a)$ represent the surface of the handle described by a. Now, we define D as follows:

$$D(a, b) = \sum_{p \in \mathcal{P}} \Big(\min_{p_a \in \mu(a)} \|p - p_a\|_2 - \min_{p_b \in \mu(b)} \|p - p_b\|_2 \Big)^2 \tag{3.2}$$

Intuitively, D calculates the sum of squared differences between two feature vectors representing the distance fields with respect to each of the handles. Each dimension of these feature vectors contains the distance between a point in a set of *probe points* $\mathcal{P}$ and the surface of the handle defined by its parameters (a and b in Equation 3.2). The main advantage of this similarity computation is its versatility: it allows us to compare any types of shape handles; the only requirement is the ability to efficiently compute $\min_{p_h \in \mu(h)} \|p - p_h\|_2$ given handle parameters h and a point p. In the following subsections, we show how to efficiently perform this computation for two types of shape handles: cuboids and sphere-meshes.

Cuboids. We choose to represent a cuboid by parameters $h = \langle \mathbf{c}, \mathbf{l}, \mathbf{r_1}, \mathbf{r_2} \rangle$, where $\mathbf{c} \in \mathbb{R}^3$ is the cuboid center, $\mathbf{l} \in \mathbb{R}^3$ is the cuboid scale factor (i.e. dimensions), $\mathbf{r_1}, \mathbf{r_2} \in \mathbb{R}^3$ are vectors describing the rotation of the cuboid. This rotation representation has continuity properties that benefit its estimation through neural networks [224]. Notice that we can build a rotation matrix $\mathbf{R}$ from $\mathbf{r_1}$ and $\mathbf{r_2}$ by following the procedure described in [224]. Now, consider the transformation $\tau_h(p) = \mathbf{R}^T p - \mathbf{c}$. Let $\mu_C(h) \in \mathbb{R}^3$ represent the surface of the cuboid parametrized by h. We can compute $\min_{p_h \in \mu_C(h)} \|p - p_h\|_2$ (i.e. distance from p to the cuboid) as follows:

$$\min_{p_h \in \mu_C(h)} \|p - p_h\|_2 = \left\| (|\tau_h(p)| - \mathbf{l})^+ \right\|_2 + \big(\max(|\tau_h(p)| - \mathbf{l}) \big)^-$$

where $(\cdot)^+$, $(\cdot)^-$ and $|\cdot|$ represent element-wise $\max(\cdot, 0)$, $\min(\cdot, 0)$ and absolute value, respectively. Since this expression can be computed in $O(1)$, we are able to compute Equation 3.2 in $O(|\mathcal{P}|)$, where the number of probing points $|\mathcal{P}|$ is relatively small. In practice, we sample 64 points in a regular grid in the unit cube.

Sphere-meshes. A triangle mesh consists of a set of vertices and triangular faces representing the vertex connectivity. Every vertex is a point in space and the surface of

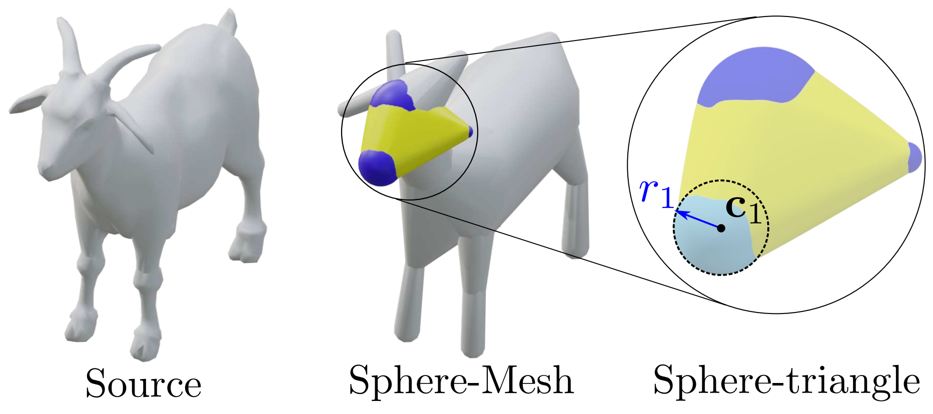


Figure 3.3: Schematic representation of sphere-meshes. A sphere-mesh (middle) is computed from a regular triangle mesh (left) as input, and it consists of multiple sphere-triangles (right), each of which is a volumetric representation

a triangle contains all the points that can be generated by interpolating the triangle's vertices using barycentric coordinates. A sphere-mesh is a generalization of a triangle mesh – every vertex is a sphere instead of a point in space. Thus, every sphere-mesh "triangle" is actually a volume delimited by the convex-hull of the spheres centered at the triangle vertices. Figure 3.3 presents a visual description of sphere-mesh components. Thiery et al. [178] introduced an algorithm to compute sphere-meshes from regular triangle meshes. They show that complex meshes can be closely approximated with a sphere-mesh containing a fraction of the original components.

We model sphere-meshes as a set of sphere-mesh triangles, called *sphere-triangles*. Similarly to a regular triangle, a sphere-triangle is fully defined by its vertices, the difference being that its vertices are now spheres instead of points. Thus, we choose to represent a sphere-triangle using parameters $h = \langle r_1, r_2, r_3, \mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3} \rangle$; where $\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3} \in \mathbb{R}^3$ are the centers of the three spheres, and $r_1, r_2, r_3 \in \mathbb{R}^+$ are their radii. Let $\mu_T(h)$ represent the the surface of the sphere-triangle parametrized by h. For calculating the similarity between two sphere-triangles: as each sphere-triangle is uniquely defined by its three spheres, it suffices to have μ_T contain only the surfaces of these three spheres, and hence it does not need to contain the entire sphere triangle. Thus, the distance of a probing point p to the handle surface is computed as follows:

$$\min_{p_h \in \mu_T(h)} \|p - p_h\|_2 = \min_{i \in \{1,2,3\}} (\|p - \mathbf{c}_i\|_2 - r_i).$$

3.2.2 Generating sets with varying cardinality

The neural network f generates shapes represented by sets of handles given an input x. Our design of f includes two main components: an encoder q that, given an input x, outputs a latent set representation z; and a decoder g that, given the latent set representation z, generates a set of handles. Even though we can use a symmetric version of Equation 3.1 to compute the similarity between the generated set $g(q(x_i))$ and the ground-truth set of handles S_i, so far our model has not taken into account the varying size (i.e. number of elements) of the generated sets. We address this issue by separating the generator into two parts: a parameter prediction branch g_p and an existence prediction branch g_e. The parameter prediction branch is trained to always output a fixed number of handle parameters where $[g_p(z)]_i$ represents the parameters of the i^{th} handle. On the other hand, the existence prediction branch $[g_e(z)]_i \in [0, 1]$ represents the probability of existence of the i^{th} generated handle. Now, we need to adapt our loss function to consider the probability of existence of a handle.

If we assume that all handles exist, our model can be trained using the following loss:

$$\mathcal{L} = Ch(g_p(z_i), S_i) + Ch(S_i, g_p(z_i)),$$

where S_i is a set of shape handles drawn from the training data and z_i is a latent representation computed from the associated input x_i. However, we want to modify this loss to take into account the probability of a handle existing or not. To do so, note that $\mathcal{L}$ has two terms. The first term measures accuracy: i.e. how close each of the handles in $g_p(z_i)$ is from the handles in S_i. For this term, we can use g_e as weights for the summation in Equation 3.1, which leads to the following definition:

$$P(z, S) = \sum_{i=1}^{K} \min_{s \in S} D([g_p(z)]_i, s)[g_e(z)]_i, \tag{3.3}$$

where z is a latent space representation, S is a set of handles and $K = |g_p(z)| = |g_e(z)|$. The intuition is quite simple: if the i^{th} handle is likely to exist, its distance to the closest handle should be taken into consideration; on the other hand, if the i^{th} handle is unlikely to exist, it does not matter if it is approximating a handle in S or not.

The second term in $\mathcal{L}$ measures coverage: every handle in S_i must have (at least) one handle in the generated set that is very similar to it. Here, we use an insight presented in [136] to efficiently compute the coverage of S_i while considering the probability of elements in a set existing or not. Let $g_p^s(z)$ be the list of generated handles $g_p(z)$ ordered in non-decreasing order according to $D([g_p^s]_i, s)$ for $i = 1, ..., |g_p(z)|$. We compute the coverage of a set S from a set generated from z as follows:

$$C(z, S) = \sum_{s \in |S|} \sum_{i=1}^{K} D([g_p^s(z)]_i, s)[g_e^s(z)]_i \prod_{j=1}^{i-1} (1 - [g_e^s(z)]_j). \tag{3.4}$$

The idea behind this computation is the following: for every handle $s \in S$, we compute its distance to every handle in $g_p(z)$, weighted by the probability of that handle existing or not. However, the distance to a specific handle is important only if no other handle closer to s exists. Thus, the whole term needs to be weighted by $\prod_{j=1}^{i}(1 - [g_e^s(z)]_j)$. Finally, we can combine Equations 3.3 and 3.4 to define the reconstruction loss $\mathcal{L}_{rec}$ used to train our model:

$$\mathcal{L}_{rec} = P(z, S) + C(z, S). \tag{3.5}$$

3.2.2.0.1 Alternate training procedure. Although minimizing the loss in Equation 3.5 at once enables generating sets of different sizes, our experiments show that

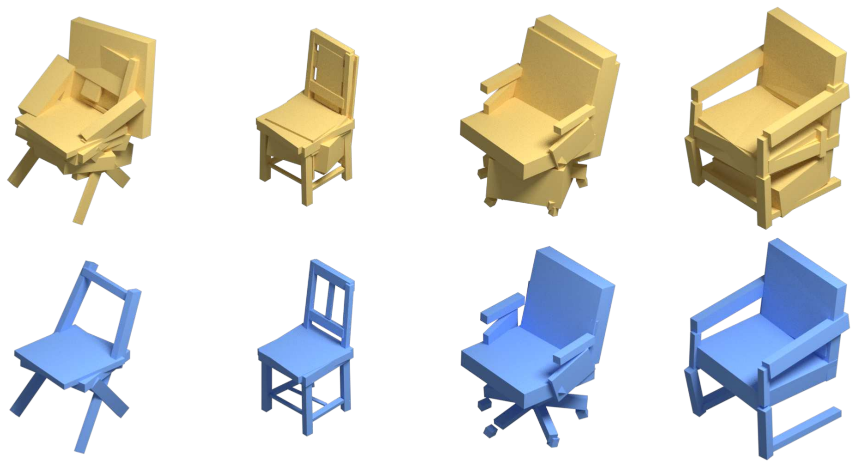


Figure 3.4: Comparison of results after the first stage (top row) and second stage (bottom row) of alternate training. While the first stage ensures coverage, some extra, unnecessary handles are also generated. The second stage trains the existence branch, which assigns a low probability of existence to the inaccurate handles.

the results can be further improved if we train g_p and g_e in an alternating fashion. Specifically, we first initialize the biases and weights of the last layer of g_e to ensure that all of its outputs are 1, i.e., the model is initialized to predict that every primitive exists. Then, in the first stage of the training, we fix the parameters of g_e and train g_p minimizing only the coverage $C(z, S)$. During the second stage of the training, we fix the parameters of g_p and update the parameters of g_e, but this time minimizing the full reconstruction loss $\mathcal{L}_{rec}$. As we show in Section 4, this alternating procedure improves the training leading to the generation of more accurate shape handles. The intuition is that while training the model to predict the handle parameters (g_p), the network should be only concerned about coverage, i.e., generating at least one similar handle for each ground-truth handle. On the other hand, while training the existence prediction branch (g_e), we want to remove the handles that are in incorrect positions while keeping the coverage of the ground-truth set.

3.3 Experiments

This section describes our experiments and validates results. We experimented with two different types of handles: cuboids computed from PartNet [122] segmentations and sphere-meshes computed from ShapeNet [24] shapes using [178]. We compare our results to two other approaches focused on generating shapes as a set of simple primitives, namely cuboids [186] and superquadrics [136]. All the experiments in the paper were implemented using Python 3.6 and PyTorch. Computation was performed on TitanX GPUs.

3.3.1 Datasets

Cuboids from PartNet [122]. We experiment with human annotated handles by fitting cuboids to the parts segmented in PartNet [122]. The dataset contains 26,671 shapes from 24 categories and 573,585 part instances. In order to compare our model with other approaches trained on the ShapeNet [24] chairs dataset, we select the subset of PartNet chairs that is also present in ShapeNet. This results in 6773 chair models segmented in multiple parts. Every model has on average 18 parts, but there are also examples with as many as 137 parts. For every part we fit a corresponding cuboid using PCA. Then, we compute the volume of every cuboid and keep at topmost 30 cuboids in terms of volume. Notice that 92% of the shapes have less than 30 cuboids, so those remain unchanged. The others will have missing components, but those usually correspond to very small details and can be ignored without degrading the overall structure.

Sphere-meshes from ShapeNet [24]. In contrast to cuboids (which are computed from human annotated parts), we compute sphere-meshes fully automatically using the procedure described in [178]. We use ShapeNet categories that are also analyzed in [136, 186]: chairs, airplanes and animals. The sphere-mesh computation procedure requires pre-selecting how many sphere-vertices to use. The algorithm starts by considering the regular triangle mesh as a trivial sphere-mesh (vertices with null radius) and then decimates the original mesh progressively through edge collapsing, optimizing for new sphere-vertex each time an edge is removed. This procedure is iterated until the required number of vertices is achieved.

In our case, since our model is capable of generating sets with different cardinalities, we are not required to set a fixed number of primitives for every shape. Therefore we use the following method to compute a sphere-mesh with adaptive number of vertices. Specifically, for every shape in the dataset, we start by computing a sphere-mesh with 10 vertices. Then, we sample 10K points both on the sphere-mesh surface and the original mesh. If the Hausdorff distance between the point clouds is smaller than $\epsilon = 0.2$ (point clouds are normalized to fit the unit sphere), we keep the current computed sphere-mesh. Otherwise, we compute a new sphere-mesh by incrementing the number of vertices. This procedure continues until we reach a maximum of 40 vertices. This adaptive sphere-mesh computation allows our model to achieve a good balance between shape complexity and summarization – simpler shapes will be naturally represented with a smaller number of primitives. We note that the sphere-mesh computation allows the resulting mesh to contain not only triangles, but also edges (i.e. degenerate triangles). For simplicity, we make no distinction between sphere-triangles or edges: edges are simply triangles that have two identical vertices.

3.3.2 Shape Parsing

The shape parsing task is to compute a small set of primitives from non-parsimonious, raw, 3D representations, like occupancy grids, meshes or point clouds. We analyze the ability of our model in performing shape parsing using a similar setup to [136, 186]. Specifically, following the notation defined in Section 3.2, we train a model f_θ using input-output pairs $\langle x_i, S_i \rangle$, where x_i corresponds to a point cloud with 1024 points and S_i is a set of handles summarizing the shape represented by x_i. We use a Point-

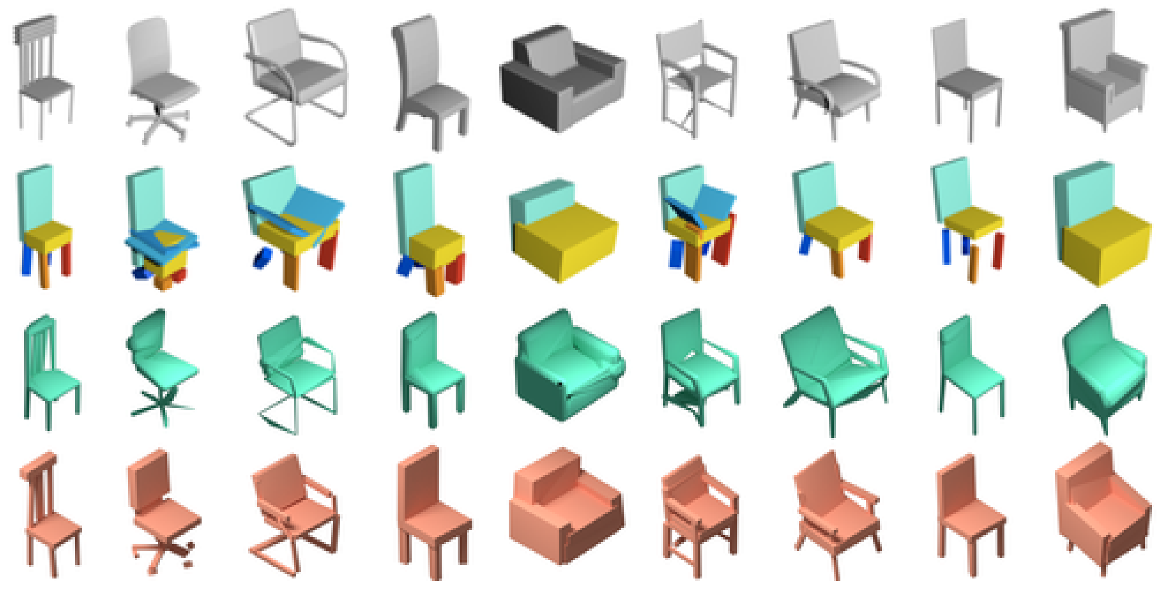


Figure 3.5: Shape parsing on the chairs dataset. From top to bottom, we show ground-truth shapes, results by Tulsiani et al. [186], results by our method using sphere-mesh handles, and our method using cuboids handles. Note how our results (last two rows) are able to generate handles with much better details such as the stripes on the back of the chair (first column), legs on wheel chairs (second column) and armrests in several other columns.

Net [169] encoder to process a point cloud with 1024 points and generate a 1024 dimensional encoding. This encoding is then used as an input for our two-branched set decoder. Both branches follow the same architecture: 3 fully connected layers with 256 hidden neurons followed by batch normalization and ReLU activations. The only difference between the two branches is in the last layer. Assume N is the maximum set cardinality generated by our model and D is the handle dimensionality (i.e. number of parameters of each handle descriptor, which happens to be $D = 12$ for both sphere-mesh and cuboid). Then g_p outputs $N \times D$ values followed by a tanh activation, while g_e outputs N values followed by a sigmoid activation. We set $N = 30$ for cuboid handles and $N = 50$ for sphere-meshes. The model is trained end-to-end by using the alternating training described in Section 3.2. Training is performed using the Adam optimizer with a learning rate of 10^{-3} for 5K iterations in each stage.

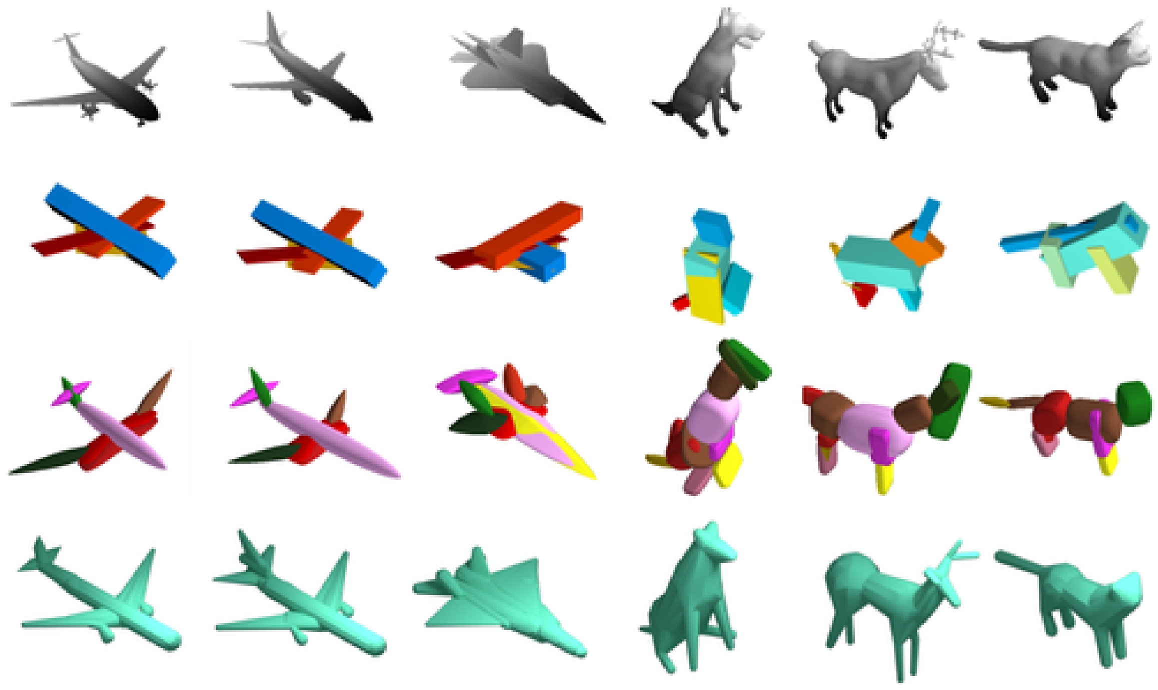


Figure 3.6: Shape parsing on the airplanes and animals datasets. From top to bottom, we show ground-truth shapes, results by Tulsiani et al. [186], results by Paschalidou et al. [136], and results by our model trained using sphere-mesh handles. Our results contain accurate geometric details, such as the engines on the airplanes and animal legs that are clearly separated.

Figures 3.5 and 3.6 show visual comparisons of our method with previous work. Qualitatively, our method generates shape handles with accurate geometric details, including many thin structures that previous methods struggle with.

Quantitative evaluation. We compare our method against [136, 186] using intersection over union (IoU) metric and results are shown in Table 3.1. As expected, when using cuboids as handles, our method leverages the annotated data from the PartNet [122] to achieve significantly more accurate shape approximations (more than twice the IoU in comparison). On the other hand, as [136, 186] are trained without leveraging annotated data, a more fair comparison is between theirs and our method using sphere-mesh handles, which are computed automatically. Our method still clearly outperforms theirs in all categories – chairs, airplanes and animals. This

	Handle type	**Category**		
		Chairs	Airplanes	Animals
[186]	Cuboid	0.129	0.065	0.334
[136]	Superquadric	0.141	0.181	0.751
Ours	Cuboid	0.311	-	-
	Sphere-mesh	**0.298**	**0.307**	**0.761**

Table 3.1: Quantitative results for shape parsing. Intersection over union computed on the reconstructed shapes. The best self-supervised results are shown in bold font.

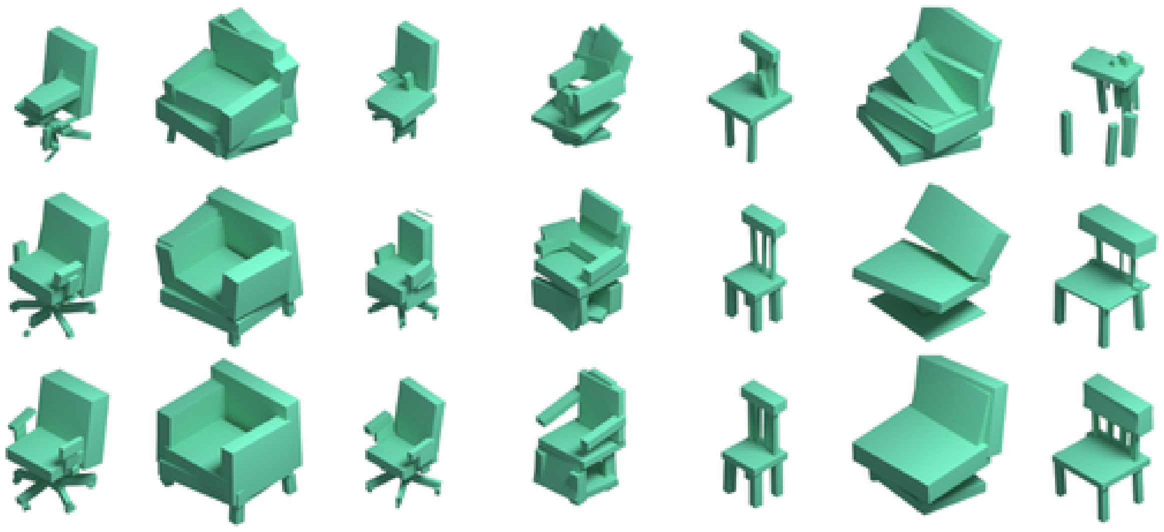


Figure 3.7: Ablation studies. Shapes generated from a model trained without our proposed handle similarity metric (first row), model trained without the two-stage training procedure (second row), and our full model (last row). Note that comparing handles using just ℓ_2-norm (first row) yields poor results. Training g_p and g_e at the same time (instead of alternating) yields reasonable results, but some parts are missing and/or poorly oriented.

shows that even though a neural network in theory should be able to learn the best parsimonious shape representations, using self-supervision generated by shape summarization techniques (e.g. sphere-meshes) can still help it achieve more accurate approximations.

3.3.3 Ablation studies

We investigated the influence of the two main contributions of this work: the similarity metric for handles and the alternating training procedure for g_p and g_e.

w/o similarity	w/o alternate	full model
0.192	0.320	**0.352**

Table 3.2: Quantitative results of ablation studies comparing our full model with two variations that lack our handle similarity metric and alternate training procedure respectively.

To do so, we adopt a shape-handle auto-encoder and compare different variations by computing the IoU of reconstructed shapes in a held-out test set. The auto-encoder architecture is very similar to the one used in shape parsing, except for the encoder – it still follows a PointNet architecture, but every "point" is actually a handle treated as a point in a D-dimensional space. We analyzed three different variations. In the first one, we simply used the ℓ_2-norm between the handle parameters (cuboids, in this case). As shown in Figure 3.7 and Table 3.2, the proposed handle similarity metric has a significant impact on the quality of the generated shapes. The second variation consists of training the same model, but without using the alternating procedure described in Section 3.2. Figure 3.7 shows that the alternating training procedure generates more accurate shapes, with fewer missing parts and better cuboid orientation.

3.3.4 Applications

In this section, we demonstrate the use of our generative model in several applications. We employed a Variational Auto-Encoder (VAE) [94] for this purpose. It follows the same architecture as the auto-encoder described in Section 3.3.3 with the only difference being that the output of the encoder (latent representation z) has dimensionality 256 instead of 512. Additionally, following [55], we added an additional regularization term to the training objective:

$$\mathcal{L}_{reg} = \|cov(Q(x) + \delta)\|_2 + \mathbb{E}_{x \sim \mathcal{D}}[Q(x)] \tag{3.6}$$

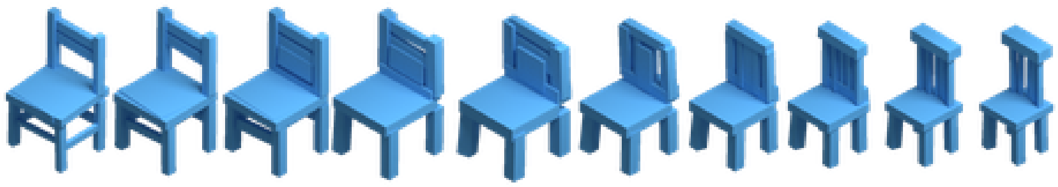


Figure 3.8: Latent space interpolation Sets of handles can be interpolated by linearly interpolating the latent representation z. Transitions are smooth and generate plausible intermediate shapes. Notice that the interpolation not only changes handle parameters, but also adds new handles / removes existing handles as necessary.

where Q is the encoder, $cov(\cdot)$ is the covariance matrix, $\|\cdot\|_2$ is the Frobenius norm, x is input handle set and δ is random noise sampled from $\mathcal{N}(0, cI)$. Thus, the network is trained minimizing the following function:

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{reg}. \tag{3.7}$$

In all our experiments, we used $\lambda = 0.1$ and $c = 0.01$. The model is trained using the alternate procedure described before, $i.e. \mathcal{L}_{rec}$ is replaced by $C(z, S)$ while training g_p.

Interpolation. Once the VAE model is trained, we are able to morph between two shapes by linearly interpolating their latent representations z. In particular, we sample two values z_1, z_2 from $\mathcal{N}(0, I)$ and generate new shapes by passing the interpolated encodings $\alpha z_1 + (1-\alpha) z_2$ through the decoder g, where $\alpha \in [0, 1]$. Results using cuboid handles are presented in Figure 3.8. Note that the shapes are smoothly interpolated, with new handles added and old handles removed as necessary when the overall shape deforms. Additionally, relationships between handles, 3.4.4.0.1 Shape editing.like symmetries, adjacency and support, are preserved, thanks to the latent space learned by our model, even though such characteristics are never explicitly specified as supervision.

Handle completion. Consider an incomplete set of handles $A = \{a_i\}_{i=1}^{N}$ as input, the handle completion task is to generate a complete set of handles A^*, such that A^* contains not only the handles in the input A but also necessary additional handles that result in a plausible shape. For example, given a single cuboid handle as shown in Figure 3.9, we want to generate a complete chair that contains that input handle. We perform this task by finding a latent representation z^* that generates a set of handles approximating the elements in A. Specifically, we solve the following optimization problem:

$$z^* = \underset{z \in \mathcal{Z}}{\operatorname{argmin}}\, C(z, A), \quad A^* = g(z^*), \tag{3.8}$$

where C is the coverage metric defined in Equation 3.4 and A^* is the completed shape (i.e. output of the decoder using z^* as input). We can also use the existence prediction branch (g_e) in this framework to reason about how complex we want the completed shapes to be. Specifically, we add an additional term to the optimization:

$$z^* = \underset{z \in \mathcal{Z}}{\operatorname{argmin}}\, C(z, A) + \gamma \sum_{i=1}^{N} [g_e(z)]_i, \tag{3.9}$$

where γ controls the complexity of the shape. If $\gamma = 0$, we are not penalizing a set with multiple handles – only coverage matters. As γ increases, existence of multiple handles is penalized more, leading to a solution with a lower cardinality. As can be seen in Figure 3.9, our model is capable of recovering plausible chairs even when given a single handle. In addition, we can generate multiple proposals for A^* by initializing the optimization with different values of z. More results can be found in the supplemental material.

Shape editing. For editing shapes, we use a similar optimization based framework. Consider an original set of handles A describing a particular shape. Assume that the user made edits to A by modifying the parameters of some handles, creating a new set A'. Our goal is to generate a plausible new shape A^* from A', while minimizing the deviation from the original shape. To achieve this goal, we solve the following minimization problem via gradient descent:

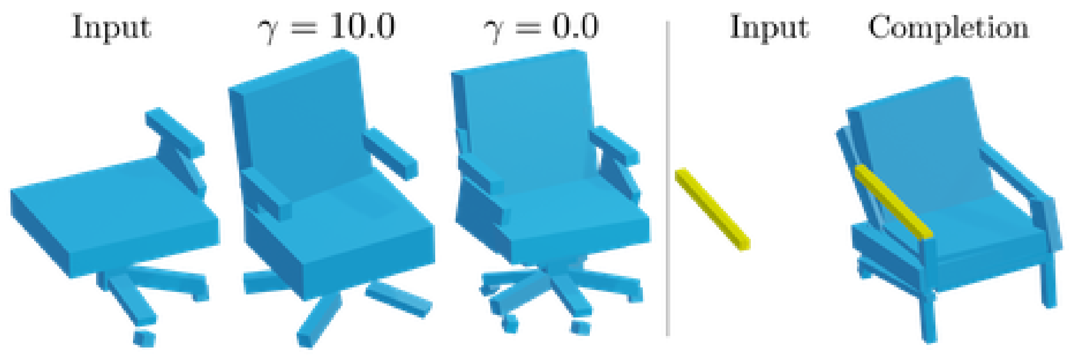


Figure 3.9: Results of handle completion. Recovering full shape from incomplete set of handles. Using γ to control the complexity of the completed shape (left). Predicting a complete chair from a single handle (right).

3.4.4.0.1 Shape editing.

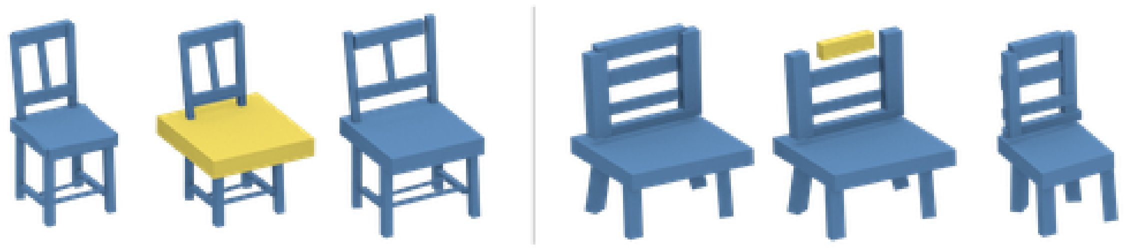


Figure 3.10: Editing chairs. Given an initial set of handles, a user can modify any handle (yellow). Our model then updates the entire set of handles, resulting in a modified shape which observes the user edits while preserving the overall structure.

$$z^* = \operatorname*{argmin}_{z \in \mathcal{Z}} C(z, A') + \gamma \left\| z - z_A \right\|_2, \quad A^* = g(z^*) \tag{3.10}$$

where z_A is the latent representation of the original shape. The intuition for Equation 3.10 is simple: we want to generate a plausible shape that approximates the user edits by minimizing $C(z, A')$ but also keep the overall characteristics of the original shape A by adding a penalty for deviating too much from z_A. Results are shown in Figure 3.10. As observed in the figure, when the user edits one of the handles, our model can automatically modify the shape of the entire chair while preserving its overall structure.

Limitations. Our method has several limitations to be addressed in future work. First, during training we set a maximum number of handles to be generated. Increasing this number would allow more complex shapes but also entail a larger network with higher capacity. Therefore, there is a trade-off between the compactness of the generative model and the desired output complexity. Furthermore, our method currently does not guarantee the output handles observe certain geometric constraints, such as parts that need to be axis-aligned or orthogonal to each other. For man-made shapes, these are often desirable constraints and even slight deviation is immediately noticeable. While our model can already learn geometric relationships among handles from the data directly, generated shapes might benefit from additional supervision enforcing geometric constraints.

3.4 Conclusion

We presented a method to generate shapes represented as sets of handles – lightweight proxies that approximate the original shape and are amenable to high-level tasks, like shape editing, parsing and animation. Our approach leverages pre-defined sets of handles as supervision, either through annotated data or self-supervised methods. We proposed a versatile similarity metric for shape handles that can easily accommodate different types of handles, and a two-branch network architecture to generate handles with varying cardinality. Experiments show that our model is capable of generating compact and accurate shape approximations, outperforming previous work. We demonstrate our method in a variety of applications, including interactive shape editing, completion, and interpolation, leveraging the latent space learned by our model to guide these tasks. In the next chapter, we will investigate how to use pre-defined sets of handles (convex polytopes) as supervisory signal for learning per-point embeddings that can be used in discriminative models.

CHAPTER 4

LABEL-EFFICIENT LEARNING ON POINT CLOUDS USING APPROXIMATE CONVEX DECOMPOSITIONS

The performance of current deep neural network models on tasks such as classification and semantic segmentation of point cloud data is limited by the amount of high quality labeled data available for training the networks. Since in many situations collecting high quality annotations on point cloud data is time consuming and incurs a high cost, there has been increasing efforts in circumventing this problem by training the neural networks on noisy or weakly labeled datasets [158], or training in completely unsupervised ways [30, 55, 72, 212, 214].

A ubiquitous technique in training deep networks is to train the network on one task to initialize its parameters and learn generically useful features, and then fine-tune the network on the final task. In particular, there has been great interest in so-called *self-supervised* tasks for initialization. These tasks, which do not require any human annotations, allow the network to be initialized by using various techniques to generate labels automatically, *i.e.*, in a self-supervised manner – *e.g.*tasks such as clustering, solving jigsaw puzzles, and colorization. There have been a few recent attempts to come up with similar tasks that help with 3D data [30, 72]. The overarching question here is *"what makes for a good self-supervision task?"* – what are the useful inductive biases that our model learns from solving such a task that is beneficial to the actual downstream target task we are interested in solving.

We propose using a classical shape decomposition method, Approximate Convex Decomposition (ACD), as the self-supervisory signal to train neural networks built to

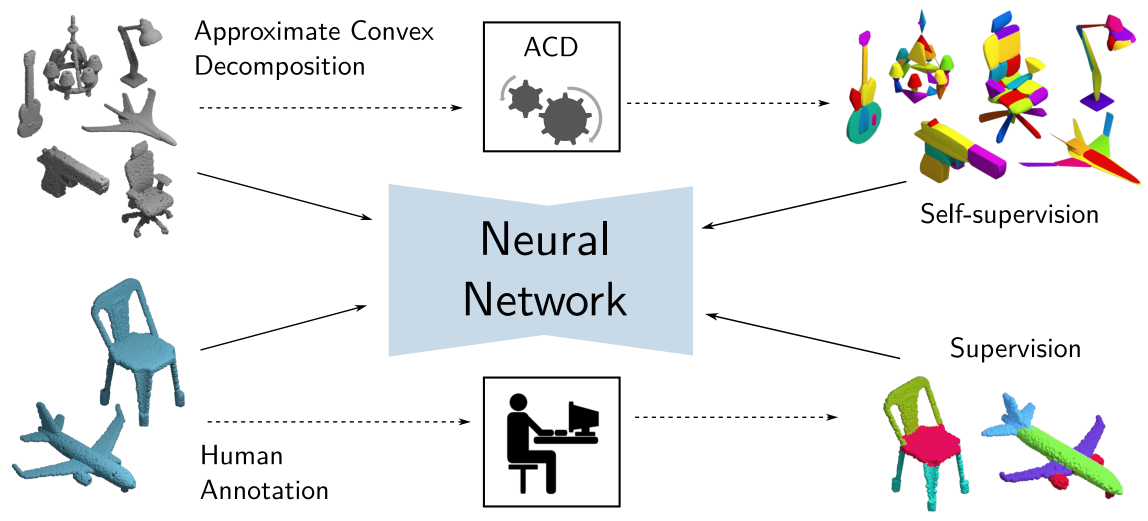


Figure 4.1: Overview of our method v.s. a fully-supervised approach. ***Top:*** Approximate Convex Decomposition (ACD) can be applied on a large repository of unlabeled point clouds, yielding a *self-supervised* training signal for the neural network without involving any human annotators. ***Bottom:*** the usual *fully-supervised* setting, where human annotators label the semantic parts of point clouds, which are then used as supervision for the neural network. The unsupervised ACD task results in the neural network learning useful representations from unlabeled data, significantly improving performance in shape classification and semantic segmentation when labeled data is scarce or unavailable.

process 3D data. We posit that being able to decompose a shape into geometrically simple constituent parts provides an excellent self-supervisory learning signal for such purposes. As shown in the Figure 4.2, ACD decomposes shapes into segments that roughly align with instances of different parts, *e.g.*two wings of an airplane are decomposed into two separate approximately convex parts. Many man-made shapes are influenced by physical and geometric constraints. Convex parts tend to be easily manufactured, and are strong and aerodynamic, thus fulfilling the above requirements. However, strictly convex decomposition often leads to highly over-segmented shapes. For that reason, we chose *approximate* convex decomposition, which we show benefits a number of learning tasks.

Our approach is illustrated in Figure 4.1. The main idea is to automatically generate training data by decomposing unlabeled 3D shapes into convex components.

Since ACD relies solely on geometric information to perform its decomposition, the process does not require any human intervention. From the model perspective, we formulate ACD as a metric learning problem on point embedding and train the model using a contrastive loss [34,69]. We demonstrate the effectiveness of our approach on standard 3D shape classification and segmentation benchmarks. In classification, we show that the representation learned from performing shape decomposition leads to features that achieve state-of-the-art performance on ModelNet40 [206] unsupervised shape classification (**89.8**%). For few-shot part segmentation on ShapeNet [24], our model outperforms the state-of-the-art by **7.5%** mIoU when using 1% of the available labeled training data. Moreover, differently from other unsupervised approaches, our method can be applied to any of the well-known neural network backbones for point cloud processing. Finally, we provide thorough experimental analysis and visualizations demonstrating the role of the ACD self-supervision on the representations learned by neural networks.

4.1 Related Work

Learning Representations on 3D data. Shape representations using neural networks have been widely studied in the field of computer graphics and computer vision. Occupancy grids have been used to represent shape for classification and segmentation tasks [116]; however it suffered from issues of computational and memory efficiency, which were later circumvented by architectures using spatial partitioning data structures [97,147,194,195]. Multi-view approaches [77,86,175] learn representations by using order invariant pooling of features from multiple rendered views of a shape. Another class of methods take a point cloud representation (*i.e.*a set of (x, y, z) co-ordinates) as input, and learn permutation invariant representations [55,72,140,141,198,212]. Point clouds are a compact 3D representation that does not suffer from the memory constraints of volumetric representations nor the visibility issues of multi-view approaches. However, all these approaches rely on massive amounts of labeled 3D data. In this paper, we focus on developing a technique to allow label-efficient representation learning of point clouds. Our approach is architecture-agnostic and relies on learning approximate convex decompositions, which can be automatically computed from a variety of shape representations without any human intervention.

Approximate Convex Decompositions. Early cognitive science literature has demonstrated that humans tend to reason about 3D shapes as a union of convex components [75]. However, performing exact convex decomposition is a NP-Hard problem that leads to an undesirable high number of components on realistic shapes [11]. Thus, we are interested in a particular class of decomposition techniques named *Approximate Convex Decomposition* (ACD) [85,107,114,225], which compute components that are approximately convex – up to a concavity tolerance ϵ. This makes the computation significantly more efficient and leads to shape approximations containing a smaller number of components. These approximations are useful for a variety of tasks, like mesh generation [107] and collision detection [199]. ACDs are also an important step in non-parametric shape segmentation methods [8,85]. Furthermore, ACD is shown to have a low rand-index compared to human segmentations in the PSB benchmark [85], which indicates that it is a reasonable proxy for our intuitions of shape parts. In this work, we used a particular type of ACD named Volumetric Hierachical Approximate Convex Decomposition (V-HACD) [114] – details in Section 4.2.1. Differently from non-parametric approaches, our goal is to use ACD as a self-supervisory task to improve point cloud representations learned by deep neural networks. We show not only that the training signal provided by ACD leads to improvements in semantic segmentation, but also to unsupervised shape classification.

Self-supervised learning. In many situations, unlabeled images or videos themselves contain information which can be leveraged to provide a training loss to learn useful representations. Self-supervised learning explores this line of work, utilizing unlabeled data to train deep networks by solving proxy tasks that do not require any human annotation effort, such as predicting data transformations [60,131,132] or clustering [21, 22]. Learning to colorize grayscale images was among the first approaches to training modern deep neural networks in a self-supervised fashion [101, 219, 220] – being able to predict the correct color for an image requires some understanding of a pixel's semantic meaning (*e.g.*skies are blue, grass is green etc.), leading to learning representations useful in downstream tasks like object classification. The contextual information in an image also lends itself to the design of proxy tasks – learning to predict the relative positions of cropped image patches as in Doersch *et al.* [42], similarity of patches tracked across videos [196, 197], inpainting a missing patch in an image by leveraging the context from the rest of the image [138, 183]. Motion from unlabeled videos also provides a useful pre-training signal, as shown in Pathak *et al.* [137] using motion segmentation, and Jiang *et al.* [84] who predict relative depth as pre-training for downstream scene understanding tasks. Other approaches include solving jigsaw puzzles with permuted image patches [131] and training a generative adversarial model [43]. An empirical comparison of various self-supervised tasks may be found in [65, 98]. In the case of limited samples *i.e.*the *few-shot classification* setting, including self-supervised losses along with the usual supervised training is shown to be beneficial in Su *et al.* [171]. Recent work has also focused on learning unsupervised representations for 3D shapes using tasks such as clustering [72] and reconstruction [157, 214], which we compare against in our experiments.

Label-efficient representation learning on point clouds. Several recent approaches [30, 72, 124, 158, 226] have been proposed to alleviate expensive labeling of shapes. Muralikrishnan *et al.* [124] learn per-point representation by training the network to predict shape-level tags. Yi et al. [215] embeds pre-segmented parts in descriptor space by jointly learning a metric for clustering parts, assigning tags to them, and building a consistent part hierarchy. Another direction of research is to utilize noisy/weak labels for supervision. Chen *et al.* [30] proposed a branched auto-encoder, where each branch learns coarse part level features, which are further used to reconstruct the shape by producing implicit fields for each part. However, this approach requires one decoder for every different part, which restricts their experiments to category-specific models. On the other hand, our approach can be directly applied to any of the well known point-based architectures, being capable of handling multiple categories at once for part segmentation and learning state-of-the-art features for unsupervised shape classification. Furthermore, [30] shows experiments on single shot semantic segmentation on manually selected shapes, whereas we show results on randomly selected training shapes in few-shot setting. Most similar to our work, Hassani *et al.* [72] propose a novel architecture for point clouds which is trained on multiple tasks at the same time: clustering, classification and reconstruction. In our experiments, we demonstrate that we outperform their method on few-shot segmentation by **7.5**% IoU and achieve the same performance on unsupervised ModelNet40 classification by using *only ACD as a proxy task*. If we further add a reconstruction term, our method achieves state-of-the-art performance in unsuperivsed shape classification. Finally, Sharma *et al.* [158] proposed learning point embedding by utilizing noisy part labels and semantic tags available freely on a 3D warehouse dataset. The model learnt in this way is used for a few-shot semantic segmentation task. In this work, we instead get part labels using approximate convex decomposition, whose computation is completely automatic and can be applied to any mesh regardless of the existence of semantic tags.

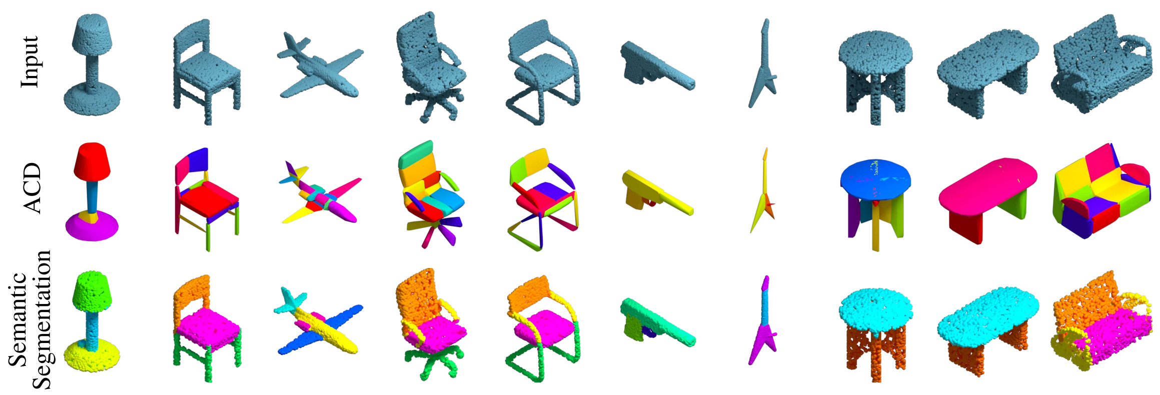


Figure 4.2: Input point clouds (first row), convex components *automatically* computed by ACD (second row) and *human-labeled* point clouds (last row) from the ShapeNet [24] part segmentation benchmark. Note – ***(i)*** different colors for the ACD components only signify different parts– no semantic meaning or inter-shape correspondence is inferred by this procedure; ***(ii)*** for the human labels, colors do convey semantic meaning: *e.g.*the backs of chairs are always orange; ***(iii)*** while the ACD decompositions tend to oversegment the shapes, they contain most of the boundaries present in the human annotations, suggesting that the model has similar criteria for decomposing objects into subparts; *e.g.*chair's legs are separated from the seat, wings and engines are separated from the airplane boundary, pistol trigger is separated from the rest, *etc*

4.2 Method

4.2.1 Approximate Convex Decomposition

In this subsection, we provide an overview of the shape decomposition approach used to generate the training data for our self-supervised task. A detailed description of the method used in this work can be found in [114].

Decomposing complex shapes as sets of convex components is a well studied problem [85, 107, 114, 225]. Given a polyhedron P, the goal is to compute the smallest set of convex polyhedra $\mathcal{C} = \{C_k | k = 1, ..., N\}$, such that the union $\cup_{k=1}^{k=N} C_i$ corresponds to P. However, exact convex decomposition of polyhedra is an NP-Hard problem [11] and leads to decompositions containing too many components, rendering it impractical for use in most applications (ours included). This can be circumvented by Approximate Convex Decomposition (ACD) techniques. ACD relaxes the convexity constraint of exact convex decomposition by allowing every component to be *approximately convex* up to a concavity ϵ. The way concavity is computed and how the components are split varies according to different methods [59, 85, 107, 114, 225]. In this work, we use an approach called Volumetric Hierarchical Approximate Convex Decomposition (V-HACD) [114]. The reasons for utilizing this approach are three-fold. First, as the name suggests, V-HACD performs computations using volumetric representations, which can be easily computed from dense point cloud data or meshes and lead to good results without having to resort to costly decimation and remeshing procedures. Second, the procedure is reasonably fast and can be applied to open surfaces of arbitrary genus. Third, V-HACD guarantees that no two components overlap, which means that there is no part of the surface that is approximated by more than one component. In the next paragraph, we describe V-HACD in detail.

V-HACD. Since the method operates on volumetric representations, the first step is to convert a shape into an occupancy grid. If the shape is represented as a point cloud, one can compute an occupancy grid by selecting which cells are occupied by the points and filling its interior. In our case, since our training shapes are from ShapeNet [24] which come with meshes, we chose to compute the occupancy grid by voxelizing the meshes using [78]. Once the voxelization is computed, the algorithm proceeds on computing convex components by recursively splitting the volume into two parts. First, the volume is centered and aligned in the coordinate system according to its principal axis. Then, one of the three axis aligned planes is selected as a splitting plane that separates the volume in two different parts. This procedure is applied multiple times until we reach the maximum number of desired components or the concavity tolerance is reached. The concavity $\eta(\mathcal{C})$ of a set of components $\mathcal{C}$ is computed as follows:

$$\eta(\mathcal{C}) = \max_{k=1,\dots N} d(C_k, CH(C_k)) \tag{4.1}$$

where $d(X, Y)$ is the difference between the volumes X and Y; $CH(X)$ is the convex hull of X; and C_k is the kth element of the set $\mathcal{C}$. The splitting plane selection happens by choosing one of the axis aligned planes which minimizes an energy $E(K, \mathbf{p})$, where K is the volume we are aiming to split and $\mathbf{p}$ is the splitting plane. This energy is defined as:

$$E(K, \mathbf{p}) = E_{con}(K, \mathbf{p}) + \alpha E_{bal}(K, \mathbf{p}) + \beta E_{sym}(K, \mathbf{p}), \tag{4.2}$$

where E_{con} is the connectivity component, which measures the sum of the normalized concavities between both sides of volume; E_{bal} is the balance component, which measures the dissimilarity between both sides; and E_{sym} is the symmetry component, which penalizes planes that are orthogonal to a potential revolution axis. α and β are weights for the last two terms. In all our experiments we used the default values of $\alpha = \beta = 0.05$. We refer the reader to [114] for a detailed description of the components in the energy term.

Assigning component labels to point clouds. The output of ACD for every shape is a set of convex components represented by convex meshes. For each shape, we sample points on the original ShapeNet mesh and on the mesh of every ACD component. We then propagate component labels to every point in the original point cloud by using nearest neighbor matching with points in the decomposition. More precisely, given an unlabeled point cloud $\{p_i\}_{i=1}^N$, this assigns a component label $\Gamma(p_i, \mathcal{C})$ to each point p_i via

$$\Gamma(p_i, \mathcal{C}) = \underset{k=1\ldots|\mathcal{C}|}{\operatorname{argmin}} \left[\min_{p_j \in C_k} ||p_i - p_j|| \right]. \tag{4.3}$$

4.2.2 Self-supervision with ACD

The component labels generated by the ACD algorithm are not consistent across point clouds, *i.e.*"component 5" may refer to the *seat* of a chair in one point cloud, while the *leg* of the chair may be labeled as "component 5" in another point cloud. Therefore, the usual cross-entropy loss, which is generally used to train networks for tasks such as semantic part labeling, is not applicable in our setting. We formulate the learning of Approximate Convex Decompositions as a metric learning problem on point embeddings via a *pairwise* or *contrastive loss* [69].

We assume that each point $p_i = (x_i, y_i, z_i)$ in a point cloud $\mathbf{x}$ is encoded as $\Phi(\mathbf{x})_i$ in some embedding space by a neural network encoder $\Phi(\cdot)$, *e.g.*PointNet [169] or PointNet++ [141]. Let the embeddings of a pair of points from a shape be $\Phi(\mathbf{x})_i$ and $\Phi(\mathbf{x})_j$, normalized to unit length (*i.e.*$||\Phi(\mathbf{x})_i|| = 1$), and the set of convex components as described above be $\mathcal{C}$. The pairwise loss is then defined as

$$\mathcal{L}^{pair}(\mathbf{x}, p_i, p_j, \mathcal{C}) = \begin{cases} 1 - \Phi(\mathbf{x})_i^\top \Phi(\mathbf{x})_j, & \text{if } [\Gamma(p_i, \mathcal{C}) = \Gamma(p_j, \mathcal{C})] \\ \max(0, \Phi(\mathbf{x})_i^\top \Phi(\mathbf{x})_j - m), & \text{if } [\Gamma(p_i, \mathcal{C}) \neq \Gamma(p_j, \mathcal{C})]. \end{cases} \tag{4.4}$$

This loss encourages points belonging to the same component to have a high similarity $\Phi(\mathbf{x})_i^\top \Phi(\mathbf{x})_j$, and encourages points from different components to have low similarity, subject to a margin m. $[\cdot]$ denotes the Iverson bracket.

Joint training with ACD. Formally, let us consider samples $\mathcal{X} = \{\mathbf{x}_i\}_{i \in [n]}$, divided into two parts: $\mathcal{X}^\mathcal{L}$ and $\mathcal{X}^\mathcal{U}$ of sizes l and u respectively. Now $\mathcal{X}^\mathcal{L} := \{\mathbf{x}_1, ..., \mathbf{x}_l\}$ consist of point clouds that are provided with human-annotated labels $\mathcal{Y}^\mathcal{L} := \{\mathbf{y}_1, ..., \mathbf{y}_l\}$, while we do not know the labels of the samples $\mathcal{X}^\mathcal{U} := \{\mathbf{x}_{l+1}, ..., \mathbf{x}_{l+u}\}$. By running ACD on the samples in $\mathcal{X}^\mathcal{U}$, we can obtain a set of components for each shape. The pairwise contrastive loss $\mathcal{L}^{pair}$ (Eq. 4.4) can then be defined over $\mathbf{x}_i \in \mathcal{X}^\mathcal{U}$ as a self-supervised objective. For the samples $\mathbf{x}_i \in \mathcal{X}^\mathcal{L}$, we have access to their ground-truth labels $\mathcal{Y}^\mathcal{L}$, which may for example, be semantic part labels. In that case, the standard choice of training objective is the *cross-entropy loss* $\mathcal{L}^{CE}$, defined over the points in an input point cloud. Thus, we can train a network on both $\mathcal{X}^\mathcal{L}$ and $\mathcal{X}^\mathcal{U}$ via a *joint loss* that combines both the supervised ($\mathcal{L}^{CE}$) and self-supervised ($\mathcal{L}^{pair}$) objectives,

$$\mathcal{L} = \mathcal{L}^{CE} + \lambda \cdot \mathcal{L}^{pair}. \tag{4.5}$$

The scalar hyper-parameter λ controls the relative strength between the supervised and self-supervised training signals. In the ***pretraining*** scenario, when we *only* have the unlabeled dataset $\mathcal{X}^{\mathcal{U}}$ available, we can train a neural network purely on the ACD parts by optimizing the $\mathcal{L}^{pair}$ objective.

4.3 Experiments

We demonstrate the effectiveness of the ACD self-supervision across a range of experimental scenarios. For all the experiments in this section we use ACDs computed on all shapes from the ShapeNetCore data [24], which contains 57,447 shapes across 55 categories. The decomposition was computed using a concavity tolerance of 1.5×10^{-3} and a volumetric grid of resolution 128^3. All the other parameters are set to their default values according to a publicly available implementation[1] of [114]. The resulting decompositions have an average of 17 parts per shape.

4.3.1 Shape classification on ModelNet

In this set of experiments, we show that the representations learned by a network trained on ACD are useful for discriminative downstream tasks such as classifying point clouds into shape categories.

Dataset. We report results on the ModelNet40 shape classification benchmark, which consists of 12,311 shapes from 40 shape categories in a train/test split of 9,843/2,468. A linear SVM is trained on the features extracted on the training set of ModelNet40. This setup mirrors other approaches for unsupervised learning on point clouds, such as FoldingNet [214] and Hassani *et al.* [72].

Experimental setup. A PointNet++ network is trained on the unlabeled ShapeNet-Core data using the pairwise contrastive loss on the ACD task, using the Adam

[1]`https://github.com/kmammou/v-hacd`

optimizer, initial learning rate of 1e-3 and halving the learning rate every epoch. However, this network architecture creates an embedding for each of the N points in an input shape, while for the shape classification task we require a single global descriptor for the entire point cloud. Therefore, we aggregate the per-point features of PointNet++ at the first two set aggregation layers (`SA1` and `SA2`) and the last fully connected layer (`fc`), resulting in 128, 256 and 128 dimensional feature vectors, respectively. Since features from different layers may have different scales, we normalize each vector to unit length before concatenating them, and apply element-wise signed square-rooting [152], resulting in a final 512-dim descriptor for each point cloud. The results are presented in Table 4.1.

Comparison with baselines. As an initial naïve baseline, we use a PointNet++ network with random weights as our feature extractor, and then perform the usual SVM training. This gives 78% accuracy on ModelNet40 – while surprisingly good, the performance is not entirely unexpected: randomly initialized convolutional neural networks are known to provide useful features by virtue of their architecture, as studied in Saxe *et al.* [154]. Training this network with ACD, on the other hand, gives a significant boost to performance (78% $\rightarrow$ **89.1**%), demonstrating the effectiveness of our proposed self-supervision task. This indicates some degree of generalization across datasets and tasks – from distinguishing convex components on ShapeNet to classifying shapes on ModelNet40. Inspired by [72], we also investigated if adding a reconstruction component to the loss would further improve accuracy. Reconstruction is done by simply adding an AtlasNet [68] decoder to our model and using Chamfer distance as reconstruction loss. Without the reconstruction term (i.e. trained only to perform ACD using contrastive loss), our result accuracy (89.1%) is the same as the multi-task learning approach presented in [72]. After adding a reconstruction term, we achieve an improved accuracy of **89.8**%. On the other hand, having just reconstruction without ACD yields an accuracy of 86.2%. This shows not only that

Table 4.1: Unsupervised shape classification on the ModelNet40 dataset. The representations learned in the intermediate layers by a network trained for the ACD task on ShapeNet data are general enough to be useful for discriminating between shape categories on ModelNet40.

Method	**Accuracy (%)**
VConv-DAE [157]	75.5
3D-GAN [204]	83.3
Latent-GAN [2]	85.7
MRTNet [55]	86.4
PointFlow [212]	86.8
FoldingNet [214]	88.4
PointCapsNet [221]	88.9
Multi-task [72]	89.1
Our baseline (with Random weights)	78.0
With reconstruction term only	86.2
Ours with ACD	89.1
Ours with ACD + Reconstruction	**89.8**

ACD is a useful task when learning representations for shape classification, but that it can also be combined with shape reconstruction to yield even better results.

Comparison with previous work. Approaches for *unsupervised* or *self-supervised* learning on point clouds are listed in the upper portion of Table 4.1. Our method achieves **89.1%** classification accuracy from purely using the ACD loss, which is met only by the unsupervised multi-task learning method of Hassani *et al.* [72]. We note that our method merely adds a contrastive loss to a standard architecture (PointNet++), without requiring a custom architecture and multiple pretext tasks as in [72], which uses clustering, pseudo-labeling and reconstruction.

4.3.2 Few-shot segmentation on ShapeNet

Dataset. We report results on the **ShapeNetSeg** part segmentation benchmark [24], which is a subset of the ShapeNetCore database with manual annotations (train/val/test splits of 12,149/1,858/2,874). It consists of 16 man-made shape categories such as

Table 4.2: Few-shot segmentation on the ShapeNet dataset (*class avg. IoU* over 5 rounds). K denotes the number of shots or samples per class for each of the 16 ShapeNet categories used for supervised training. Jointly training with the ACD task reduces overfitting when labeled data is scarce, leading to significantly better performance over a purely supervised baseline.

Samples/cls.	**k=1**	**k=3**	**k=5**	**k=10**
Baseline	53.15 ± 2.49	59.54 ± 1.49	68.14 ± 0.90	71.32 ± 0.52
w/ ACD	61.52 ± 2.19	69.33 ± 2.85	72.30 ± 1.80	74.12 ± 1.17
	k=20	**k=50**	**k=100**	**k=inf**
Baseline	75.22 ± 0.82	78.79 ± 0.44	79.67 ± 0.33	81.40 ± 0.44
w/ ACD	76.19 ± 1.18	78.67 ± 0.72	78.76 ± 0.61	81.57 ± 0.68

airplanes, chairs, and tables, with manually labeled semantic parts (50 in total), such as wings, tails, and engines for airplanes; legs, backs, and seats for chairs, and so on. Given a point cloud at test time, the goal is to assign each point its correct part label out of the 50 possible parts. Few-shot learning tasks are typically described in terms of "n-way k-shot" – the task is to discriminate among n classes and k samples per class are provided as training data. We modify this approach to our setup as follows – we select k samples from each of the $n = 16$ shape categories as the labeled training data, while the task remains semantic part labeling over the 50 part categories.

Experimental setup. For this task, we perform *joint training* with two losses – the usual cross-entropy loss over labeled parts for the training samples from ShapeNetSeg, and an additional contrastive loss over the ACD components for the samples from ShapeNetCore (Eq. 4.5), setting $\lambda = 10$. In our initial experiments, we found joint training to be more helpful than pre-training on ACD and then fine-tuning on the few-shot task (an empirical phenomenon also noted in [209]), and thereafter consistently used joint training for the few-shot experiments. All overlapping point clouds between the human-annotated ShapeNetSeg and the unlabeled ShapeNetCore were removed from the self-supervised training set. The (x, y, z) coordinates of the points in each

Table 4.3: Comparison with state-of-the-art semi-supervised part segmentation methods on ShapeNet. Performance is evaluated using *instance-averaged IoU*.

Method	1% labeled **IoU**	5% labeled **IoU**
SO-Net [103]	64.0	69.0
PointCapsNet [221]	67.0	70.0
MortonNet [177]	-	77.1
Multi-task [72]	68.2	77.7
ACD (*ours*)	**75.7**	**79.7**

point cloud are used an the input to the neural network; we do not include any additional information such as normals or category labels in these experiments.

Comparison with baselines. Table 4.2 shows the few-shot segmentation performance of our method, versus a fully-supervised baseline. Especially in the cases of very few labeled training samples ($k = 1, \ldots, 10$), having the ACD loss over a large unlabeled dataset provides a consistent and significant gain in performance over purely training on the labeled samples. As larger amounts of labeled training samples are made available, naturally there is limited benefit from the additional self-supervised loss – *e.g.*when using all the labeled data, our method is within standard deviation of the purely supervised baseline. Qualitative results are shown in Fig. 4.3.

Comparison with previous work. The performance of recent *unsupervised* and *self-supervised* methods on ShapeNet segmentation are listed in Table 4.3. Consistent with the protocol followed by the multi-task learning approach of Hassani *et al.* [72], we provide 1% and 5% of the training samples of ShapeNetSeg as the labeled data and report instance-average IoU. Our method clearly outperforms the state-of-the-art unsupervised learning approaches, improving over [72] at both the 1% and 5% settings (68.2 $\rightarrow$ **75.7**% and 77.7 $\rightarrow$ **79.7**%, respectively).

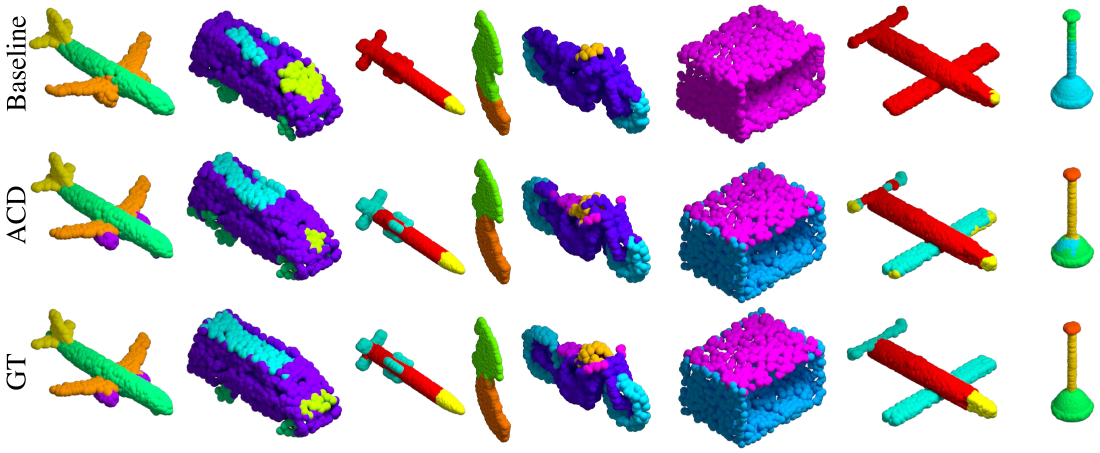


Figure 4.3: Qualitative comparison on 5-shot ShapeNet [24] part segmentation. The baseline method in the first row corresponds to training using only 5 examples per class, whereas the ACD results in the second row were computed by performing joint training (cross-entropy from 5 examples + contrastive loss over ACD components from ShapeNetCore). The network backbone architecture is the same for both approaches – PointNet++ [141]. The baseline method merges parts that should be separated, *e.g.*engines of the airplane, details of the rocket, top of the table, seat of the motorcycle, etc.

4.3.3 Analysis of ACD

On the effect of backbone architectures. Differently from [30, 72, 214], the ACD self-supervision does not require any custom network design and should be easily applicable across various backbone architectures. To this end, we use two recent high-performing models – *PointNet++* (with multi-scale grouping [141]) and *DGCNN* [198] – as the backbones, reporting results on ModelNet40 shape classification and few-shot segmentation ($k = 5$) on ShapeNetSeg (Table 4.4). On shape classification, both networks show large gains from ACD pre-training: 11% for PointNet++ (as reported earlier) and 14% for DGCNN. On few-shot segmentation with 5 samples per category (16 shape categories), PointNet++ improves from 68.14% IoU to 72.3% with the inclusion of the ACD loss. The baseline DGCNN performance with only 5 labeled samples per class is relatively lower (64.14%), however with the additional ACD loss

Table 4.4: Comparing embeddings from PointNet++ [141] and DGCNN [198] backbones: shape classification accuracy on ModelNet40 (*Class./MN40*) and few-shot part segmentation performance in terms of class-averaged IoU on ShapeNet (*Part Seg./ShapeNet*).

Task / Dataset	Method	PointNet++	DGCNN
Class./MN40	Baseline	77.96	74.11
	w/ ACD	**89.06**	**88.21**
5-shot Seg./ShapeNet	Baseline	68.14 ± 0.90	64.14 ± 1.43
	w/ ACD	**72.30** ± 1.80	**73.11** ± 0.95

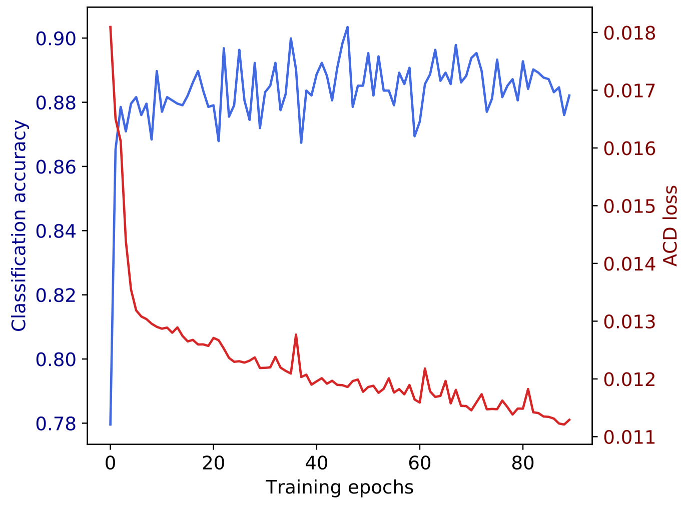


Figure 4.4: Classification accuracy of a linear SVM on the ModelNet40 *validation set* v.s. the ACD *validation loss* over training epochs.

on unlabeled samples, the model achieves 73.11% IoU, which is comparable to the corresponding PointNet++ performance (72.30%).

On the role of ACD in shape classification. Fig. 4.4 shows the reduction in validation loss on learning ACD (red curve) as training progresses on the unlabeled ShapeNet data. Note that doing well on ACD (in terms of the validation loss) also leads to learning representations that are useful for the downstream tasks of shape classification (in terms of SVM accuracy on a validation subset of ModelNet40 data, shown in blue).

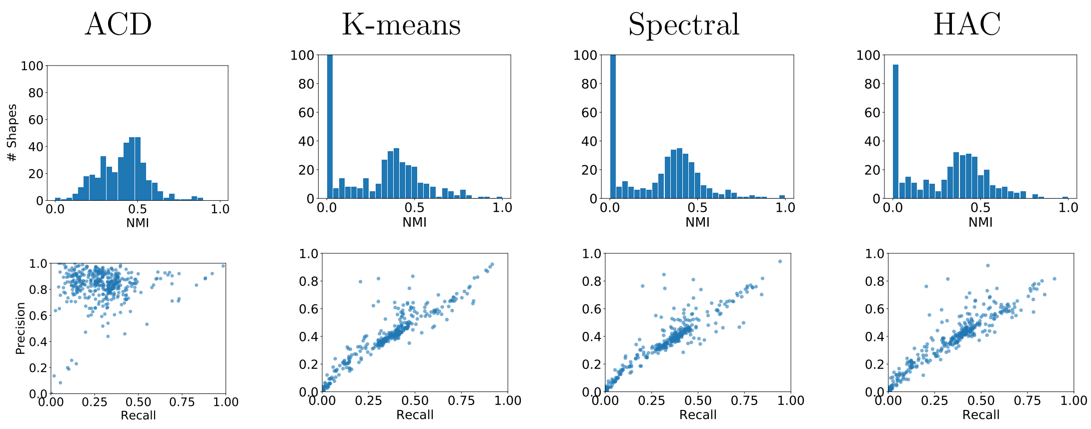


Figure 4.5: Correspondence between human part labels and shape decompositions: comparing ACD with basic clustering algorithms – K-means, spectral clustering and hierarchical agglomerative clustering (HAC). ***Row-1:*** histogram of *normalized mutual information* (NMI) between human labels and clustering – ACD is closer to the ground-truth parts than others (y-axes clipped at 100 for clarity). ***Row-2:*** plotting *precision v.s. recall* for each input shape, ACD has high precision and moderate recall (tendency to over-segment parts), while other methods are usually lower in both metrics.

However, the correlation between the two quantities is not very strong (Pearson $\rho = 0.667$) – from the plots it appears that after the initial epochs, where we observe a large gain in classification accuracy as well as a large reduction in ACD loss, continuing to be better at the pretext task does not lead to any noticeable gains in the ability to classify shapes: training with ACD gives the model some useful notion of grouping and parts, but it is not intuitively obvious if *perfectly* mimicking ACD will improve representations for classifying point-clouds into shape categories.

Comparison with clustering algorithms. We quantitatively analyse the connection between convex decompositions and semantic object parts by comparing ACD with human part annotations on 400 shapes from ShapeNet, along with simple clustering baselines – K-means [6], spectral clustering [161, 192] and hierarchical agglomerative clustering (HAC) [123] on (x, y, z) coordinates of the point clouds. For the baselines, we set the number of clusters to be the number of ground-truth parts in each shape. For each sample shape, given the set of M part categories $\Omega = \{\omega_1, \omega_2, \ldots \omega_M\}$

and the set of N clusters $\mathcal{C} = \{C_1, C_2, \dots C_N\}$, clustering performance is evaluated using *normalized mutual information* (NMI) [191], defined as

$$\text{NMI}(\Omega, \mathcal{C}) = \frac{I(\Omega; \mathcal{C})}{[H(\Omega) + H(\mathcal{C})]/2}, \tag{4.6}$$

where $I(\cdot;\cdot)$ denotes the mutual information between classes Ω and clusters $\mathcal{C}$, and $H(\cdot)$ is the entropy [38]. A better clustering results in *higher* NMI w.r.t. the ground-truth part labels. The first row of Fig. 4.5 shows the histograms of NMI between cluster assignments and human part annotations: ACD, though not exactly aligned to human notions of parts, is significantly better than other clustering methods, which have very low NMI in most cases.

We plot the *precision* and *recall* of clustering for each of the 400 shapes on the second row of Fig. 4.5. The other baseline methods show that a naïve clustering of points does not correspond well to semantic parts. ACD has high precision and moderate recall on most of the shapes – this agrees with the visual impression that though ACD tends to oversegment the shapes, the decompositions contain most of the boundaries present in the human annotations. For example, ACD typically segments the legs of a chair into four separate components. Part annotations on ShapeNet however label all the legs of a chair with the same label, since the benchmark does not distinguish between the individual legs of a chair. We note that the correspondence of ACD to human part labels is not perfect, and this opens an interesting avenue for further work – exploring other decomposition methods like generalized cylinders [223] that may correspond more closely to human-defined parts, and in turn could lead to improved downstream performance on discriminative tasks.

4.4 Conclusions

Self-supervision using approximate convex decompositions (ACD) has been shown to be effective across multiple tasks and datasets – few-shot part segmentation on

ShapeNet and shape classification on ModelNet, consistently surpassing existing self-supervised and unsupervised methods in performance. A simple pairwise contrastive loss is sufficient for introducing the ACD task into a network training framework, without dependencies on any custom architectures or losses.

The method can be easily integrated into existing state-of-the-art architectures operating on point clouds such as PointNet++ and DGCNN, yielding significant improvements in both cases. Extensive ablations and analyses are presented on the approach, helping us develop a better intuition about the method. Given the demonstrated effectiveness of ACD in self-supervision, this opens the door to incorporating other shape decomposition methods from the classical geometry processing literature into deep neural network based models operating on point clouds.

In the next chapters, instead of dealing with the lack of annotated 3D shapes, we expand our focus to explore scenarios where 3D data itself is missing. More specifically, the next chapter focus on priors induced by networks representing manifold parametrizations whereas the following ones analyze volumetric occupancy grids and differentiable operators that allow learning 3D directly from images (with different levels of annotation).

CHAPTER 5
DEEP MANIFOLD PRIOR

The goal of this chapter is to characterize how the choice of the network architecture impacts the properties of the low-dimensional manifolds parametrized by neural networks. We present and analyze a *deep manifold prior*, an approach to represent a manifold as a collection of transformations (atlas) of an Euclidean space parameterized using deep networks (Section 5.2). We show that random networks induce smooth surfaces whose limiting behavior can be understood in terms of a Gaussian process (GP) [33, 126, 201]. We analyze how the different network architectures affect the distribution of position, normals and curvature of surfaces (Section 5.3). We also derive the properties of implicit surfaces induced by the level-set of a scalar field $\{f(x) = c\}$ parameterized using a deep network.

As a concrete application we study the problem of interpolating and denoising point clouds sampled from contours or surfaces of shapes, as seen in Figures 5.1 and 5.2. The manifold parametrization allows us to efficiently sample point clouds, which can be combined with a Chamfer metric to measure a reconstruction error with respect to the sampled data. We show that smooth surfaces are obtained when the parameters of the networks are learned to minimize the reconstruction error starting from a *random initialization* (Figure 5.2). The approach is also effective for the level-set formulation, where the objective is to learn a deep network that correctly classifies points as *inside* or *outside* the surface. However, an advantage of the explicit parametrization is that it does not require the notion of what is inside. In addition we introduce a *regularization* that reduces self-intersections, overlaps, and distortion

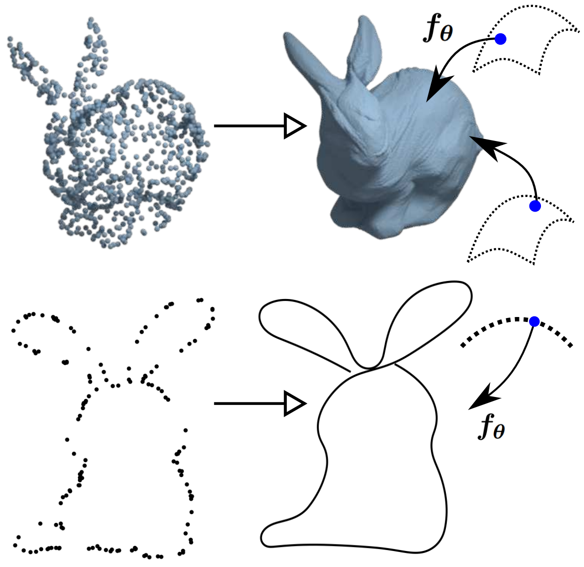
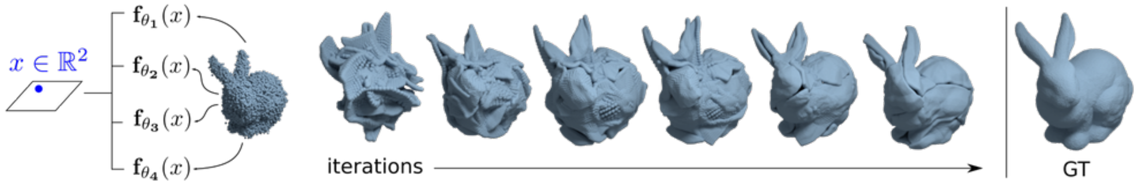


Figure 5.1: Deep manifold prior. Points interpolated by using deep networks to map points in a 2D grid (top) and 1D grid (bottom) to the target shape (a 3D surface and a 2D curve respectively). The networks are randomly initialized and trained to minimize the Chamfer distance to the target.

of the parametrization, which is desirable for applications such as texture mapping (Section 5.2). Our approach requires *no* prior learning, works across a range of 3D shapes, and outperforms strong baselines for point cloud denoising, such as Screened Poisson Surface Reconstruction (SPSR) and Robust Implicit Moving Least Squares (RIMLS). It is also more lightweight than approaches that operate on volumetric representations of 3D shapes (Section 5.4).

Our analysis sheds lights on the impressive performance of several recently proposed architectures for 3D surface generation, such as MRTNet [55], AtlasNet [68], FoldingNet [214], and Pixel2Mesh [193], as well as implicit surface approaches [31, 58, 119, 135]. These can be be interpreted as different ways of parameterizing a manifold. In particular, AtlasNet generates a 3D shape as a collection of surfaces, each represented as a transformation of a unit grid using a fully-connected network. However, the generated pieces exhibit significant overlap which results in a poor surface reconstruction and is less desirable for applying materials and textures to the sur-



$x \in \mathbb{R}^2$
$\mathbf{f}_{\theta_1}(x)$
$\mathbf{f}_{\theta_2}(x)$
$\mathbf{f}_{\theta_3}(x)$
$\mathbf{f}_{\theta_4}(x)$
iterations
GT

Figure 5.2: Manifold reconstruction pipeline. Manifold parametrizations are encoded by neural networks ($\mathbf{f}_{\theta_\mathrm{i}}$) and trained to minimize the reconstruction error with respect to the noisy target (left). Prior induced by the neural networks makes the generated surface much closer to the ground-truth (right), without ever seeing any additional training data.

face (Section 5.4). The proposed regularization alleviates this problem. Moreover, by replacing the fully-connected networks of AtlasNet with convolutional variants we improve the performance on standard benchmarks for shape generation [35] with networks that have a fraction of the parameters, faster inference time, as well as smaller memory footprint (Section 5.4).

5.1 Related Work

Manifold 3D shape generation 3D shape generation is an active area of research with methods that generate 3D shapes as volumetic representations such as occupancy grids [35,52,70,146,174,187,204], signed distance functions [31,58,119,135], mutliview depth and normals [108,111,166,173], or point clouds [2,49,51,55]. Our work is closely related to techniques for generating 3D shapes through a predefined connectivity or parametrization structure over the surface of the shape. Pixel2Mesh [193] utilizes graph convolutional networks to generate meshes that are homeomorphic to a sphere. AtlasNet [68] and FoldingNet [214] learn a parametrization of a surface by adopting deep networks to transform point coordinates in a 2D plane to the shape surface. Specifically, each point is generated as $\left(f_\theta^1(x), f_\theta^2(x), f_\theta^3(x)\right)$ where f_θ^i is a deep network and $x = (x_1, x_2)$ is a point in the unit grid. Alternate approaches [31,58,119,135] represent the surface as the level-set of a scalar field, $f(x) = 0, x \in \mathbb{R}^3$, e.g., of the signed distance function. While these have been applied for shape generation by training on

3D shape datasets, our goal is to analyze the role of these parameterizations as an *implicit prior* for manifold denoising and interpolation tasks.

Deep implicit priors Our work is related to the deep image prior [188] that generates images as a convolutional network transformation of a random signal on a unit grid. By optimizing the randomly initialized network to minimize a reconstruction loss with respect to the noisy target, their approach was shown to yield excellent denoising results. Our approach generalizes this idea to manifold data, which is more appropriate for interpolating and denoising contours and surfaces (see Figure 5.6 for a comparison). Our work is also related to the recently proposed deep geometric prior [202]. Their approach was used to estimate a surface from point cloud data by partitioning the surface into small overlapping patches and reconstructing the local manifold using a deep network. Consistency in the overlapping regions was enforced by minimizing the Earth Movers distance (EMD). In contrast to their work, we learn a small collection of non-overlapping parametrizations (atlas) by minimizing a regularized term and Chamfer distance, which is much more efficient than EMD. We also consider diverse tasks such as point cloud denoising, interpolation, and shape reconstruction across a category where the atlases needs to be consistent across instances. Finally, we present a theoretical analysis of the local properties of the generated surface by analyzing its limiting behavior as a Gaussian process.

Embedding a manifold Our work is related to techniques for embedding manifolds into a low-dimensional Euclidean space (*e.g.*, IsoMap [176] or LLE [149]). Our approach parameterizes the inverse mapping from the Euclidean space to the data manifold using a deep network. Interestingly, invertability can be guaranteed by using networks with easy to compute inverses (*e.g.*, NICE [41] or GLOW [95]). In computer graphics, a number of techniques have been developed for shape surface denoising and reconstruction. Screened Poisson Reconstruction [91] constructs an implicit surface on a 3D volumetric grid based on oriented point samples by solving the Poisson equation. Approaches based on Moving Least Squares [3,134,159] reconstruct a surface by estimating an approximation of each local patch, similar to the deep geometric prior [202] approach. Our approach outperforms these baselines by a significant margin (Table 5.1).

Deep networks and Gaussian processes A Gaussian process (GP) is commonly viewed as a prior over functions. Let T be an index set (*e.g..*, $T \in \mathbb{R}^d$), let $\mu(t)$ be a real-valued mean function and $K(t, t')$ be a non-negative definite kernel or covariance function on T. If $f \sim GP(\mu, K)$, then, for any finite number of indices $t_1, ..., t_n \in T$, the vector $(f(t_i))_{i=1}^n$ is Gaussian distributed with mean vector $(\mu(t_i))_{i=1}^n$ and covariance matrix $(K(t_i, t_j))_{i,j=1}^n$. Neal [126] showed that a two-layer network with infinite number of hidden units approaches a GP. The mean and covariance of commonly used non-linearities have been derived in several subsequent works [33,201]. We use this machinery to analyze the limiting GP of deep manifold priors.

5.2 Method

Background Our focus is to define priors over *manifolds*. We first introduce some basic notation. A *n-manifold* is a topological space $\mathcal{M}$ for which every point in $\mathcal{M}$ has a neighborhood homeomorphic to the Euclidean space $\mathbb{R}^n$. Let $\mathcal{U} \subset \mathcal{M}$ and $\mathcal{V} \subset \mathbb{R}^n$ be open sets. A homeomorphism $\phi : \mathcal{U} \to \mathcal{V}, \phi(u) = (x_1(u), x_2(u), ..., x_n(u))$ is a *coordinate system* on $\mathcal{U}$ and $x_1, x_2, ..., x_n$ are *coordinate functions.* The pair $\langle \mathcal{U}, \phi \rangle$ is a *chart*, whereas $\zeta = \phi^{-1}$ is a *parameterization* of $\mathcal{U}$. An *atlas* on $\mathcal{M}$ is a collection of charts $\{\mathcal{U}_\alpha, \phi_\alpha\}$ whose union covers $\mathcal{M}$. Intuitively, surfaces are 2-manifolds where as contours are 1-manifolds. Thus the dimensionality of the input of the parameterization or the output of the chart corresponds to the order n of the manifold. Atlases can be used to represent manifolds that cannot be decomposed using a single parametrization (*e.g.*, the surface of a sphere can be diffeomorphically mapped to two planes but not one.)

General framework In our work we will replace the search over $\mathcal{U}$ by a search over the parameters θ of the DNN f_θ that encodes the parameterization $f_\theta = \zeta = \phi^{-1}$. More specifically, given a set of points $P \in \mathcal{M}$, we aim to recover the manifold $\mathcal{M}$ by computing the following:

$$\theta^* = \operatorname*{argmin}_\theta \mathcal{L}_C(f_{\theta, x\sim\mathbb{R}^n}(x), P). \tag{5.1}$$

The approximated manifold can then be reconstructed in the domain on which it is embedded f_{θ^*}. In practice, we restrict x to the unit hypercube $[0,1]^n$. Here $\mathcal{L}$ is a loss function that computes a discrepancy between sets. Thus, reconstructing a manifold represented by an atlas of k charts is done by computing the following:

$$\theta_1^*, \theta_2^*, ...\theta_k^* = \operatorname*{argmin}_{\theta_1,\theta_2,...\theta_k} \mathcal{L}_C(\bigcup_{i=1}^{k} f_{\theta_i}(x), P) \tag{5.2}$$

Parameterization We explore two choices of parameterizations of the coordinate function $f_\theta(x)$ as a deep neural network. The first uses a multi-layer perception (MLP) to represent the parameterization explicitly: the network receives as an input a value $x \in \mathbb{R}^n$ and outputs the coordinates of point in the manifold. We use ReLU non-linearities throughout the network, except for the last layer where we use tanh. This representation is analogous to the ones used in [68, 214]. The second choice is to encode $\mathcal{M}$ directly through a convolutional network $g(z)$, where z is a stationary signal (Gaussian noise). We use 2D convolutional layers followed by ReLU activations and bilinear upsampling, except for the last layer where we use tanh. The convolutional parametrization induces a stationary prior (see Supplementary for details), and we observe the resulting architectures are more memory-efficient and compact than the first choice.

Loss function A key part of our method is computing a distance between two sets of points P_1 and P_2. Such distance metric needs to be differentiable and reasonably efficient to compute, since the cardinality of the sets might be large. Thus, similarly to previous work [55, 68, 193, 214], we employ the Chamfer distance $\mathcal{L}_C$ defined as follows:

$$\mathcal{L}_C(P_1, P_2) = \sum_{p_1 \in P_1} \min_{p_2 \in P_2} \|p_1 - p_2\|_2^2 + \sum_{p_2 \in P_2} \min_{p_1 \in P_1} \|p_1 - p_2\|_2^2 .$$

Stretch regularization Representing the manifold as a set of multiple parameterizations output by DNNs has some drawbacks. First, there is no guarantee that the charts are invertible, which means that a surface generated by f_θ might contain self-intersections. Second, multiple charts might be representing the same region of the manifold. In theory this is not a problem as long as overlapping regions are consistent. However, in practice this consistency is hard to achieve when point clouds are sparse and noisy. We propose to alleviate those issues by penalizing the stretch of the computed parameterization. Let $\mathcal{N}(w)$ be the neighborhood of w in $\mathbb{R}^n$, the *stretch regularization* $\mathcal{L}_S$ can be defined as follows:

$$\mathcal{L}_S(\theta) = \mathbb{E}_{x \sim [0,1]^n} \left[\sum_{x' \in \mathcal{N}(x)} \|f_\theta(x) - f_\theta(x')\|_2^2 \right] . \tag{5.3}$$

Notice that we can compute the neighbors of x ahead of time which makes the computation significantly cheaper. In practice, we sample x from a set of predefined regularly spaced values in $[0, 1]$ – a regular grid in the 2D case. Now we can define our full loss function as follows.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_C(\boldsymbol{f}_{\boldsymbol{\theta}, x \sim \mathbb{R}^n}(x), P) + \lambda \mathcal{L}_S(\boldsymbol{\theta}), \tag{5.4}$$

where $\boldsymbol{\theta} = \theta_1, \theta_2, ...\theta_k$ and $\boldsymbol{f}_{\boldsymbol{\theta}}(x) = \bigcup_{i=1}^{k} f_{\theta_i}(x)$.

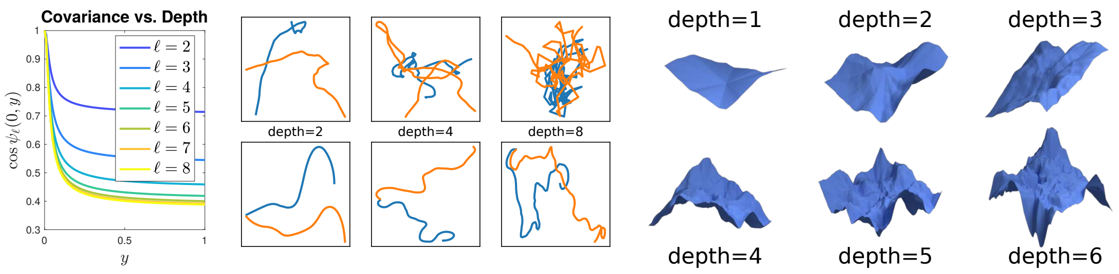


Figure 5.3: Characterizing the deep manifold prior. (left) a plot demonstrating the relationship between the network depth and the covariance function for the limiting GP. **(middle)** Random curves generated by the coordinate (top rows) and arc-length (bottom rows) parametrizations using deep networks with varying depths. **(right)** Random surfaces generated by deep networks of varying depths.

Manifolds as deep level-sets An alternative approach is to represent d-manifold as the level-set of a scalar function over $d+1$ dimensions. For example, a surface can be represented as the level set, $f(x) = 0$, where $x \in \mathbb{R}^3$. Prior work [31,58,119,135] has explored this approach to generate a 3D surface by approximating its signed distance function. Level-set formulation can naturally handle shapes with different topologies, but require the knowledge of what is inside the surface, which can be challenging to estimate for imperfect point-cloud data. In this work, we also characterize and experiment with the manifold prior induced by the level-set of a deep network $f_\theta(x) = 0$ initialized randomly.

5.3 Limiting GP for the Deep Manifold Prior

Consider the case when the manifold coordinates are parametrized using a deep network $f_\theta(x)$. We show that random networks, *e.g.*, whose parameters are drawn i.i.d. from a Gaussian distribution, produces smooth manifolds. This is done by analyzing the limiting behavior of the function as a Gaussian process. In practice this is a good approximation to networks that are relatively shallow and have hundreds of hidden units in each layer.

Concretely, the mean $\mathbb{E}_\theta[f_\theta(x)]$ and covariance $\mathbb{E}_\theta[f_\theta(x)f_\theta(y)^T]$ of the parameterization characterize the structure of the generated manifold. For example, the covariance function of a smooth manifold decays slowly as a function of distance in the input space compared to a rough one. Following prior work [33,126,201], we first derive the mean and covariance for a two layer network with a scalar output. We then generalize the analysis to vector outputs and multi-layer networks.

Consider a two-layer fully-connected network on an input $x \in \mathbb{R}^n$. Let H be the number of units in the hidden layer represented using parameters $U = (u_1, u_2, \dots u_H)$ where $u_j \in \mathbb{R}^n$ and the second layer has one output parameterized by weights $v \in \mathbb{R}^H$. Denote the non-linearity applied to each unit as the scalar function $h(\cdot)$. The output of the network is: $f(x) = \sum_{k=1}^{H} v_k h(u_k^T x)$. When the parameters U and v are drawn from a Gaussian distributions $N(0, \sigma_u^2 \mathbb{I})$ and $N(0, \sigma_v^2 \mathbb{I})$ respectively, we have:

$$\mathbb{E}_{U,v}[f(x)] = \mathbb{E}_{U,v}\left[\sum_{k=1}^{H} v_k h\left(u_k^T x\right)\right] = 0,$$

since U and v are independent and zero mean. Similarly, the covariance function $K(x,y)$ can be shown to be:

$$K(x,y) = \mathbb{E}_{U,v}[f(x)f(y)] = H\sigma_v^2 \mathbb{E}_U\left[h\left(u_k^T x\right) h\left(u_k^T y\right)\right].$$

This follows since each u_k is drawn i.i.d, each v_k is independent and drawn identically from a Gaussian distribution with zero mean. The quantity $V(x,y) = E_u\left[h(u^T x)h(u^T y)\right]$ can be computed analytically for various transfer functions. Williams [201] showed that when $h(t) = \mathrm{erf}(t) = 2/\sqrt{\pi}\int_0^t e^{-t^2} dt$, then

$$V_{\mathrm{erf}}(x,y) = \frac{2}{\pi}\sin^{-1}\frac{x^T \Sigma y}{\sqrt{(x^T \Sigma x)\,(y^T \Sigma y)}}. \tag{5.5}$$

Here $\Sigma = \sigma^2\mathbb{I}$ is the covariance of u. For the ReLU non-linearity $h(t) = \max(0, t)$, Cho and Saul [33] derived the expectation as:

$$V_{\text{relu}}(x, y) = \frac{1}{\pi}\|x\|\|y\| \left(\sin\psi + (\pi - \psi)\cos\psi\right), \tag{5.6}$$

where $\psi = \cos^{-1}\left(\frac{x^T y}{\|x\|\|y\|}\right)$. We refer the reader to [33, 201] for kernels corresponding of other transfer functions.

An application of the Central Limit Theorem shows that by letting σ_v^2 scale as $1/H$ and $H \to \infty$, the output of a two layer convolutional network converges to a Gaussian distribution with zero mean and covariance

$$K_1(x, y) = E_{U,v}\left[f(x)f(y)\right] = V\left(x, y\right). \tag{5.7}$$

Hence the limiting behavior of the DNN can be approximated as a Gaussian process with a zero mean and covariance function $K(x, y) = V(x, y)$.

Extending to multiple outputs The above analysis can be extended to the case when the function $f(x)$ is vector valued. For example a 2-manifold in 3D can be represented as $f(x) = (f^1(x), f^2(x), f^3(x))$, with $x \in \mathbb{R}^2$. In our case, the functions share a common backbone and each $f^i(x)$ is constructed from the outputs of the last hidden layer parameterized with weights v^i, *i.e.*, $f^i(x) = \sum_{k=1}^{H} v_k^i h(u_k^T x)$. From the earlier analysis we have that each $f^i(x)$ has zero mean in expectation. And the covariance between dimension i and j of f is:

$$K_1^{i,j}(x, y) = \mathbb{E}_{U,v_i,v_j}\left[f^i(x)f^j(y)\right] = V\left(x, y\right)\mathbf{1}[i = j].$$

This follows from the fact that each v_k^i is independent and drawn from a zero mean distribution. Thus, the covariance is a diagonal matrix with entries $V(x, y)$ in its the diagonal.

Extending to multiple layers The analysis can be extended to multiple layers by recursively applying the formula for the two-layer network. Denote $K_\ell(x,y)$ as the covariance function of a scalar valued fully-connected network with $\ell+1$ layers and $J(\theta) = \sin\theta + (\pi - \theta)\cos\theta$. Following [33] for the ReLU non-linearity we have the following recursion:

$$K_{\ell+1}(x,y) = \frac{1}{\pi}\left(K_\ell(x,x)K_\ell(y,y)\right)^{1/2} J\left(\psi_\ell\right).$$

Where $\psi_\ell(x,y) = \cos^{-1}\frac{K_\ell(x,y)}{\sqrt{K_\ell(x,x)K_\ell(y,y)}}$ and $K_0(x,y) = x^T y$. Note that if in each layer we add a bias term sampled from a $N(0,\sigma_b^2)$ the covariance changes to $K_\ell(x,y)+\sigma_b^2$ and the mean remains unchanged at zero.

5.3.1 Discussion and Analysis

The above analysis shows that random networks induce certain priors over the coordinates of the manifold. The effect of increasing the depth of the network can be seen by visualizing how the covariance $\cos\psi_\ell(x,y)$ varies as a function of depth. Figure 5.3 plots $\cos\psi_\ell(x,y)$ at $x=0$ for a curve as a function of the depth of the network for $\sigma_b = 0.01$. The covariance decays faster with depth, indicating that the deeper networks produce manifolds with higher spatial frequencies (or curvatures). This can also be seen in Figure 5.3 which shows random curves (middle) from a surfaces (right) for networks with varying depths.

One potential drawback of fully-connected network parameterization is that the generated manifold does not have a stationary (translationally invariant) covariance function. A covariance function $K(x,y)$ is stationary if it can be written as $K(x,y) = k(x-y)$. On the other hand, a convolutional network that produces coordinates through a series of convolutional layers operating on a random noise has a stationary covariance [32]. This is identical to the approach for generating natural images in the deep image prior [188] and we explore this alternative in Section 5.4.2.

Normals and curvature While we have shown that the outputs $f(x)$ induced by random networks is a GP in the limit, what can be said about intrinsic properties such as normals and curvature? Consider the curve $\gamma(t) = (x(t), y(t))$. Since derivatives are linear operators, it follows that distribution of derivatives, $\dot{x}$ and $\dot{y}$, are also Gaussian [165]. The curvature is given by $\kappa = (\ddot{x}\dot{y} - \ddot{y}\dot{x})/(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}$. Unfortunately, since each of the derivatives converge to a zero mean Gaussian distribution, the limiting distribution of the curvature κ does not exist. The pathology arises because the parameterization has a speed ambiguity, *i.e.*, replacing t with any monotonic function of t results in the same curve. To avoid this one can directly parametrize the derivatives as $\dot{x} = \cos(f(t))$ and $\dot{y} = \sin(f(t))$ where f is a deep network. This is an arc-length (unit speed) parametrization since $\dot{x}^2 + \dot{y}^2 = 1$. Once the derivatives are generated, the curve can be reconstructed by integration, *i.e.*, $x = \int_0^t \cos(f(t))dt$. In this case the limiting distribution of the coordinates, normal, and curvature all exist and are also GPs. We derive the mean and covariance function in the Supplementary material. Figure 5.3-middle shows draws from the GP with direct (top) and arc-length (bottom) parametrizations. One can see that arc-length parametrizations lead to more length-uniform curves.

Unlike curves, it is much more challenging to design arc-length parametrizations of surfaces. The difficulty arises due to the fact the gradients need to satisfy additional constraints for the surface to be integrable [172]. Hence, we directly parameterized the coordinate function and proposed the stretch regularization to minimize distortion. Alternatives ways of parameterizing the surface to satisfy properties such as conformality [127] is left for future work.

Deep level-set prior Finally, the GP analysis applies in a straightforward manner to the level-set formation $f_\theta(x) = 0$ where f_θ is a ReLU network mapping the 3D position $x \in \mathbb{R}^3$ to a scalar. The induced distribution over the scalar field is a GP for random networks. Since for a differentiable function f with non-zero gradient, the

	Surface	Contour	Implicit	RIMLS [134]	SPSR [91]
bunny	**2.71E-04**	6.64E-04	5.52E-04	1.43E-03	3.96E-04
dragon	**4.18E-04**	6.12E-04	1.20E-03	1.65E-03	1.46E-02
car	**2.73E-04**	4.57E-04	6.83E-02	1.50E-03	2.10E-03
cup	**2.59E-04**	5.80E-04	2.64E-02	1.74E-03	1.00E-02
mobius	**3.51E-04**	4.95E-04	3.26E-03	1.96E-03	1.89E-02
chair	**3.95E-04**	4.22E-04	7.32E-03	2.09E-03	2.58E-02
spiral	1.05E-03	**7.31E-04**	1.64E-02	2.98E-03	7.90E-02
ring	5.69E-04	**5.54E-04**	4.81E-02	2.46E-03	3.76E-02
avg.	**4.48E-04**	5.65E-04	2.13E-02	1.98E-03	2.36E-02

Table 5.1: Quantitative results for point cloud denoising. *Surface*, *Contour* and *Implicit* represent different *deep manifold priors* based on a 2-manifold, 1-manifold and level-set paramertization.

	S1R	S8R	S1	S8	-	-
avg.	4.48E-03	**4.48E-04**	2.75E-03	1.35E-03	-	-
	C1R	C8R	C1	C8	RIMLS [134]	SPSR [91]
avg.	1.08E-03	5.77E-04	1.00E-03	5.82E-04	1.98E-03	2.36E-02

Table 5.2: Ablation studies. Comparison between different variations of our approach. Naming follows the following convention: S corresponds to a 2-manifold parameterization (surface), whereas C corresponds to a 1-manifold (contour). The following number (1 or 8) corresponds to the number of parameterizations. A R letter is added if stretch regularization was used ($\lambda = 1.0$).

gradient is orthogonal to the level set, one can characterize the surface by analyzing the gradient field ∇f. The limiting distribution over the gradient field is also a GP and one can estimate the mean and convariance functions by a similar analysis (see Supplementary material for details). However, the training objective of the level-set prior is different from the explicit parameterization as the network must classify points as inside or outside the surface. This supervision can be challenging to obtain from noisy data, especially for thin structures. We provide a comparison with this approach in Section 5.4.

5.4 Experiments

In this section we will present quantitative and qualitative results for applying the manifold prior to multiple manifold reconstruction tasks. All the experiments

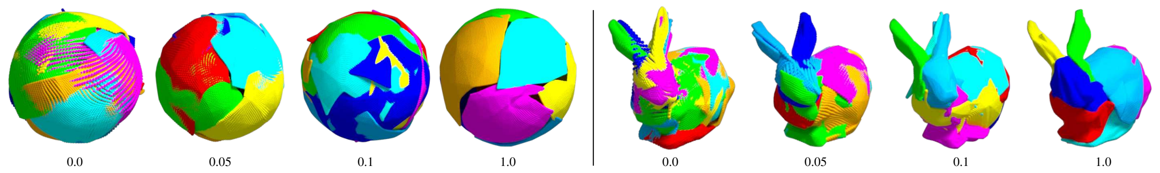


Figure 5.4: Effect of the regularization weight on the reconstructed manifold. For this experiment, we use our method to reconstruct a sphere using an atlas with 8 charts and render each one with a different color. Without any regularization, there is a significant amount of deformation applied to each surface (hence the space between the points) and a considerable amount of overlap between different parts. As the regularization weight increases, those aspects are noticeably reduced.

in this paper were implemented using Python 3.6 and PyTorch. Computation was performed on TitanX GPUs.

5.4.1 Denoising and Interpolation

Benchmark Our benchmark consists of 8 different 3D shapes with diverse characteristics. The shapes are normalized to fit a unit cube and 16K points are sampled on their surfaces. The point positions are perturbed by a Gaussian noise with standard deviation 2×10^{-3} and zero mean. Figure 5.7 shows the ground-truth shapes as well as their noisy counterpart. Since the level-set representation and the baseline methods (RIMLS [134], SPSR [91]) require normal information, we estimate the normal for every point by using the local frame defined by its nearest neighbors. We experimented multiple numbers of neighbors for both baselines and used the value that led to the best results: 20 neighbors for SPSR and the level-set representation, 30 neighbors for RIMLS. The network used in the level-set representation follows the same architecture and training protocol as the one used for the explicit parametrizations (described in the next paragraph). However, it is trained to predict every point as outside (+1) or inside the surface (-1). Points with positive values are generated by translating every point in the point cloud along the normal direction for a distance $\epsilon = 2 \times 10^{-3}$. Points with negative values are generated in the same way, but applying a displacement to the opposite direction. For RIMLS, we used a relative spatial filter size of 10, 15 projection iterations and a volumetric grid with 200^3 resolution. For SPSR, we used an octree with depth 7 and 8 iterations.

Experimental setup Our method performs denoising by minimizing Equation 5.4. In this framework, P is the noisy point cloud we are trying to reconstruct and $\boldsymbol{f_\theta}$ is a neural network. In all experiments we use a neural network with 3 fully connected layers, where the layers have 256, 128 and 64 hidden units, respectively. The output of the networks is a point in $\mathbb{R}^3$. The input can be either a point in $\mathbb{R}$ (1-manifold) or $\mathbb{R}^2$ (2-manifold). We use *ReLU* activations followed by batch normalization at each layer, except for the last, where we use a *tanh* non-linearity. We vary the architecture of $\boldsymbol{f_\theta}$ with respect to the number of parameterizations (1 or 8) and dimensionality (1 or 2). Additionally, we try each one of these architectural variations with $\lambda = 0$ and $\lambda = 1.0$. When using 8 parametrizations, 4096 points are sampled per parametrization. When using just one parametrization, 16K points are sampled. We optimize our objective through gradient descent using the Adam optimizer with learning rate 10^{-3}. For evaluation, we uniformly sampled 16K points in the computed manifold (represented as a triangular mesh) and compute the Chamfer distance with respect to the ground-truth.

Results and discussion. Our methods significantly outperform the baselines for most of the shapes. Quantitative results can be seen in Table 5.1 and the qualitative results are shown in Figure 5.7. The numbers are computed using 8 parametrizations (for surfaces and curves) and $\lambda = 1.0$. A comparison between different variations of our approach is displayed in Table 5.2. RIMLS, SPSR and level-set representations (*Implicit* in Table 5.1) have trouble reconstructing point clouds with a significant amount of noise. This is due to the fact that those methods rely on accurate surface normal estimates to infer inside/outside regions of the shape. Besides, RIMLS and methods based on implicit functions (SPSR and level-set representations) work better when dealing with closed surfaces. Shapes that are better approximated by contours (ring, spiral, chair's legs) are particularly challenging for those approaches. On the other hand, the networks parametrizing explicit functions (*Surface* and *Contour* in Table 5.1) are able to adapt to different structures and present a fair performance across a diverse set of shapes.

The results in Table 5.2 suggest that using multiple parametrizations gives a better approximation than just using a single one. This happens because complex shapes are easier to represent by multiple parametrizations. For example, while using a single 2-manifold parametrization, the ring tends to be approximated by a disk, which significantly increases the reconstruction error when the points are uniformly sampled over the final mesh. This behavior is illustrated in Figure 5.5. Our ablation studies also indicate that using stretch regularization helps parametrizations of both surfaces and contours. Figure 5.4 shows the effect of stretch regularization for two different shapes. As the regularization weight increases, the overlap between different parameterizations becomes smaller. When overlaps exist, the manifold representation is suboptimal – the same regions are being generated multiple times.

Interpolation We also explored using the manifold prior for point cloud interpolation. This experiment follows the same experimental setup as denoising. However, instead of perturbing the points with Gaussian noise, we randomly select 1K points out of 16K. Interpolation is performed by minimizing Equation 5.4. Results can be seen in Figure 5.5. For these experiments we use a single parameterization and include stretch regularization, without which the surface has holes and significant folds. Our method is able to reconstruct reasonable surfaces from a small set of points.

Comparison with the deep image prior We also compare our approach to the deep image prior [188] for interpolating points in 2D images. Results are presented in Figure 5.6. We use the same architecture from [188] while minimizing the mean

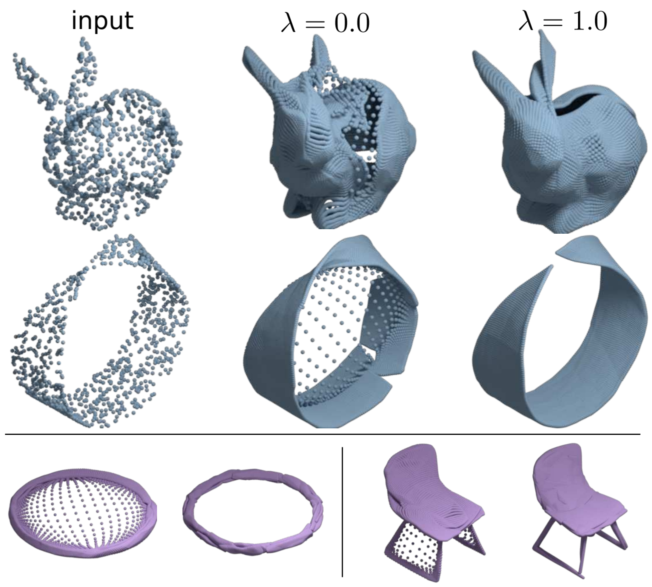


Figure 5.5: *Interpolation results on the top.* Stretch regularization ($\lambda = 1.0$) helps generate smoother surfaces. *On the bottom, denoising using one vs. multiple parametrizations.* Shapes on the left were reconstructed using a single parameterization, whereas shapes on the right used 8 parameterizations. Using multiple parameterizations helps reconstruct complex shapes.

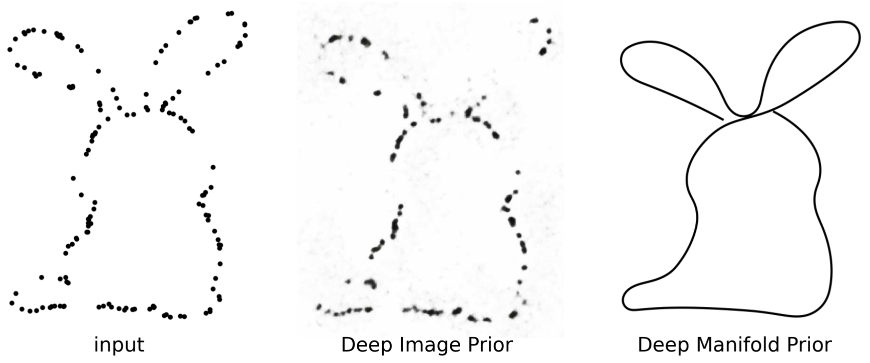


Figure 5.6: Comparison to the deep image prior [188]. Image-based prior (middle) is not able to connect the dots in the input image (left). On the other hand, the manifold prior is able to reasonably interpolate the dotted drawing.

squared error with respect to the image pixels. For the manifold prior, we use a single 1-manifold parameterization following the architecture described before, differing only in the dimensionality of the output: points this this case are in $\mathbb{R}^2$ instead of $\mathbb{R}^3$. Coordinates of the black pixels in the input image are used to form a point cloud and the manifold is computed by minimizing Chamfer distance with respect to it.

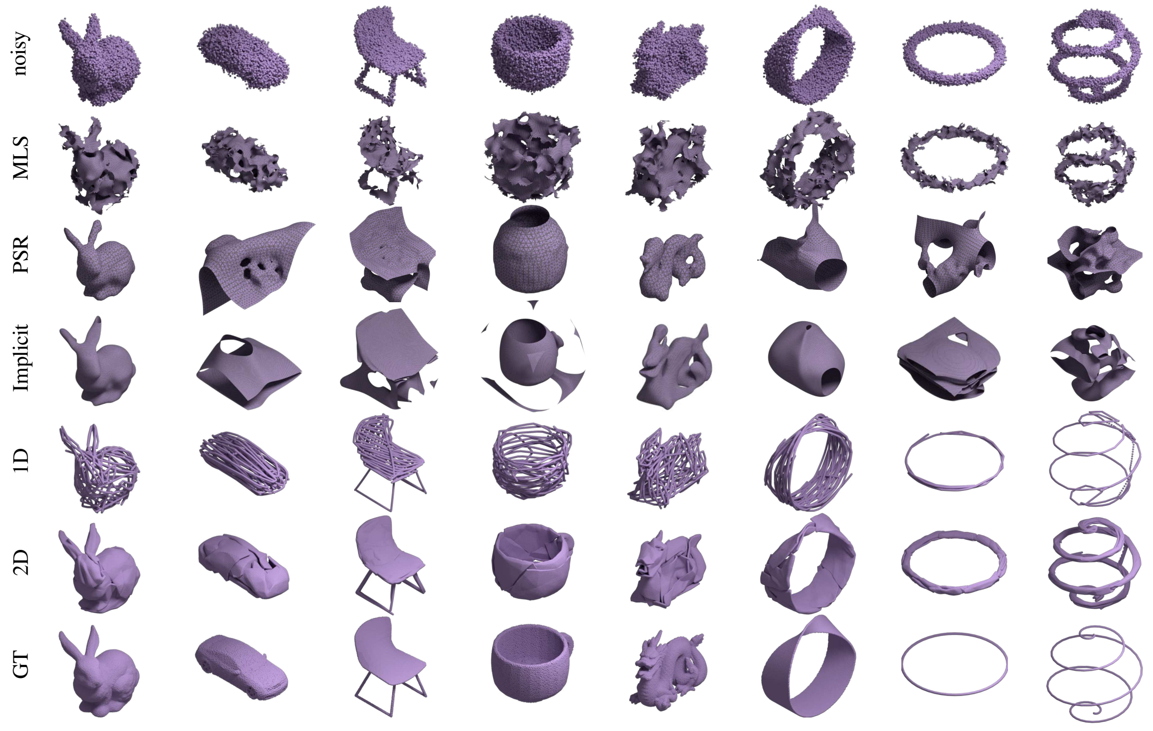


Figure 5.7: Qualitative comparison between different denoising methods. Rows display different methods, whereas columns display different shapes. Baseline methods do not perform as well as the deep manifold prior, even for closed surfaces like the bunny (first column) and the dragon (fifth column). As we can see, 2-manifold parameterizations are better for reconstructing surfaces, whereas 1-manifold counterparts reconstruct the curves (last two columns) more acurattely.

5.4.2 Learning from data

Finally, we show how the insights presented in the earlier sections, in particular convolutional parameterization and stretch regularization, can also improve generative models of 3D shapes when trained on a large collection of shapes.

To measure the effect of the stretch regularization in a learning-based scenario, we train a model using the same architecture as AtlasNet [68] on a subset of $50,000$ shapes across 13 categories of the ShapeNet dataset [24]. Adding stretch regularization did not significantly impact the Chamfer metric – error of 1.46×10^{-3} and 1.47×10^{-3} with and without regularization. However, the results are qualitatively better. As seen in Figure 5.8 the regularization reduces the stretch and overlap of the generated surfaces, and eliminates artifacts where holes are incorrectly filled.

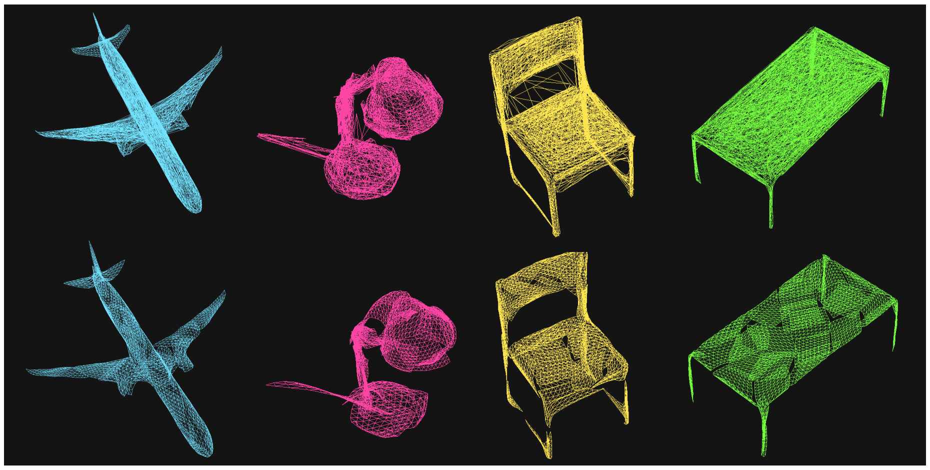


Figure 5.8: Autoencoder results. Results on using AtlasNet [68] trained w/o (top) and w/ (bottom) stretch regularization. The latter results in meshes with reduced deformation and overlap, and removes artifacts where the chair's back is incorrectly filled.

Architecture	mean/cat.	mean/inst.	#params.
MRTNet	4.80	4.26	81.6M
AtlasNet	4.74	4.38	42.6M
ConvAtlas	**4.53**	**4.00**	**14.5M**

Table 5.3: Quantitative results for single-view image-to-shape reconstruction. The table reports the mean Chamfer distance metric (scaled by 10^3) computed per category and per instance.

We also train a convolutional decoder with stretch regularization on the single-view reconstruction benchmark [35]. Our approach called ConvAtlas is compared against AtlasNet and MRTNet [55] in Table C.1. For a fair comparison, we use 4K points for evaluation across all methods. ConvAtlas outperforms both approaches in terms of per-category and per-instance error, and also leads to more compact models. Per-category results and experimental details are in the Supplementary material.

5.5 Conclusion

We presented a manifold prior induced by deep neural networks. Our experiments show that the prior can be effectively used for a variety of manifold reconstruction tasks: denoising, interpolation and single-view reconstruction. Besides, we analyzed the influence of the architecture in the characteristics of the prior by posing the models as GP. In conjunction to the prior induced by deep networks, we showed that using a stretch regularization procedure enables better manifold approximation and improves the quality of the generated meshes, reducing large deformations and overlaps between different parameterizations.

CHAPTER 6

SHAPE RECONSTRUCTION USING DIFFERENTIABLE PROJECTIONS AND DEEP PRIORS

Consider the problem of reconstructing a 3D shape from silhouettes. The classic visual hull algorithm that intersects the visible volumes from each viewpoint is easy to implement but is sensitive to errors in viewpoint estimation and silhouette noise. A Bayesian approach for this problem would be to add appropriate priors over the shape and viewpoint estimates and perform posterior inference. This is challenging for two reasons. First, the search space of 3D shape is large since there is no compact shape basis to search over for general shapes. Second, Bayesian inference is typically expensive for high-dimensional data.

To this end we present *differentiable projection operators* $\mathcal{T}$ and *deep shape priors* for which Bayesian inference can be performed via stochastic gradient descent and their variants [200]. While many priors exist, of interest is the "deep shape prior" of Ulyanov *et al.* [188] which showed that the space of natural images $\boldsymbol{x}$ can represented as a parametric family $f_{\boldsymbol{\theta}}(\eta)$ where f is a convolutional network, θ its parameters, and η is a fixed input. Their work showed that search over natural images can be replaced by a search over the parameters of the network $\boldsymbol{\theta}$, which can be efficiently done via gradient descent.

Our work takes this idea further. First, we endow the deep image prior with 3D convolutions resulting in a deep shape prior. Second, we incorporate differentiable projection operators $\mathcal{T}$ that model projection measurements, such as silhouettes, given projection parameters ϕ such as viewpoints. Thus inferring a shape $\boldsymbol{x}$ given noisy

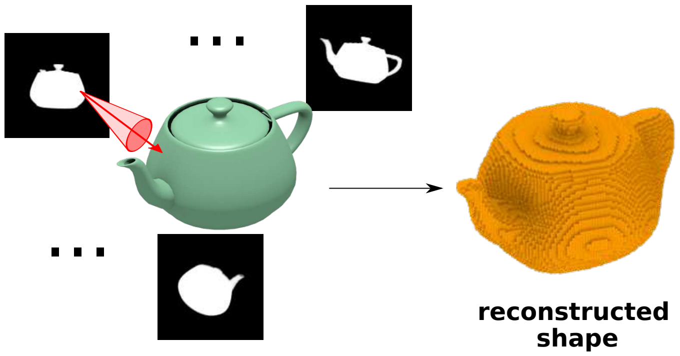


Figure 6.1: Shape reconstruction from binary images with uncertain viewpoints. We propose to use deep networks together with differentiable projection operators for shape reconstruction. Our approach leverages the shape prior induced by neural networks to reconstruct shapes from projections without any learning procedure. Additionally, our approach can use differentiable operators to reconstruct shapes under noisy projection measurements, like perturbed viewpoint information.

projection measurements $\boldsymbol{y}$ reduces to the following optimization over network parameters $\boldsymbol{\theta}$ and projection parameters ϕ:

$$\min_{\phi,\boldsymbol{\theta}\in\mathbb{R}^D} E\left(\boldsymbol{y}, \mathcal{T}(f_{\boldsymbol{\theta}}(\boldsymbol{\eta}), \phi)\right) + P(\phi), \tag{6.1}$$

where $P(\phi)$ is a prior over projection parameters, which is often a simple function. We show that for a number of shape construction problems such as tomographic reconstruction, shape from silhouettes or depth maps, it is possible to construct projection operators using existing neural network building blocks that are differentiable with respect to both the input and projection parameters. Thus the objective can be minimized using "backpropagation" machinery, which is generally much faster than Bayesian inference using Markov Chain Monte Carlo (MCMC) techniques.

Apart from choosing the network architecture and the projection operator, the approach does not require any task-specific training. Nevertheless, it yields compelling results for tomographic reconstruction in the low sampling regime, where it outperforms a state-of-the-art approach based on iterative BM3D [113]. Our work also shows that the deep image prior generalized to 3D volumes is effective at modeling 3D shapes. In problems such as visual hull reconstructions, or reconstruction from depth maps, we can accurately estimate the 3D shape of an object from only a few views, even when there are uncertainties in the view estimates, or when depth maps are corrupted by noise. The reconstruction results are significantly better than handcrafted priors. These tasks are illustrated in Figures 6.3-6.9.

6.1 Related work

In this section we briefly summarize techniques for solving inverse problems for image and volumetric reconstruction of the form:

$$\min_{\boldsymbol{x} \in \mathcal{X}} P(\boldsymbol{x}) + E(\boldsymbol{y}, \mathcal{T}(\boldsymbol{x})). \tag{6.2}$$

The data term E and the projection operator $\mathcal{T}$ are application specific, but there is considerable flexibility on modeling the prior term P. These include smoothness priors such as total variation (TV) [150] and L_0 gradients [210], Gaussian mixture models over patches [227], denoising autoencoders [190]. The deep image prior [188] represents images as the output convolutional network with random parameters from a fixed (random) input. The authors showed that outputs of networks consisting of several convolutional and pooling layers, followed by several deconvolutional layers with few or no skip connections in between tend to generate natural images. Recently, an extension to the deep image prior shows that it is asymptotically equivalent to a Gaussian Process [32]. This suggests a Bayesian approach to the problem: conducting posterior inference through Langevin dynamics avoids the need for early stopping and improves results for denoising and inpainting tasks. The deep image prior is also related to procedural priors such as bilateral filtering [182], non-local means [20], or block matching 3D (BM3D) [39]. These models use non-local self-similarity of patches in images to collectively denoise them.

For complex projection operators $\mathcal{T}$ involving noisy and incomplete measurements $\boldsymbol{y}$, applying procedural priors is non-trivial. Suppose $\boldsymbol{y}$ and $\boldsymbol{z}$ denote the observed and unobserved projection measurements corrupted by noise: $(\boldsymbol{y}, \boldsymbol{z}) = T(\boldsymbol{x}) + \delta$. For example $\boldsymbol{y}$ could denote the subset of frequencies in the Fourier transform, or projections of data in a compressed sensing application. Maggioni *et al.* [113] proposed the following iterative scheme:

1. Estimate $\boldsymbol{x}$ by inverting the measurement $\boldsymbol{x}^{(k)} = \mathcal{T}^{-1}(\boldsymbol{y}, \boldsymbol{z}^{(k)})$ starting from $\boldsymbol{z}^{(1)} = 0$.

2. Denoise $\boldsymbol{x}^{(k)}$ using BM3D to obtain $\boldsymbol{x}^{(k+1)}$.

3. Re-estimate $(., \boldsymbol{z}^{(k+1)}) = \mathcal{T}(\boldsymbol{x}^{(k+1)}) + \delta^{(k)}$. Note that only the unobserved part of projection is estimated keeping $\boldsymbol{y}$ fixed across iterations.

The iterative BM3D can be applied to problems where the support of $\mathcal{Y}$ is small. This procedure is related to the alternating direction method of multipliers (ADMM) [16] which has been applied for solving linear inverse problems of the form: $\min_{\boldsymbol{x}} ||\boldsymbol{y} - A\boldsymbol{x}||_2^2 + \lambda P(\boldsymbol{x})$. ADMM solves the augmented Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u})$:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}) = ||\boldsymbol{y} - A\boldsymbol{z}||_2^2 + \lambda P(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z} + \boldsymbol{u}||_2^2$$

over auxiliary variables $\boldsymbol{z}$ and $\boldsymbol{u}$ for $\rho > 0$ by alternatively optimizing $\boldsymbol{x}$, $\boldsymbol{z}$, and $\boldsymbol{u}$ as:

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &\leftarrow \operatorname*{argmin}_{\boldsymbol{x}} \lambda P(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{x} - \boldsymbol{z}^{(k)} + \boldsymbol{u}^{(k)}||_2^2 \\
\boldsymbol{z}^{(k+1)} &\leftarrow \operatorname*{argmin}_{\boldsymbol{z}} ||\boldsymbol{y} - A\boldsymbol{z}||_2^2 + \frac{\rho}{2}||\boldsymbol{x}^{(k+1)} - \boldsymbol{z} + \boldsymbol{u}^{(k)}||_2^2 \\
\boldsymbol{u}^{(k+1)} &\leftarrow \boldsymbol{x}^{(k+1)} - \boldsymbol{z}^{(k+1)} + \boldsymbol{u}^{(k)}
\end{aligned}$$

The optimization decouples the reconstruction and the prior. The first involves inference with an image prior and squared-loss term. The second objective is quadratic in $\mathbf{z}$ can be solved with conjugate gradient decent. The decoupling allows use of explicit or implicit priors, as well as learned proximal projection operators [25, 213] $\mathbf{proj}(\boldsymbol{z}-\boldsymbol{u}, \rho)$ that map a vector $\boldsymbol{z}-\boldsymbol{u}$ to $\boldsymbol{x}$ in the manifold of natural images within a distance ρ from it, similar to a denoising autoencoder, to solve the inverse problem.

Finally, a class of approaches directly learn the inverse mapping $\mathcal{G} : \mathcal{Y} \rightarrow \mathcal{X}$ using rich parametric models such as a neural network in a fully-supervised manner. These models amortize inference during training and enable efficient inference given noisy measurements. Such models have been successfully applied for various inverse problems such as super resolution [44], denoising [208], colorization [101, 219], and estimating depth and normals from images [48]. However a disadvantage is the architecture and parameters of the model are likely to be specific to the noise and projection operators, which require separate training for each task.

Closely related to this work, recent approaches have employed geometric transformations on deep features to generate novel views of a 3D object [129, 164]. In contrast to our approach, those techniques do not explicitly define the projection operators – they are parameterized by a deep neural network. As a consequence, the inferred representation does not directly correspond to a 3D shape, but to a higher lever representation learned by the model.

6.2 Method

Our approach for Bayesian inference will be to optimize the objective in Equation 6.1 using Stochastic Gradient Descent (SGD). This corresponds to a Maximum Likelihood Estimate (MLE), or Maximum A-Posteriori (MAP) estimate if priors over parameters $\boldsymbol{\theta}$ are added. Although more sophisticated schemes for SGD based posterior sampling exist [32, 200], we find that SGD works reasonably well for the problems we consider.

Solving reconstruction problems with SGD requires formulating differentiable projection operators and differentiable priors over the shapes. We use the deep image prior for image-based reconstruction tasks, and a 3D convolutional version for shape reconstruction tasks. In earlier work the deep image prior was used to solve a number of reconstruction problems with linear measurements [188]. For example in denoising the projection operator is the identity transformation, while in inpainting the projection operator is a mask indicating which pixels are present and absent. In this section, we present three differentiable projection operators that can be combined with deep neural networks for reconstructing shapes from partial and noisy observations.

6.2.1 Radon Projection ($\mathcal{T}_R$)

In [144], Radon proposed the utilization of the inverse of an integral transform to reconstruct images from a CT scan. The forward version of this transform is known as Radon transform R and can be described by the following:

$$R(\phi, r) = \int_L s(x,y)dl,\ L = \{(x,y)|x\sin\phi - y\cos\phi = r\} \tag{6.3}$$

where s represents a density function, ϕ is the angle of projection, and this transform represents data obtained as the output of a CT scan. Let $T_\psi(s)$ be an operator that rotates s by ψ degrees, i.e. $T_\psi(s)(x,y) = s(x\cos\psi - y\sin\psi, x\sin\psi + y\cos\psi)$. Plugging this in Equation (6.3) we have:

$$\begin{aligned} R(\phi, r) &= \int_L T_\psi(s)(x,y)dl \\ L &= \{(x,y)|x\sin(\phi+\psi) - y\cos(\phi+\psi) = r\} \end{aligned}$$

Taking $\psi = -\phi$:

$$R(\phi, r) = \int_L T_{-\phi}(s)(x,y)dl, \quad L = \{(x,y)|y = r\} \tag{6.4}$$

$$R(\phi, r) = \int_{\mathbb{R}} T_{-\phi}(s)(x, r)dx \tag{6.5}$$

In practice, s is represented by image and $T_{-\phi}(s)$ is computed by rotating a regular grid and resampling the image as described in [82]. Specifically, let $I_{i,j}^{(\phi)}$ be the value of the pixel i, j in the image formed by s rotated by $-\phi$ degrees, the discrete version of the Radon transform is:

$$R(\phi, r) = \sum_{i=1}^{S} I_{i,r}^{(\phi)}, \tag{6.6}$$

where S is the size of the image. Notice that the result of the Radon transform R is also an image (called sinogram and is parametrized by ϕ and r) as can be seen in Figure 6.3. Finally, our operator $\mathcal{T}_R$ receives an image I of size $S \times S$, a set of values ϕ representing the projection angles and outputs an image of size $S \times |\phi|$. The process is differentiable and can be implemented as a sum over one dimension of multiple rotated images.

6.2.2 Silhouette Projection ($\mathcal{T}_S$)

Shape reconstruction from silhouettes consists in the following problem: given a set of silhouette images of the same object from different views, estimate the 3D shape of the object. Silhouette projection can be formulated as a differentiable operator $\mathcal{T}_S(V, \phi)$. To do so, we represent 3D shape as a voxel grid V, and the projection $\mathcal{T}_S(V, \phi)$ generates a silhouette of the shape V captured from a view ϕ. The formulation of $\mathcal{T}_S$ follows [52]. Specifically, let $V : \mathbb{Z}^3 \to [0, 1] \in \mathbb{R}$ be the voxel grid, representing the occupancy value at a given integer 3D coordinate $c = (i, j, k)$. The rotated version of the voxel grid $V(c)$ is defined as $V_\phi(c) = \Phi(V, T_\phi(c))$, where $T_\phi(c)$ is the coordinate obtained by rotating c around the origin according to ϕ and $\Phi(V, c)$ is a procedure that samples a value of V in a position c – trilinear or nearest neighbor sampling.

The next step consists in performing the projection to create an image from the rotated voxel grid. This is done by applying the projection operator $P(V)_{i,j} = 1 -$

$e^{-\tau \sum_k V(i,j,k)}$. The intuition behind this operator is similar to the idea of the Radon transform: compute a line integral of the occupancy function V along each line of sight (assuming othographic projection), with the difference that here we apply an exponential falloff to create a smooth and differentiable function. The smoothness can be controlled by the parameter τ: bigger values result in binary images. If there all voxels along the line of sight are empty, the projection results in a value of 0; as the number of non-empty voxels increases, the value approaches 1. Combined with the rotated version of the voxel grid, we define our final projection operator as: $\mathcal{T}_S(V, \phi)_{i,j} = 1 - e^{-\tau \sum_k V_\phi(i,j,k)}$ where i, j is the pixel coordinate of the resulting image.

6.2.3 Depth Image Projection ($\mathcal{T}_D$)

Given a 3D shape represented as a voxel occupancy grid V and a view ϕ, the depth image captures the distance values from the viewpoint to the visible points on the shape. This is useful in practical applications as depth images are frequently captured by LiDAR and similar depth sensors. Here, we demonstrate that the depth projection operator can be built upon the silhouette projection operator. To do so, we first define a visibility function $A(V, \phi, c)$ that describes whether a given voxel c inside the grid V is visible, when seen from a view ϕ:

$$A(V, \phi, i, j, k) = \exp\left\{ -\tau \sum_{l=1}^{k} V_\phi(i, j, l) \right\} \tag{6.7}$$

Intuitively, this is the complement of the silhouette projection, the difference is that we are incrementally accumulating the occupancy (from the first voxel on the line of sight) as we traverse the voxel grid, instead of summing all voxels on entire the line of sight. If voxels on the path from the first to the current voxel are all empty, the value of A is 1 (indicating the current voxel is 'visible' to the view ϕ). If there is at least one non-empty voxel on the path, the value of A will be close to 0 (indicating this voxel is not visible).

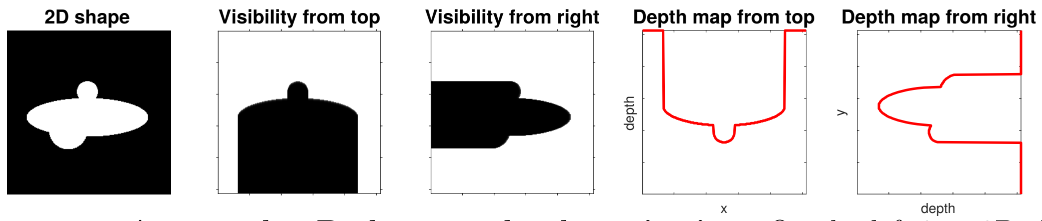


Figure 6.2: A example 2D shape to depth projection. On the left is a 2D shape visualized as a binary occupancy (white is occupied). The visibility map for each pixel from the top and right views are shown next – a pixel is white (value=1) if it is visible. The depth maps are obtained by summing the visibility maps along the vertical and horizontal directions for the top and right views receptively.

Now that we have the visibility value of each voxel, the depth value of a pixel in the projected image is simply the line integral of A along the line of sight: $D(i,j) = \sum_k A(V,\phi,i,j,k)$. This accumulates the number of voxels along the entire line of sight that are visible, therefore it gives the depth value. Refer to Figure 6.2 for illustrations.

While using this operator along with a neural network, we found that it works better if we apply an exponential decay. Thus, we can define the depth projection operator $\mathcal{T}_D$ as follows:

$$\mathcal{T}_D(V,\phi)_{i,j} = 1 - \exp\left\{ -\sum_k A(V,\phi,i,j,k)\right\} \tag{6.8}$$

This smoothly maps the depth value to the range between [0,1]. Specifically, it maps a depth value of 0 to 0, and infinity to 1, while still remaining a differentiable operator.

6.3 Experiments

This section presents the results of applying our shape projection operators along with deep shape priors for three reconstruction tasks.

Network Architecture. In the volumetric reconstruction experiments (i.e. reconstructing 3D shapes from silhouette images and depth images respectively), the

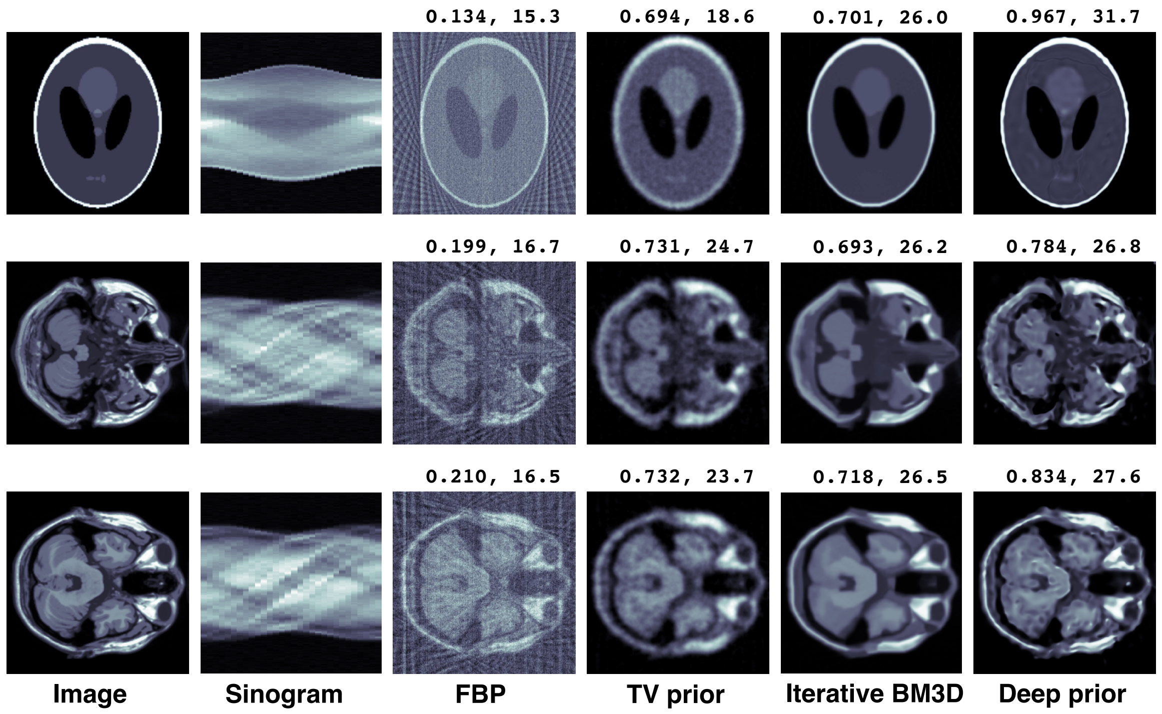


Figure 6.3: Tomographic reconstruction results from sinograms (radon transforms) sampled with $n = 30$ angles and noise ($\sigma = 1$). The sinogram is rescaled to the image size with nearest neighbor interpolation for visibility. From left to right in each row is the noise free image, the noisy sinogram, reconstruction with the filtered backprojection (FBP), TV prior, BM3D, and deep image prior. The SSIM and PSNR are shown for each approach on top of the corresponding figure. Our approach outperforms the other learning free baselines by a significant margin. *Zoom in for details.*

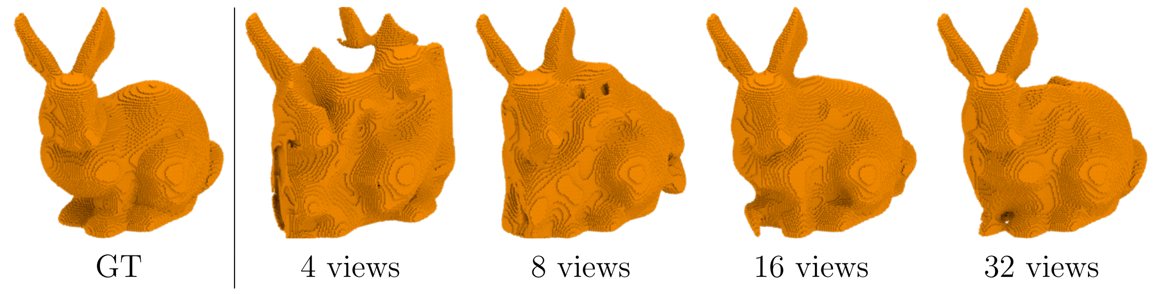


Figure 6.4: Effect of the number of views in the reconstruction from silhouettes. 3D shape reconstructed from silhouette images of the same object. Even without having access to enough 3D information, our method is still capable of generating plausible shapes.

network architecture is a fully convolutional UNet [148] where the encoder has 5 layers with 8, 16, 32, 64 and 128 filters. The decoder is a mirrored version of the encoder and skip connections are applied just in the 2 innermost layers. The upsampling is done through bilinear/trilinear interpolation followed by a convolution. All convolutions have filter size 3 and are followed by batch normalization and ReLU activation function. The input to the network is a tensor of the same size as the output, and its values sampled from $\mathcal{N}(0, 1)$. In all the experiments, we used Adam optimizer with learning rate=10^{-2}.

For the image reconstruction (i.e. tomography) we doubled the number of filters in each layer keeping the rest of the network architecture identical to account for higher spatial frequency of the underlying signal. The only other difference between the network that produces images and the one that produces voxel grids is that the convolutional operations are performed in 2D instead of 3D. Even though the network can be used to generate data of any size (since it is fully convolutional), in our experiments we set our image resolutions to 256×256 and voxel resolution to 128^3.

6.3.1 Tomography Reconstruction

In tomographic reconstruction our goal is to invert the sinograms as described in Section 6.2. With deep image prior the reconstruction involves solving the following optimization problem:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} ||R - \mathcal{T}_R(f_{\boldsymbol{\theta}}(\boldsymbol{\eta}))||_1, \tag{6.9}$$

where f is our neural network described above, $\boldsymbol{\eta}$ is its noise input, and R is the input sinogram (which may have low angular sampling rate and/or be corrupted by noise). To test the ability of our algorithm when handling challenging input, we use a low angular sample rate ($n = 30$) and simulate noisy sinograms by adding a Gaussian noise of $\sigma = 1$. Figure 6.3 shows the reconstruction results of the Shepp-Logan phantom image [160] and two separate slices of a sample from the BrainWeb database [36]. These images have been commonly used to evaluate CT reconstruction algorithms. For each reconstruction we compute the structured similarity (SSIM) index and PSNR values with respect to the groundtruth image (higher is better).

The standard solution for tomography is Filtered Back Projection (FBP): it inverts the Radon transform using the Fourier slice theorem. When angular sampling rate is low, the reconstruction using FBP turns out to have severe aliasing artifacts as seen in Figure 6.3 third column. The TV prior significantly improves the reconstructions for all three images. The iterative BM3D approach [113] described was run for 100 iterations. We noticed that the PSNR values converged after 100 iterations with the largest gains in PSNR in the first 20 iterations. Note that running BM3D on the FBP reconstruction corresponds to one iteration of this approach. For the deep prior we obtain results by running 2000 gradient steps. Compared to iterative BM3D, the deep prior produces reconstructions with significantly better SSIM values and comparable or better PSNR values (last two columns in Figure 6.3). The relatively poor performance of BM3D may be because the aliasing noise in CT reconstructions tends to be more structured and less like natural image noise when compared to the noise observed in image denoising applications. It takes many iterations for the iterative BM3D algorithm to get rid of the artifacts produced by the inverse radon

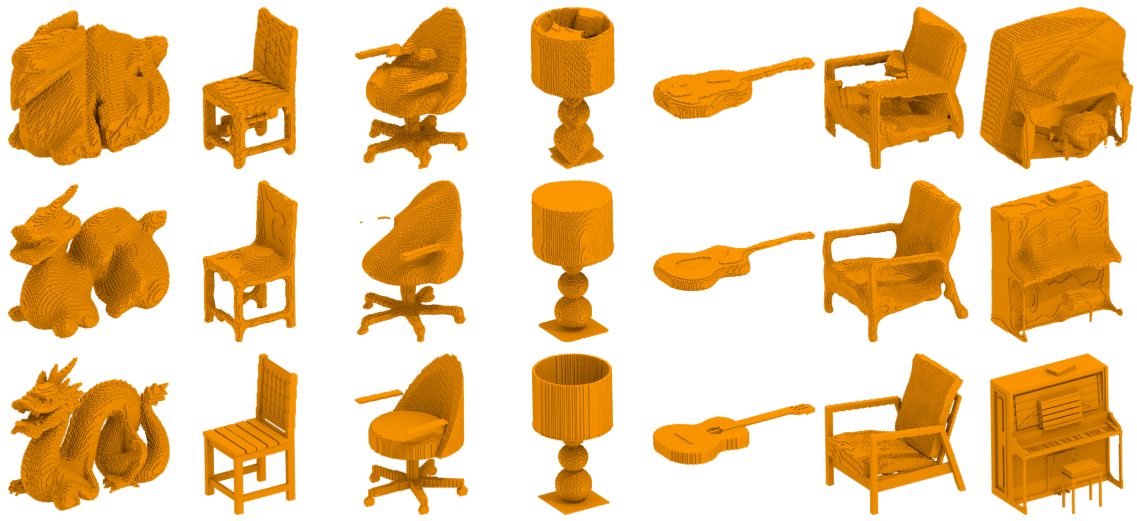


Figure 6.5: Reconstruction from silhouettes without viewpoint noise. 3D shapes reconstructed from 8 silhouette images of the same object. Viewing angles were sampled uniformly at random. Top row using space carving baseline, middle row using the deep image prior, bottom row is ground-truth.

transform but this causes smoothing of the underlying structures leading to lower SSIM scores.

6.3.2 Shape-from-Silhouette 3D Reconstruction

For 3D shape reconstruction from silhouette images, we employ the 3D convolutional neural network described as before to generate a voxel grid V where each voxel represents an occupancy value. The output of the network is then passed to the projection operator $\mathcal{T}_S$ along with a view direction ϕ. Given a set of N viewpoints $\boldsymbol{\phi} = \{\phi_1, \phi_2, ..., \phi_n\}$ and its associated images I_{ϕ_i}, our problem is described by the following optimization:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \sum_{i=1}^{N} ||I_{\phi_i} - \mathcal{T}_S(f_{\boldsymbol{\theta}}(\boldsymbol{\eta}), \phi_i)||_1, \tag{6.10}$$

where f is our neural network and $\boldsymbol{\eta}$ its noise input. We solve this minimization using gradient descent and then use $f_{\boldsymbol{\theta}}(\boldsymbol{\eta})$ to generate our final reconstruction. The results can be seen in Figure 6.4. Even with a small number of silhouette images, our method is able to reconstruct reasonable 3D shapes. The viewpoints for this example are chosen by evenly rotating the object along the horizontal axis (e.g. with 4 views, each view is 90 degrees apart; with 8 views, each is 45 degrees apart and so on). A baseline approach for this problem is space carving, which takes the intersection of all the projected views to generate the occupancy grid. We show a qualitative comparison with space carving in Figure 6.5. Space carving provides reasonable reconstructions for most of the shapes, but some of the objects contain artifacts like creases or even missing parts. On the other hand, the deep shape prior tends to create overly smooth shapes, which sometimes means removing some parts of the object (chairs in Figure 6.5) or adding content where should exist a sharp boundary (lamp in Figure 6.5).

View uncertainties. In the previous formulation, we assume that the set viewpoints $\boldsymbol{\phi}$ corresponds exactly to the observed views. However, a more realistic scenario is to assume that we are given a set of noisy viewpoint measurements. In this case, besides estimating the parameters of the network predicting the shape, we are also looking to estimate viewpoints $\hat{\boldsymbol{\phi}}$. We assume that the noisy viewpoints $\boldsymbol{\phi}$ are sampled independently from $VonMises(\boldsymbol{\phi}^*, \kappa)$, where κ is the dispersion of the Von-Mises distribution with mean $\boldsymbol{\phi}^*$ (the ground-truth viewpoints). This leads to adding an extra term to Equation 6.10 and also optimizing over the predicted viewpoints $\hat{\boldsymbol{\phi}}$:

$$\min_{\hat{\boldsymbol{\phi}},\boldsymbol{\theta}} \sum_{i=1}^{N} ||I_{\boldsymbol{\phi}_i^*} - \mathcal{T}_S(f_{\boldsymbol{\theta}}(\boldsymbol{\eta}), \hat{\boldsymbol{\phi}}_i)||_1 + \lambda \cos(\hat{\boldsymbol{\phi}}_i - \boldsymbol{\phi}_i), \tag{6.11}$$

where λ is the weight of the viewpoint regularization term. We use $\lambda = 0.1$ in our experiments. Notice that our projection operator is fully differentiable with respect to the viewpoint parameters and can be easily implemented using automatic differentiation packages.

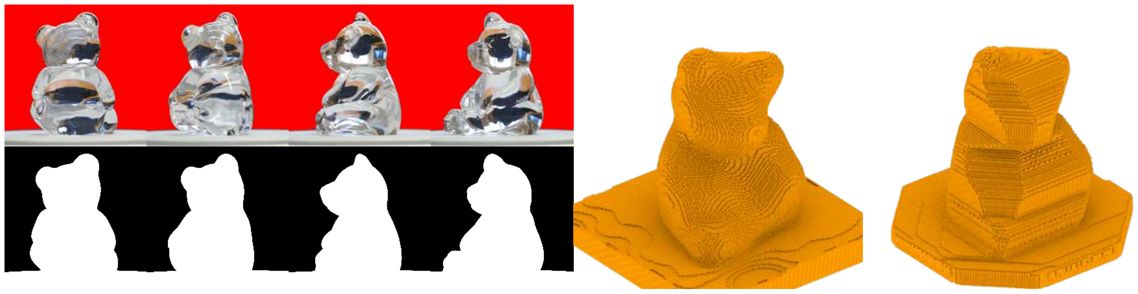


Figure 6.6: Shape-from-silhouette reconstruction using captured images. For this glass object, we photographed 4 views, with 45° angle apart, against a uniform background color. We then applied background-color removal and converted each image to binary silhouette image. The first reconstructed model is the result using our deep prior, whereas the second is the result using the space carving baseline.

	plane	bunny	car	desk	dragon	guitar	lamp	piano	plant	sofa	table	teapot	mean
Ours	0.35	**0.88**	**0.72**	**0.81**	0.59	0.64	**0.62**	**0.86**	**0.79**	**0.78**	**0.82**	**0.84**	**0.72**
Carving	0.49	0.77	0.59	0.41	0.55	0.51	0.26	0.64	0.58	0.51	0.44	0.83	0.55
Carving*	**0.51**	0.85	**0.72**	0.50	**0.62**	**0.71**	0.28	0.60	0.61	0.57	0.55	0.81	0.61

Table 6.1: 3D reconstruction from silhouettes with uncertain viewpoints. Intersection over union of the reconstructed occupancy from 12 different shapes. We randomly sample viewpoints to generate 8 binary images for each shape. Those viewpoints are slightly perturbed before being used by the methods, except for the last (Carving*) which corresponds to using space carving without noisy viewpoints. Our approach significantly outperforms the space carving baseline in all scenarios.

Evaluation To evaluate our approach we selected twelve meshes from standard benchmarks. Three of them are well know 3D shapes (Stanford bunny, dragon and Utah teapot) while the others were selected from 9 different categories of the ModelNet40 dataset [205]. We voxelize those shapes filling their interior to generate binary occupancy grids of resolution 108^3. Those voxel grids will correspond to our ground-truth data. Our network generates 128^3 occupancy grids, but we use data in a smaller resolution to zero-pad the volume and avoid artifacts in the boundaries. Next, we randomly sample 8 viewpoints and render a binary image I_{ϕ_i} from each sampled view. Since we want to evaluate the ability of the methods to reconstruct the 3D shape while dealing with view uncertainty, we sample views $\hat{\boldsymbol{\phi}}$ from $VonMises(\boldsymbol{\phi}, \kappa)$

and associate them with the corresponding binary images. We use $\kappa = 100$ for all the experiments. In other words, even though an image I_ϕ was rendered from a viewpoint ϕ, we assign a slightly perturbed viewpoint $\hat{\phi}$ to this image . Finally, we use the binary images I_ϕ and the perturbed viewpoints $\hat{\boldsymbol{\phi}}$ to reconstruct the 3D shape by minimizing the objective described in Equation 6.11. This is done through 500 steps of gradient descent. We compare our approach with a space carving baseline and report the intersection over union of the estimated occupancy grids in Table 6.1.

Our method outperforms vanilla space carving even when the viewpoints given are not perturbed, which demonstrates the robustness of our method to viewpoint perturbations. Figure 6.7 shows a qualitative comparison of the reconstructed shapes. Our approach reconstructs the shapes with high fidelity, preserving details and thin structures. On the other hand, space carving ends up reconstructing objects with missing parts and and rough structures as we can observe in Figure 6.7.

Reconstructions using captured images. We have also evaluated our method using images captured from a camera. Results are presented in Figure 6.6. The subject is a glass object, for which we photographed 4 views evenly spaced with $45°$ horizontal rotation angle apart from each other, against a uniform background color. We then use [1] to remove background and convert each image to a binary silhouette image. We compare results using our method with standard visual hull (i.e. space carving). As can be observed, our method leads to smooth reconstructions and the resulting objects look more natural. In contrast, the visual hull results contain artifacts and sharp transitions around changing views, which would require significantly more number of views to eliminate.

6.3.3 Shape-from-Depth Images 3D Reconstruction

The setup for 3D reconstruction from depth images is the same for the binary images except for the use of the projection $\mathcal{T}_D$ instead of $\mathcal{T}_S$. All the input depth images

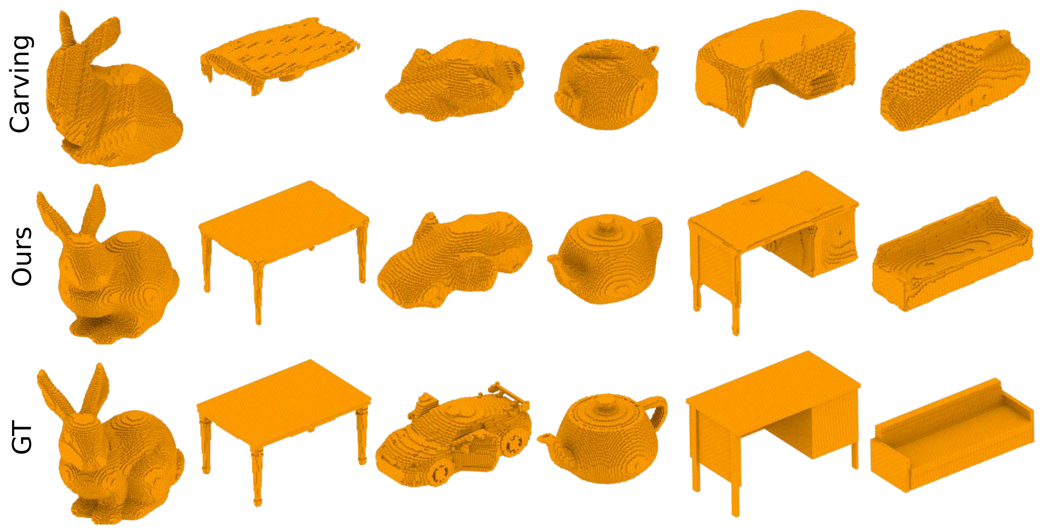


Figure 6.7: Shape-from-silhouette reconstruction with perturbed viewpoints. Results for the space carving baseline in the first row, our method in the second row, ground-truth shapes in the third row. Our results are computed minimizing Equation 6.11 through 500 gradient descent steps. Our method is capable of updating the initial viewpoint parameters and is capable to recover from imprecise viewpoint assignment. The space carving baseline is not robust to viewpoint perturbations which means it ends up carving the wrong regions of the volume, leading to poor reconstructions and eliminating thin object structures.

have their range scaled to be in $[0, 1]$ using the exponential map in Equation (7.4). We analyzed the ability of the method to reconstruct 3D shapes from depth images perturbed by different levels of Gaussian noise while using 4 views. Results can be seen in Figure 6.8. Additionally, we analyzed the reconstruction quality while varying the number of views. Results are presented in Figure 6.9. For these experiments, we kept the noise level very high ($\sigma = 0.1$). We notice that even when dealing with very noisy projections, our method is able to reconstruct high quality shapes if enough views are given.

6.4 Conclusion

We showed that by combining the deep image or volumetric prior with differentiable projection operators, signals can be reconstructed from a few noisy projection

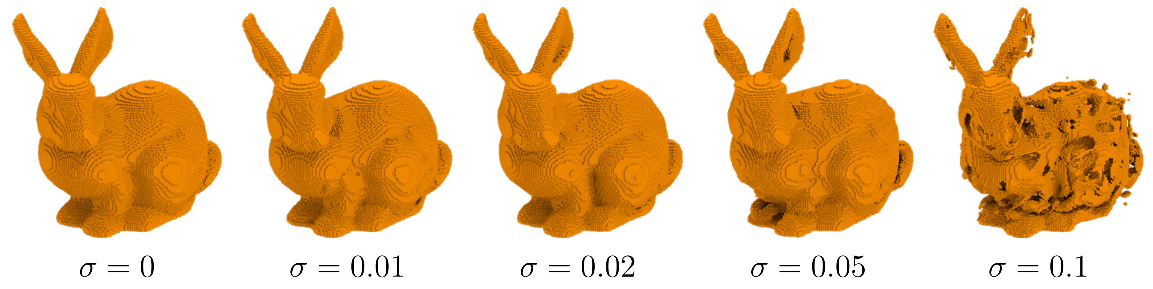


Figure 6.8: Effect of noise in the reconstruction. 3D shape reconstructed from 4 noisy depth images of the same object. The variance of the Gaussian noise increases from left to right. Shape prior can reconstruct high quality shapes even with considerable amount of noise.

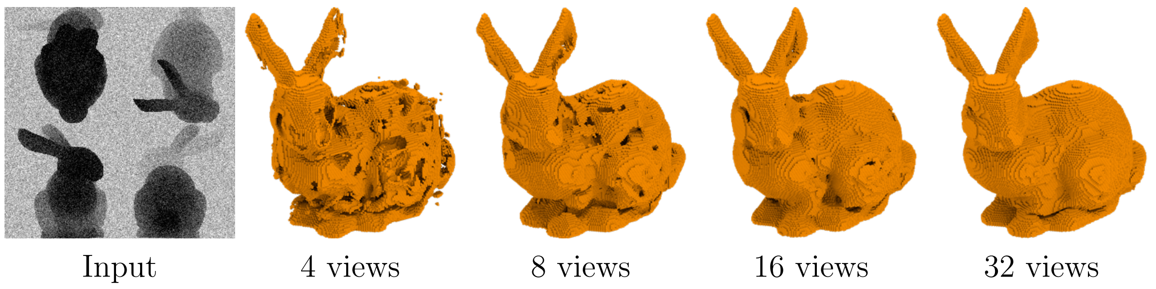


Figure 6.9: Effect of the number of views in the reconstruction from depth images. 3D shape reconstructed from very noisy ($\sigma = 0.1$) depth images of the same object. On the left, example of the input depth images. If provided enough views, our method is able to reconstruct high quality shapes even from highly noisy inputs.

measurements using stochastic gradient descent. The approach is learning free and can be used as a generic prior. Nevertheless, with a relatively simple network architecture our approach outperformed several handcrafted and procedural priors for image based and volumetric reconstruction tasks. Although we presented results for tomography and for shape reconstruction from silhouettes and depth maps, the approach can be used whenever the rendering or measurement process is differentiable. These include problems such as estimating shape from shading and geometry from multiple shaded images.

A potential issue is the use of volumetric representations for shapes which incurs high memory requirements and longer running times. A possible line of research is to investigate shape priors for more compact 3D representations like point clouds or multi-view. Combining deep priors with work on differentiable computer graphics pipelines opens up the possibility of applying this approach for solving inverse problems in many applications.

CHAPTER 7

INFERRING 3D SHAPES FROM IMAGE COLLECTIONS USING ADVERSARIAL NETWORKS

The ability to infer 3D shapes of objects from their 2D views is one of the central challenges in computer vision. For example, when presented with a catalogue of airplane silhouettes as shown in the top of Figure 7.1, one can mentally infer their 3D shapes by simultaneously reasoning about the shape and viewpoint variability. In this work, we investigate the problem of learning a generative model of 3D shapes from a collection of images of an unknown set of objects within a category taken from an unknown set of views. The images can be thought of as generalized projections of 3D shapes into a 2D space in the form of silhouettes, depth maps, or even part segmentations. The problem is challenging as one is not provided with the information about which object instance was used to generate each image, the viewpoint from which each image was taken, the parameterization of the underlying shape distribution, or even the number of underlying instances. Hence, traditional techniques based on structure from motion [13, 71] or visual hulls [102], cannot be directly applied.

We use the framework of generative adversarial networks (GANs) [63] and augment the 3D shape generator with a *projection module*, as illustrated in Figure 7.2. The generator produces 3D shapes, the projection module renders the shape from viewpoint sampled from a viewpoint distribution, and the adversarial network discriminates real images from generated ones. The projection module is a *differentiable renderer* that allows us to map 3D shapes to 2D images, as well as back-propagate the gradients of 2D images to 3D shapes. Once trained, the model can be used to

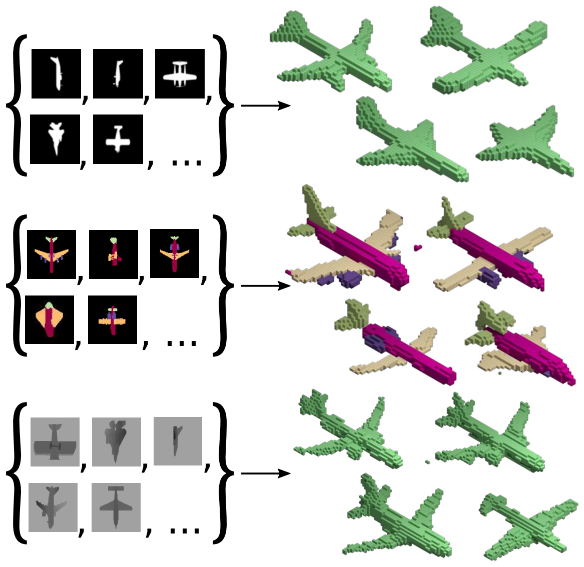


Figure 7.1: Our algorithm infers a generative model of the underlying 3D shapes given a collection of unlabeled images rendered as silhouettes, semantic segmentations or depth maps. To the left, images representing the input dataset. To the right, shapes generated by the generative model trained with those images.

infer 3D shape distributions from a collection of images (Figure 7.1 shows some samples drawn from the generator), and to infer depth or viewpoint from a single image, without using any 3D or viewpoint information during learning. We call our approach *projective generative adversarial network* (PrGAN).

While there are several ways of rendering a 3D shape, we begin with a silhouette representation. The motivation is that silhouettes can be easily extracted when objects are photographed against clear backgrounds, such as in catalogue images, but nevertheless they contain rich shape information. Real-world images can also be used by removing background and converting them to binary images. Our generative 3D model represents shapes using a voxel representation that indicates the occupancy of a volume in a fixed-resolution 3D grid. Our projection module is a feed-forward operator that renders the volume as an image. The feed-forward operator is differentiable, providing the ability to adjust the 3D volume based on projections. Finally, we assume that the distribution over viewpoints is known (assumed to be uniform in our experiments, but it could be any distribution).

We then extend our analysis first presented in our earlier work [52] by incorporating recent advances in training GANs and designing projection modules to incorporate richer supervision. The latter includes the availability of viewpoint information for each image, depth maps instead of silhouettes, or semantic segmentations such as part labels during learning. Such supervision is easier to collect than acquiring full 3D scans of objects. For example, one can use a generic object viewpoint estimator [170] as weak supervision for our problem. Similarly, semantic parts can be labeled on images directly and already exist for many object categories such as airplanes, birds, faces, and people. We show that such information can be used to improve 3D reconstruction by using an appropriate projection module.

To summarize our main contributions are as follows: (i) we propose PrGAN, a framework to learn probabilistic distributions over 3D shapes from a collection of 2D views of objects. We demonstrate its effectiveness on learning shape categories such as chairs, airplanes, and cars sampled from online shape repositories [24, 206]. The results are reasonable even when views from multiple categories are combined; (ii) PrGAN generates 3D shapes of comparable quality to GANs trained directly on 3D data [204]; (iii) The learned 3D representation can be used for unsupervised estimation of 3D shape and viewpoint given a novel 2D shape, and for interpolation between two different shapes, (iv) Incorporating additional cues as weak supervision improves the 3D shapes reconstructions in our framework.

7.1 Related work

Estimating 3D shape from image collections. The difficulty of estimating 3D shape can vary widely based on how the images are generated and the assumptions one can make about the underlying shapes. Visual-hull techniques [102] can be used to infer the shape of an object by computing the intersection of the projected silhouettes taken from known viewpoints. When the viewpoint is fixed and the lighting is known, photometric stereo [203] can provide accurate geometry estimates for rigid and diffuse surfaces. Structure from motion (SfM) [71] can be used to estimate the shape of *rigid objects* from their views taken from unknown viewpoints by jointly reasoning about point correspondences and camera projections. Non-rigid SfM can be used to recover shapes from image collections by assuming that the 3D shapes can be represented using a compact parametric model. An early example is that of Blanz and Vetter [13] for estimating 3D shapes of faces from image collections where each shape is represented as a linear combination of bases (Eigen shapes). However, 3D shapes need to be aligned in a consistent manner to estimate the bases which can be challenging. Recently, non-rigid SfM has been applied to categories such as cars and airplanes by manually annotating a fixed set of keypoints across instances to provide correspondences [88]. Our work augments non-rigid SfM using a learned 3D shape generator, which allows us to generalize the technique to categories with diverse structures *without* requiring correspondence annotations. Our work is also related to recent work of Kulkarni *et al.* [99] for estimating a disentangled representation of images into shape, viewpoint, and lighting variables (dubbed "inverse graphics networks"). However, the shape representation is not explicit, and the approach requires the ability to generate training images while varying one factor at a time.

Inferring 3D shape from a single image. Optimization-based approaches put priors on geometry, material, and light to estimate all of them by minimizing the reconstruction error when rendered [9,10,100]. Our approach on the other hand exploits implicit priors induced by deep networks [32,56] for generative modeling. Recognition-based methods have been used to estimate geometry of outdoor scenes [76,155], indoor environments [47, 156], and objects [5, 153]. More recently, convolutional networks have been trained to generate views of 3D objects given their attributes and camera parameters [45], to generate 3D shape given a 2D view of the object [173], and to generate novel views of an object [222]. Most of these approaches are trained in a fully-supervised manner and require 3D data or multiple views of the same object during training.

Generative models for images and shapes. Our work builds on the success of GANs for generating images across a wide range of domains [63]. Recently, Wu *et al.* [204] learned a generative model of 3D shapes using GANs equipped with 3D convolutions. However, the model was trained with aligned 3D shape data. Our work aims to solve a more difficult question of learning a 3D-GAN from 2D images. Several recent works are in this direction. Rezende *et al.* [145] show results for 3D shape completion for simple shapes when views are provided, but require the viewpoints to be known and the generative models are trained on 3D data. Yan *et al.* [211] learn a mapping from an image to 3D using multiple projections of the 3D shape from known viewpoints and object identification, i.e., which images correspond to the same object. Their approach employs a 3D volumetric decoder and optimizes a loss that measures the overlap of the projected volume on the multiple silhouettes from known viewpoints, similar to a visual-hull reconstruction. Tulsiani *et al.* [187] learn a model to map images to 3D shape provided with color images or silhouettes of objects taken from known viewpoints using a "ray consistency" approach similar to our projection module. Kanazawa *et al.* [87] employs additional supervision in the form of keypoint annotations to generate textured 3D meshes. On the other hand, our method does not assume known viewpoints, object associations of the silhouettes making the problem considerably harder. If object associations are given and viewpoints are unknown, a possible solution is to use multi-view consistency across similar objects, as demonstrated in [185]. More similar to our setup, Henderson and Ferrari [74] propose a method to learn a generative model of 3D shapes from a set of images without viewpoint supervision. However, their approach uses a more constrained shape representation – sets of blocks or deformations in a subdivided cube – and other visual cues such as lighting configuration and normals.

Differentiable renderers. Our generative models rely on a differentiable projection module to incorporate image-based supervision. Since our images are rendered as silhouettes, the process can be approximated using differentiable functions composed of spatial transformations and projections as described in Section 7.2. However, more sophisticated differentiable renders, such as [89,105,110], that take into account shading and material properties could provide richer supervision or enable learning from real images. These renderers rely on mesh-based or surface-based representations which are challenging to generate due to their unstructured nature. Recent work on generative models of 3D shapes with point clouds [2,49,51,55,68] or multiview [111,173] representations provide a possible alternative to our voxel based approach that we aim to investigate in the future.

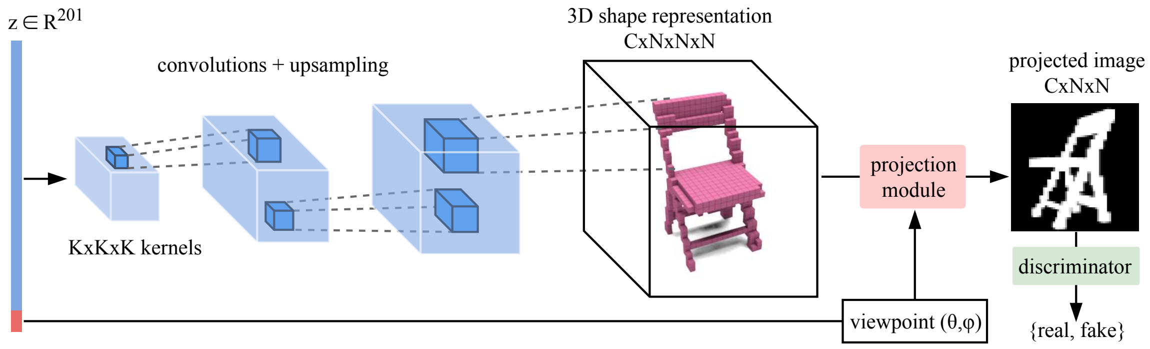


Figure 7.2: *The PrGAN architecture for generating 2D silhouettes of shapes factorized into a 3D shape generator and viewpoint generator and projection module.* A 3D voxel representation ($\mathrm{C} \times \mathrm{N}^3$) and viewpoint are independently generated from the input z (201-d vector). The projection module renders the voxel shape from a given viewpoint (θ, ϕ) to create an image. The discriminator consists of 2D convolutional and pooling layers and aims to classify if the generated image is "real" or "fake". The number of channels C in the generated shape is equal to one for an occupancy-based representation and is equal to the number of parts for a part-based representation.

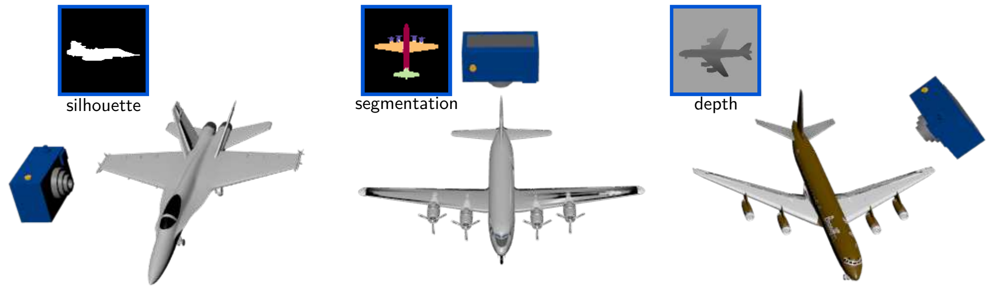


Figure 7.3: The input to our model consists of multiple renderings of different objects taken from different viewpoints. Those image are *not* annotated with identification or viewpoint information. Our model is able to handle images from objects rendered as silhouettes (left), semantic segmentation maps (middle) or depth maps (right).

7.2 Method

Our method builds upon GANs proposed in Goodfellow *et al.* [63]. The goal of a GAN is to train a generative model in an adversarial setup. The model consists of two parts: a *generator* and a *discriminator*. The generator G aims to transform samples drawn from a simple distribution $\mathcal{P}$ that appear to have been sampled from the original dataset. The discriminator D aims to distinguish samples generated by the generator from real samples (drawn from a data distribution $\mathcal{D}$). Both the generator and the discriminator are trained jointly by optimizing:

$$\min_G \max_D \mathbb{E}_{x\sim\mathcal{D}}[\log{(D(x))}] + \mathbb{E}_{z\sim\mathcal{P}}[\log{(1 - D(G(z)))}]. \tag{7.1}$$

Our main task is to train a generative model for 3D shapes without relying on 3D data itself, instead relying on 2D images from those shapes, without any view or shape annotation[1]. In other words, the data distribution consists of 2D images taken from different views and are of different objects. To address this mismatch we factorize the 2D image generator into a 3D shape generator ($\mathcal{G}_{3D}$), viewpoint generator (θ, ϕ), and a projection module $\mathcal{P}_{\theta,\phi}$ as seen in Figure 7.2. The challenge is to identify a representation for a diverse set of shapes and a differentiable projection module to

[1]We later relax this assumption to incorporate extra supervision.

create final 2D images and enable end-to-end training. We describe the architecture employed for each of these next.

3D shape generator (G_{3D}). The input to the entire generator is $z \in \mathbb{R}^{201}$ with each dimension drawn independently from a uniform distribution $\mathrm{U}(-1, 1)$. Our 3D shape generator $\mathcal{G}_{3D}$ transforms the first 200 dimensions of z to a $N \times N \times N$ voxel representation of the shape. Each voxel contains a value $v \in [0, 1]$ that represents its occupancy. The architecture of the 3D shape generator is inspired by the DCGAN [143] and 3D-GAN [204] architectures. It consists of several layers of 3D convolutions, upsampling, and non-linearities, as shown in Figure 7.2. The first layer transforms the 200 dimensional vector to a $256 \times 4 \times 4 \times 4$ vector using a fully-connected layer. Subsequent layers have batch normalization and ReLU layers between them and use 3D kernels of size $5 \times 5 \times 5$. At every layer, the spatial dimensionality is increased by a factor of 2 and the number of channels is decreased by the same factor, except for the last layer whose output only has one channel (voxel occupancy). The last layer is succeeded by a sigmoid activation instead of a ReLU in order to keep the occupancy values in $[0, 1]$.

Viewpoint generator (θ, ϕ). The viewpoint generator takes the last dimension of $z \in \mathrm{U}(-1, 1)$ and transforms it to a viewpoint vector (θ, ϕ). The training images are assumed to have been generated from 3D models that are upright oriented along the y-axis and are centered at the origin. Most models in online repositories and the real world satisfy this assumption (*e.g.*, chairs are on horizontal planes). We generate images by sampling views uniformly at random from one of eight pre-selected directions evenly spaced around the y-axis (*i.e.*, $\theta = 0$ and $\phi = 0°$, 45°, 90°, ..., 315°), as seen in Figure 7.3. Thus the viewpoint generator picks one of these directions uniformly at random.

Projection module (Pr). The projection module Pr renders the 3D shape from the given viewpoint to produce an image. For example, a silhouette can be rendered in the following steps. The first step is to rotate the voxel grid to the corresponding viewpoint. Let $V : \mathbb{Z}^3 \rightarrow [0,1] \in \mathbb{R}$ be the voxel grid, a function that given an integer 3D coordinate $c = (i,j,k)$ returns the occupancy of the voxel centered at c. The rotated version of the voxel grid $V(c)$ is defined as $V_{\theta,\phi} = V(\lfloor R(c,\theta,\phi) \rfloor)$, where $R(c,\theta,\phi)$ is the coordinate obtained by rotating c around the origin according to the spherical angles (θ,ϕ). Notice that R is straightforwardly implemented as a matrix multiplication and can be extended to model other types of transformations, *e.g.* perspective transformations. Refer to the Appendix A in [211] for more details.

The second step is to perform the projection to create an image from the rotated voxel grid. This is done by applying the projection operator $Pr((i,j),V) = 1 - e^{-\sum_k V(i,j,k)}$. Intuitively, the operator sums up the voxel occupancy values along each line of sight (assuming orthographic projection), and applies exponential falloff to create a smooth and differentiable function. When there is no voxel along the line of sight, the value is 0; as the number of voxels increases, the value approaches 1. Combined with the rotated version of the voxel grid, we define our final projection module as: $Pr_{\theta,\phi}((i,j),V) = 1 - e^{-\sum_k V_{\theta,\phi}(i,j,k)}$. As seen in Figure 7.3 the projection module can well approximate the rendering of a 3D shape as a binary silhouette image, and is differentiable. Section 7.4 presents projection modules that render the shape as a depth image or one labeled with part segmentations using similar projection operations, as seen in Figure 7.3. Thus the 2D image generator G_{2D} can be written compositionally as $G_{2D} = Pr_{(\theta,\phi)} \circ G_{3D}$.

Discriminator (D_{2D}). The discriminator consists of a sequence of 2D convolutional layers with batch normalization layer and LeakyReLU activation [112] between them. Inspired by recent work [143,204], we employ multiple convolutional layers with stride 2 while increasing the number of channels by 2, except for the first layer, whose input has 1 channel (image) and output has 256. Similar to the generator, the last layer of the discriminator is followed by a sigmoid activation instead of a LeakyReLU.

Training details. The entire architecture is trained by optimizing the objective in Equation 7.1. Usually, updates to minimize each one of the losses is applied once at each iteration. However, in our model, the generator and the discriminator have a considerably different number of parameters, as the generator is trying to create 3D shapes, while the discriminator is trying to classify 2D images. To mitigate this issue, we employ an adaptive training strategy. At each iteration of the training, if the discriminator accuracy is higher than 75%, we skip its training. We also set different learning rates for the discriminator and the generator: 10^{-5} and 0.0025, respectively. Similarly to the DCGAN architecture [143], we use ADAM with $\beta_1 = 0.5$ for the optimization.

7.3 Experiments

This section describes a set of experiments to evaluate our basic method and several extensions. First, we compare our model with a traditional GAN for the task of image generation and a GAN for 3D shapes. We present quantitative and qualitative results. Second, we demonstrate that our method is able to induce 3D shapes from unlabeled images even when the collection contains only a single view per object. Third, we present 3D shapes induced by our model from a variety of categories such as airplanes, cars, chairs, motorbikes, and vases. Using the same architecture, we show how our model is able to induce coherent 3D shapes when the training data contains images mixed from multiple categories. Finally, we show applications of our method in predicting 3D shape from a novel 2D shape, and performing shape interpolation.

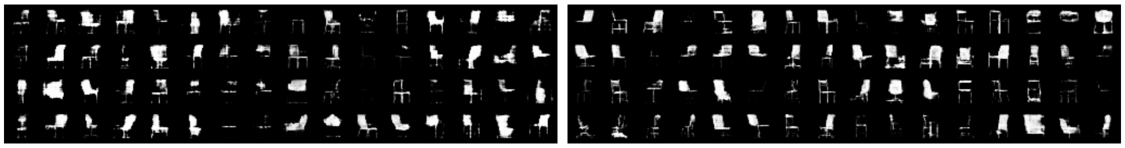
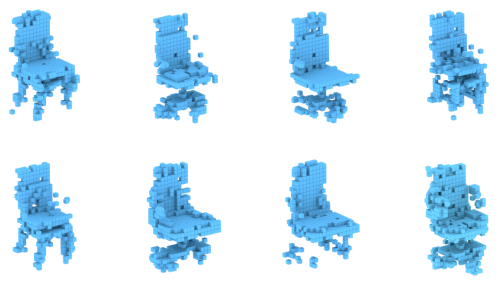

(a) Results from 2D-GAN. (a) Results from PrGAN.

Figure 7.4: Comparison between 2D-GAN [63] and our PrGAN model for image generation on the chairs dataset. Refer to Figure 7.9 third row, left column for samples of the input data.

7.3.0.0.1 Input data. We generate training images synthetically using 3D shapes available in the ModelNet [206] and ShapeNet [24] databases. Each category contains a few hundred to thousand shapes. We render each shape from 8 evenly spaced viewing angles with orthographic projection to produce binary images. Hence our assumption is that the viewpoints of the training images (which are unknown to the network) are uniformly distributed. If we have prior knowledge about the viewpoint distribution (e.g. there may be more frontal views than side views), we can adjust the projection module to incorporate this knowledge. To reduce aliasing, we render each image at 64×64 resolution and downsample to 32×32. We have found that this generally improves the results. Using synthetic data allows us to easily perform controlled experiments to analyze our method. It is also possible to use real images downloaded from a search engine as discussed in Section 7.4.

7.3.1 Results

We quantitatively evaluate our model by comparing its ability to generate 2D and 3D shapes. To do so, we use 2D image GAN similar to DCGAN [143] and a 3D-GAN similar to the one presented in [204]. At the time of this writing the implementation of [204] is not public yet, therefore we implemented our own version. We will refer to them as 2D-GAN and 3D-GAN, respectively. The 2D-GAN has the same discriminator architecture as the PrGAN, but the generator contains a sequence of 2D transposed convolutions instead of 3D ones, and the projection module


(a) Results from 3D-GAN.

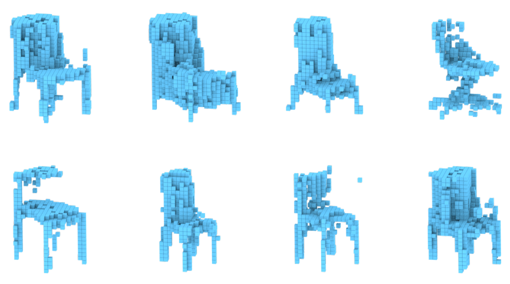

(a) Results from PrGAN.

Figure 7.5: Comparison between 3D-GAN [204] and our PrGAN for 3D shape generation. The 3D-GAN is trained on 3D voxel representation of the chair models, and the PrGAN is trained on images of the chair models (refer to Figure 7.9 third row).

is removed. The 3D-GAN has a discriminator with 3D convolutions instead of 3D ones. The 3D-GAN generator is the same as the PrGAN, but without the projection module.

The models used in this experiment are chairs from ModelNet dataset [206]. From those models, we create two sets of training data: voxel grids and images. The voxel grids are generated by densely sampling the surface and inside of each mesh, and binning the sample points into $32 \times 32 \times 32$ grid. A value 1 is assigned to any voxel that contains at least one sample point, and 0 otherwise. Notice that the voxel grids are only used to train the 3D-GAN, while the images are used to train the 2D-GAN and our PrGAN.

Our quantitative evaluation is done by taking the Maximum Mean Discrepancy [67] (MMD) between the data created by the generative models and the training data. We use a kernel bandwidth of 10^{-3} for images and 10^{-2} for voxel grids. The training data consists of 989 voxel grids and 7912 images. To compute the MMD, we draw 128 random data points from each one of the generative models. The distance metric between the data points is the hamming distance divided by the dimensionality of the data. Because the data represents continuous occupancy values, we binarize them

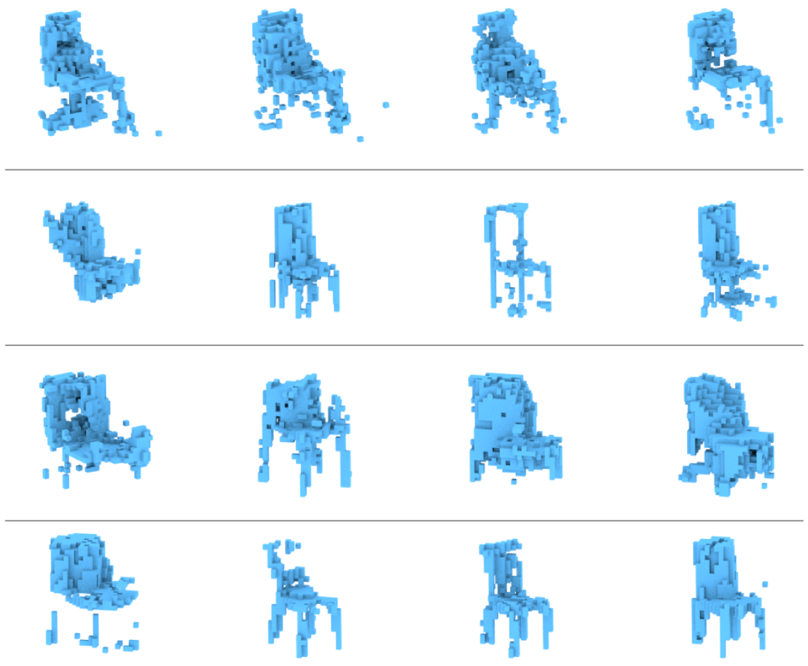


Figure 7.6: Shapes generated from PrGAN by varying the number of views per object in the training data. From the top row to the bottom row, the number of views per object in the training set are 1, 2, 4, and 8 respectively.

by using a threshold of 0.001 for images or voxels created by PrGAN, and 0.1 for voxels created by the 3D-GAN.

Results show that for 2D-GAN, the MMD between the generated images and the training data is **90.13**. For PrGAN, the MMD is **88.31**, which is slightly better quantitatively than 2D-GAN. Figure 7.4 shows a qualitative comparison. The results are visually very similar. For 3D-GAN, the MMD between the generated voxel grids and the training voxel grids is **347.55**. For PrGAN, the MMD is **442.98**, which is worse compared to 3D-GAN. This is not surprising as 3D-GAN is trained on 3D data, while PrGAN is trained on the image views only. Figure 7.5 presents a qualitative comparison. In general PrGAN has trouble generating interior structures because the training images are binary, carry no shading information, and are taken from a limited set of viewing angles. Nonetheless, it learns to generate exterior structures reasonably well.

7.3.1.1 Varying the number of views per model

In the default setting, our training data is generated by sampling 8 views per object. Note that we do not provide the association between views and instances to the generator. Here we study the ability of our method in the more challenging case where the training data contains fewer number of views per object. To do so, we generate a new training set that contains only 1 randomly chosen view per object and use it to train PrGAN. We then repeat the experiments for 2 randomly chosen views per object, and also 4. The results are shown in Figure 7.6. Notice that the 3D shapes generated by PrGAN become slightly better as the number of views increase. An interesting question is what's the root cause for such improvements – it may be due to the fact that more training data is available as the number of views per object increases; or it could be that the presence of multiple views of the same object lead to better reconstruction. Thus we further investigate this question by performing an additional experiment, where the training data consists of 8 views per instance, but only using half of the instances available in the dataset. In other words, this setup has the same number of images as the experiment with 4 views of all instances, which makes them comparable in terms of the total amount of training data. We observed no qualitative or quantitative difference in the objects generated in these two scenarios. Quantitative results using the model and metrics described in Section 7.4 are shown in Table 7.1. Therefore, we believe the improved quality is most likely a consequence of extra data available during training. Nevertheless, it is important to highlight that differently from other approaches that require object correspondence [185, 211] our method is able to induce reasonable shapes, even in the case of a single view per object.

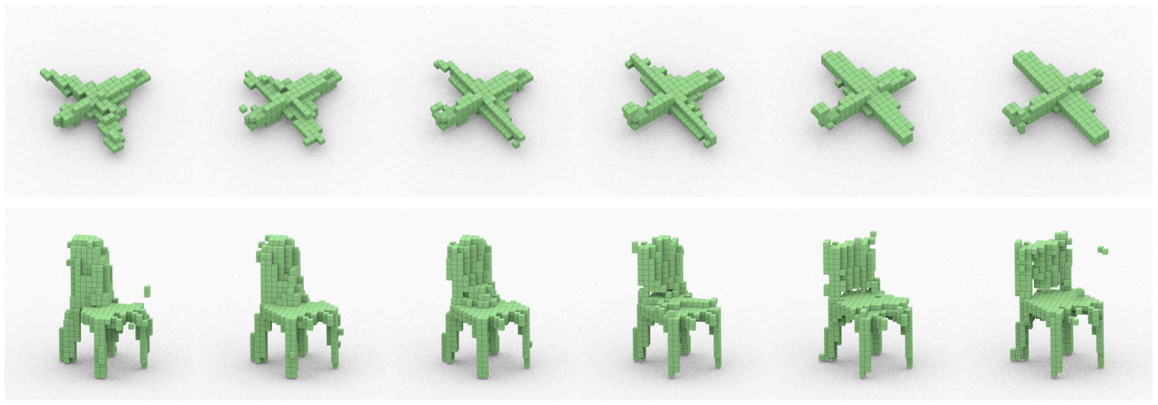


Figure 7.7: Shape interpolation by linearly interpolating the encodings of the starting shape and ending shape.

7.3.1.2 Shape interpolation

Once the generator is trained, any encoding z supposedly generates a plausible 3D shape, hence z represents a 3D shape manifold. Similar to previous work, we can interpolate between 3D shapes by linearly interpolating their z codes. Figure 7.7 shows the interpolation results for two airplane models and two chair models.

7.3.1.3 Unsupervised shape and viewpoint prediction

Our method is also able to handle unsupervised prediction of shapes in 2D images. Once trained, the 3D shape generator is capable of creating shapes from a set of encodings $z \in \mathbb{R}^{201}$. One application is to predict the encoding of the underlying 3D object given a single view image of the object. We do so by using the PrGAN's generator to produce a large number of encoding-image pairs, then use the data to train a neural network (called encoding network). In other words, we create a training set that consists of images synthesized by the PrGAN and the encodings that generated them. The encoding network is fully connected, with 2 hidden layers, each with 512 neurons. The input of the network is an image and the output is an encoding. The last dimension of z describes the view, and the first 200 dimensions describe the code of the shape, which allows us to further reconstruct the 3D shape as a 32^3 voxel grid. With the encoding network, we can present to it a single view image, and it outputs the shape code along with the viewing angle. Experimental results are shown in in Figure 7.8. This whole process constitutes a completely unsupervised approach to creating a model that infers a 3D shape from a single image.

7.3.1.4 Visualizations across categories

Our method is able to generate 3D shapes for a wide range of categories. Figure 7.9 show a gallery of results, including airplanes, car, chairs, vases, motorbikes. For each category we show 64 randomly sampled training images, 64 generated images from PrGAN, and renderings of 128 generated 3D shapes (produced by randomly sampling the 200-d input vector of the generator). One remarkable property is that the generator produces 3D shapes in a consistent horizontal and vertical axes, even though the training data is only consistently oriented along the vertical axis. Our hypothesis for this is that the generator finds it more efficient to generate shapes in a consistent manner by sharing parts across models. Figure 7.10 shows selected examples from Figure 7.9 that demonstrates the quality and diversity of the generated shapes.

The last row in Figure 7.9 shows an example of a “mixed” category, where the training images combine the three categories of airplane, car, and motorbike. The same PrGAN network is used to learn the shape distributions. Results show that PrGAN learns to represent all three categories well, without any additional supervision.

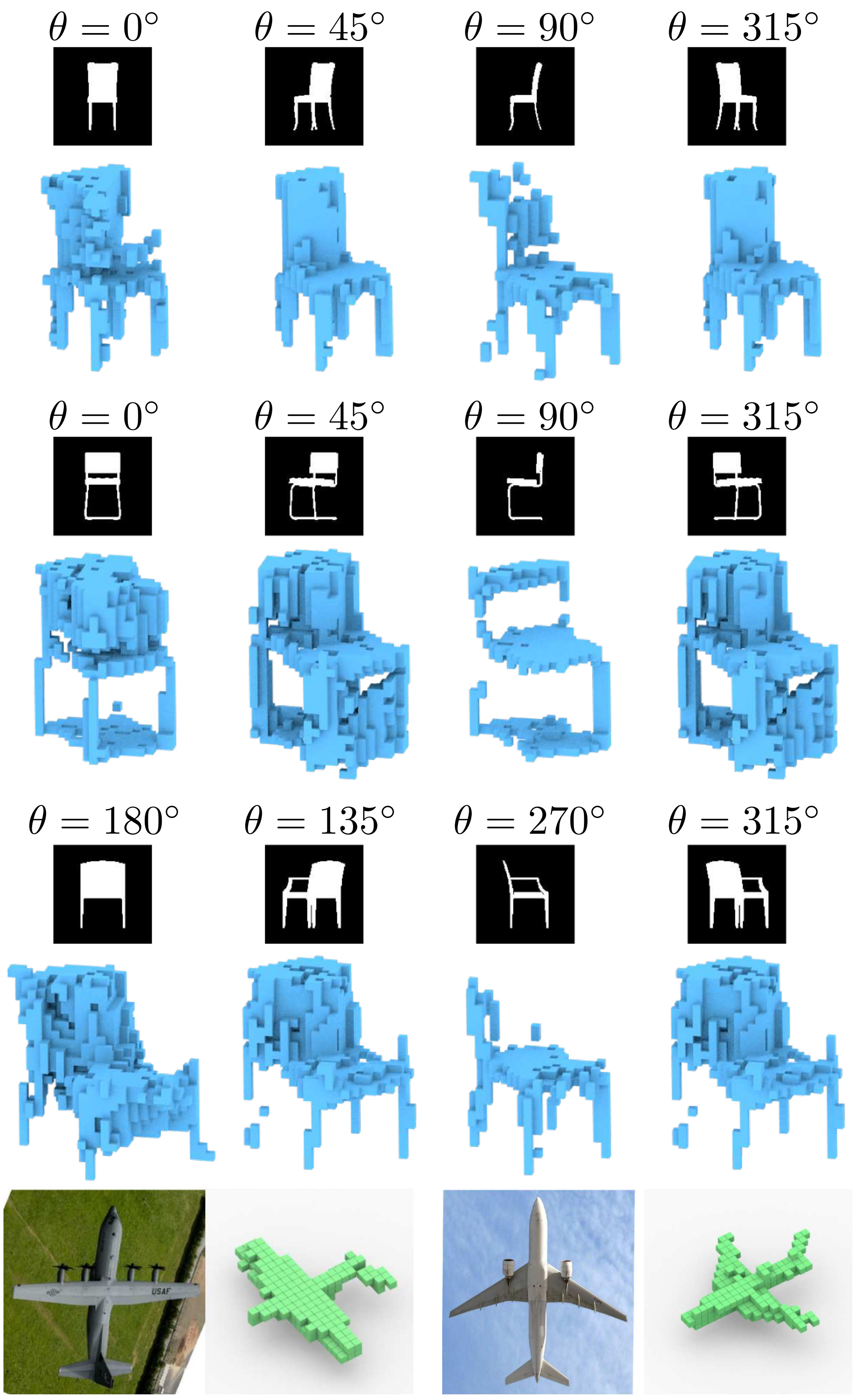


Figure 7.8: At top 3 rows, the four images are different views of the same chair, with predicted viewpoint on the top. Shapes are different but plausible given the single view. In the bottom row, shape inferred (right) by a single view image (left) using the encoding network. Input images were segmented, binarized and resized to match the network input.

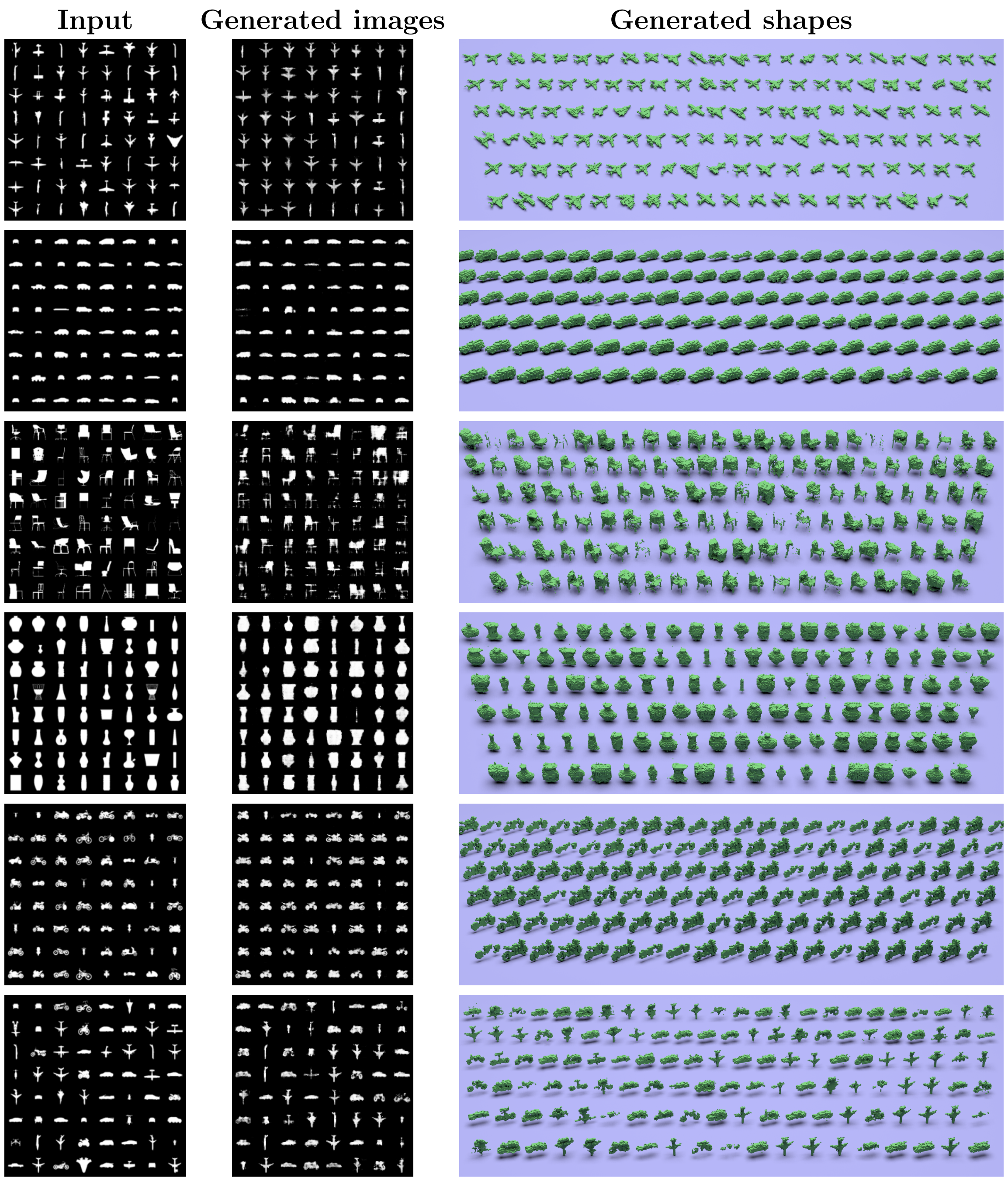


Figure 7.9: Results for 3D shape induction using PrGANs. From top to bottom we show results for airplane, car, chair, vase, motorbike, and a 'mixed' category obtained by combining training images from airplane, car, and motorbike. In each row, we show on the left 64 randomly sampled images from the input data to the algorithm, on the right 128 sampled 3D shapes from PrGAN, and in the middle 64 sampled images after the projection module is applied to the generated 3D shapes. The model is able to induce a rich 3D shape distribution for each category. The mixed-category produces reasonable 3D shapes across all three combined categories. Zoom in to see details.

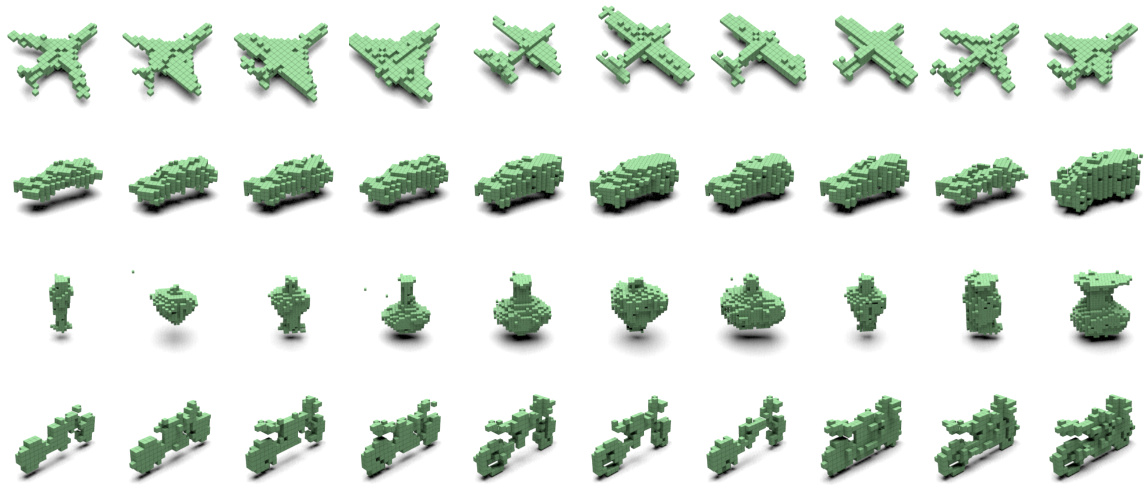


Figure 7.10: A variety of 3D shapes generated by PrGAN trained on 2D views of (from the top row to the bottom row) airplanes, cars, vases, and bikes. These examples are chosen from the gallery in Figure 7.9 and demonstrate the quality and diversity of the generated shapes.

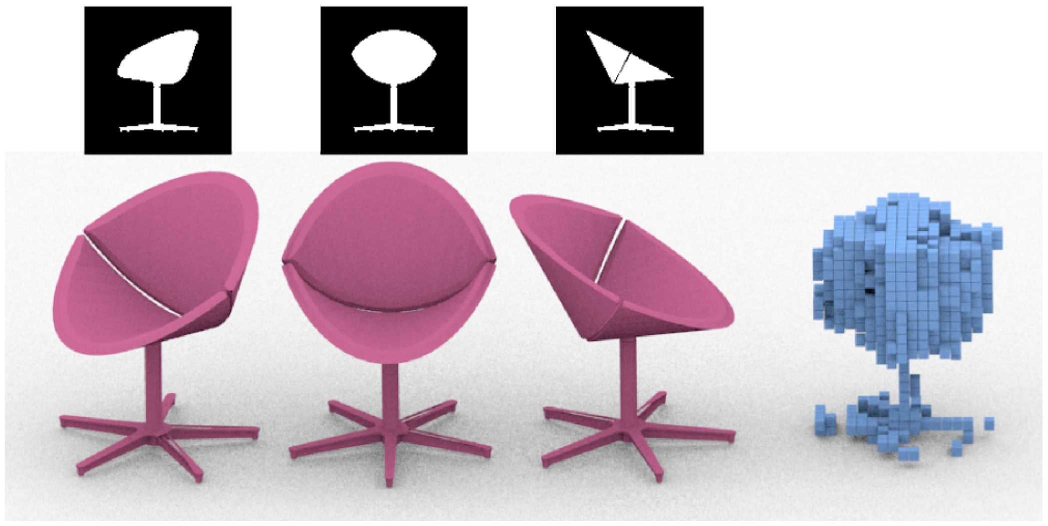


Figure 7.11: Our method is unable to capture the concave interior structures in this chair shape. The pink shapes show the original shape used to generate the projected training data, shown by the three binary images on the top (in high resolution). The blue voxel representation is the inferred shape by our model. Notice the lack of internal structure.

7.3.2 Failure cases

Compared to 3D-GANs, the proposed PRGAN models cannot discover structures that are hidden due to occlusions from all views. For example, it fails to discover that some chairs have concave interiors and the generator simply fills these since it does not change the silhouette from any view as we can see at Figure 7.11. However, this is a natural drawback of view-based approaches since some 3D ambiguities cannot be resolved (*e.g.*, Necker cubes) without relying on other cues. Despite this, one advantage over 3D-GAN is that our model does not require consistently aligned 3D shapes since it reasons over viewpoints.

7.4 Improving PrGAN with richer supervision

This section shows how the generative models can be improved to support higher resolution 3D shapes and by incorporating richer forms of view-based supervision.

7.4.1 Higher-resolution models

We extend the vanilla PRGAN model to handle higher resolution volumes. There are two key modifications. First, we replace the transposed convolutions in the generator by trilinear upsampling followed by a 3D convolutional layer. In our experiments, we noticed that this modification led to smoother shapes with less artifacts. This fact was also verified for image generators [133]. Second, we add a feature matching component to the generator objective. This component acts by minimizing the difference between features computed by the discriminator from real and fake images. More precisely, the feature matching loss can be defined as:

$$\mathcal{L}_{FM}(G, D) = \left\| \mathbb{E}_{x \sim \mathcal{D}}[D_k(x)] - \mathbb{E}_{z \sim \mathcal{N}(0,I)}[D_k(G(z))] \right\|_2^2 \tag{7.2}$$

where $D_k(x)$ are the features from the kth layer of the discriminator when given an input x. In our experiments we define k to be the last convolutional layer of the discriminator. We empirically verified that this component promotes diversity in the generated samples and makes the training more stable.

7.4.2 Using multiple cues for shape reasoning

So far our approach only relies on binary silhouettes for estimating the shape, which contributes to the lack of geometric details. One strategy is replace the projection module with a differentiable function, *e.g.*, a convolutional network, to approximate a sophisticated rendering pipeline, like the one presented in [125, 128]. Such a *neural renderer* could be a plug-in replacement for the *projection module* in the PrGAN framework. This would provide the ability to use collections of realistically-shaded images for inferring probabilistic models of 3D shapes and other properties.

We explore an alternate direction using differentiable projection operators that do not rely on training procedures. This choice fits well in the PrGAN formulation as it does not rely on 3D supervision for training any part of the model. In this section, we present differentiable operators to render depth images and semantic segmentation maps. We demonstrate that the extra supervision enables generating more accurate 3D shapes and allows relaxing the prior assumption on viewpoint distribution.

Learning from depth images. Our framework can be adapted to learn from depth images instead of binary images. This is done by replacing the binary projection operator Pr to one that can be used to generate depth images. We follow an approach inspired by the binary projection. First, we define an accessibility function $A(V, \phi, c)$ that describes whether a given voxel c inside the grid V is visible, when seen from a view ϕ:

$$A(V, \phi, i, j, k) = \exp\Bigg\{ -\tau \sum_{l=1}^{k-1} V_\phi(i, j, l) \Bigg\}. \tag{7.3}$$

Intuitively, we are incrementally accumulating the occupancy (from the first voxel on the line of sight) as we traverse the voxel grid instead of summing all voxels on the entire the line of sight. If voxels on the path from the first to the current voxel are all empty, the value of A is 1 (indicating the current voxel is "accessible" to the view ϕ). If there is at least one non-empty voxel on the path, the value of A will be close to 0 (indicating this voxel is inaccessible). A similar approach was used in our earlier work [56].

Using A, we can define the depth value of a pixel in the projected image as the line integral of A along the line of sight: $Pr_\phi^D(i,j,V) = \sum_k A(V,\phi,i,j,k)$. This operation computes the number of accessible voxels from a particular direction ϕ, which corresponds to the distance of the surface seen in (i,j) to the camera. Finally, we apply a smooth map to the previous operation in order to have depth values in the range $[0,1]$. Thus, the projection module is defined as:

$$Pr_\phi^D((i,j),V) = 1 - \exp\Big\{ -\sum_k A(V,\phi,i,j,k)\Big\}. \tag{7.4}$$

Learning from part segmentations. We also explore learning 3D shapes from sets of images with dense semantic annotation. Similarly to the depth projection, we modify our projection operator to enable generation of images whose pixels correspond to the label of particular class (or none if there is no object). In this case, the output of the generator is multi-channel voxel grid $V : \mathbb{Z}^3 \times C \to [0,1] \in \mathbb{R}$, where C is the number of parts present in a particular object category.

Let G to be the aggregated occupancy grid defined as $G = \sum_{c=1}^{C} V(i,j,k,c)$. The semantic projection operator $Pr_\phi^S((i,j,c),V)$ is defined as:

$$Pr_\phi^S((i,j,c),V) = 1 - \exp\Big\{ \sum_k V_\phi(i,j,k,c) A(G_\phi,i,j,k)\Big\}, \tag{7.5}$$

where A is the accessibility operator defined previously. Intuitively, $A(G,\phi)$ encodes if a particular voxel is visible from a viewpoint ϕ. When we multiply the visibility computed with the aggregated occupancy grid by the value of a specific channel c in V, we generate a volume that contains visibility information per part. Finally, we

Model	Supervision	$\mathcal{D} \to G(z)$	$G(z) \to \mathcal{D}$	Avg.
PrGAN*	Silhouette	0.431	0.391	0.411
PrGAN†	Silhouette	0.429	0.391	0.410
PrGAN	Silhouette	0.442	0.400	0.421
PrGAN	Silhouette + View	0.439	0.431	0.435
PrGAN	Depth	0.497	0.448	0.472
PrGAN	Part Segmentation	0.496	0.507	0.502
3D-GAN	Volumetric	0.538	0.530	0.534

Table 7.1: Quantitative comparison between models trained with different projection operators. The Chamfer similarity under the volumetric intersection over union (IoU) is shown for PrGAN trained with varying amounts of supervision and a 3D-GAN trained with volumetric supervision. The metric (higher the better) indicates that PrGAN with richer supervision are better and approaches the quality of 3D-GAN. PrGAN* is trained using only 4 out of 8 views per object. PrGAN† is trained using all 8 views but for half of the objects.

take the line integral along the line of sight to generate the final image. Examples of images and shapes generated by this operator can be seen in Figure 7.12.

Learning with viewpoint annotation. We also experiment with the less challenging setup where our model has access to viewpoint information for every training image. Notice that this problem is different from [89, 211], since we still do not know which images correspond to the same object. Thus, multi-view losses are not a viable alternative. Our model is able to leverage viewpoint annotation by using conditional discriminators. The conditional discriminator has the same architecture as the vanilla discriminator but the input image is modified to contain its corresponding viewpoint annotation. This annotation is represented by an one-hot encoding concatenated to every pixel in the image. For example, if a binary image from a dataset with shapes rendered from 8 viewpoints will be represented as a 9-channel image. This procedure is done for images generated by our generator and images coming from the dataset.

7.4.3 Experiments

Setup. We generate training images using airplanes from the ShapeNet part segmentation dataset [24]. Those shapes have their surface densely annotated as belonging to one of four parts: body, wing, tail or engine. We render those shapes using the same viewpoint configuration described in Section 7.3. However, in this scenario we use 64×64 images instead of 32×32. The models are rendered as binary silhouettes, depth maps and part segmentation masks. We train a high resolution PrGAN model for every set of rendered images using the corresponding projection operator. Each model is trained for 50 epochs and trained with the Adam optimizer. We use a learning rate of 2.5×10^{-3} for the generator and 2×10^{-5} for the discriminator.

Evaluation. The models trained with different visual clues are evaluated through the following metric:

$$\frac{1}{|\mathcal{D}|}\sum_{x\in\mathcal{D}}\min_{g\in\mathcal{G}} IoU(x,g) + \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}\min_{x\in\mathcal{D}} IoU(x,g) \tag{7.6}$$

where IoU corresponds to intersection over union, $\mathcal{G}$ is a set of generated shapes and $\mathcal{D}$ is a set of shapes from the training data. In our setup, both $\mathcal{G}$ and $\mathcal{D}$ contain 512 shapes. Shapes in $\mathcal{D}$ are randomly sampled from the same dataset that originated the images, whereas shapes in $\mathcal{G}$ are generated through $G(z)$. Noticeably, the shapes generated by PrGAN do not have the same orientation as the shapes in $\mathcal{D}$ but are consistently oriented among themselves. Thus, before computing Equation 7.6, we select one of 8 possible transformations that minimizes IoU – there are 8 rendering viewpoints in the training set. Additionally, the components in Equation 7.6 indicate two different aspects: the first term ($\mathcal{D} \to G(z)$) indicates how the variety in the dataset is covered whereas the second term ($G(z) \to \mathcal{D}$) indicates how accurate the generated shapes are. A comparison between models trained with different projection operators can be seen in Table 7.1. The model trained with part segmentation clues yields the best results. As expected, using only silhouettes leads to worse results in both metrics and adding viewpoint supervision improves upon this baseline. Interestingly, depth and part segmentation supervision clues lead to models that generate shapes with similar variety (similar $\mathcal{D} \rightarrow G(z)$). However, shapes generated from models using part segmentation clues are more similar to the ones in the dataset (higher $G(z) \rightarrow \mathcal{D}$).

7.5 Conclusion and Future Work

We proposed a framework for inferring 3D shape distributions from 2D shape collections by augmenting a convnet-based 3D shape generator with a projection module. This complements existing approaches for non-rigid SfM since these models can be trained without prior knowledge about the shape family, and can generalize to categories with variable structure. We showed that our models can infer 3D shapes for a wide range of categories, and can be used to infer shape and viewpoint from a silhouettes in a completely unsupervised manner. We believe that the idea of using a differentiable render to infer distributions over unobserved scene properties from images can be applied to other problems.

A limitation is that our approach cannot directly learn from real-world images, as they usually have background pixels, and contain complex shading. In the future, our method can be extended to accommodate real images using semantic segmentation to extract foreground object from the background. In addition, it is possible to incorporate photorealistic differentiable rendering modules capable of handling richer surface colors, materials, and camera parameters. One could also incorporate other forms of supervision, such as viewpoint or coarse shape estimates, to improve the 3D shape inference. For example, camera parameters can be estimated using a generic viewpoint estimator [170, 184].

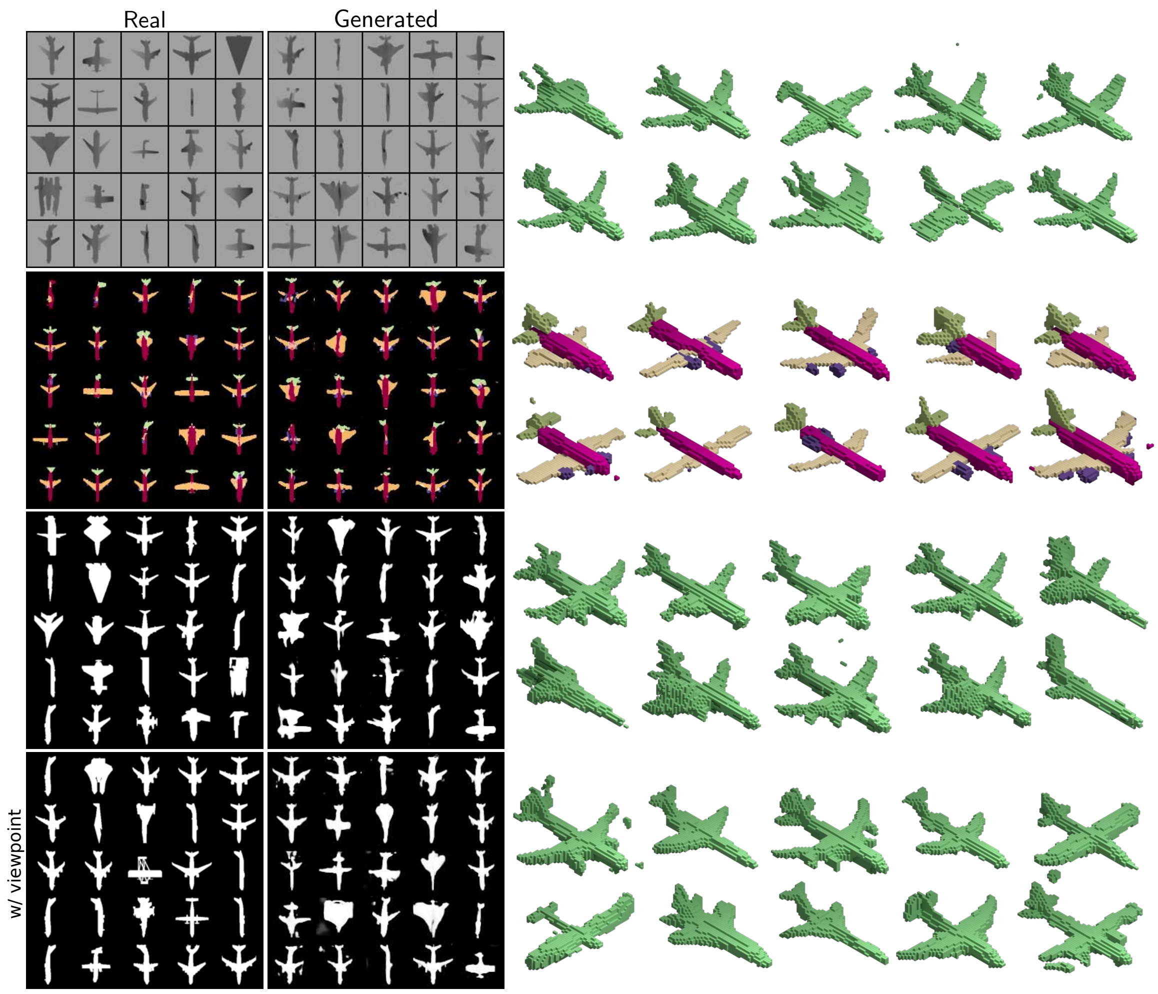


Figure 7.12: Shapes generated using new part segmentations and depth maps. From top to bottom, results using depth images, images with part segmentation, silhouettes and silhouettes annotated with viewpoints. Models trained with images containing additional visual cues are able to generated more accurate shapes. Similarly, viewpoint annotation also helps. Notice that shapes generated from images with part annotation are able to generate part-annotated 3D shapes, highlighted by different colors.

CONCLUSION AND FUTURE WORK

This thesis presented a set of models and learning techniques for 3D data generation and understanding. It focused on two fundamental aspects: *models* and *data*. In the first three chapters, this thesis presented techniques for handling irregularly structured representations, i.e. 3D shapes are described as permutation-invariant sets of geometrical entities. For shape recognition and reconstruction, we presented architectures for dealing with point clouds. We showed that one can circumvent their irregular nature by automatically arranging them in spatial partitioning data structures. Those data structures allow us to efficiently represent point clouds using simple linear basis and define multi-resolution convolutional operators. Those operators can then be used as neural network building blocks in several scenarios: point cloud classification, segmentation, single view reconstruction and within variational auto-encoders. However, point clouds are not a very good representation if user wants to manipulate the shape. To this end, this thesis also presented a class of models capable of directly generating shape handles. Those generative models are quite flexible and can be easily applied to tasks like shape completion, interpolation and editing using various types of handles. Both point clouds and shape handles fall under the broader category of irregularly structured (or set-based) representations. In comparison to the more common multi-view and occupancy grid counterparts, both of them have remarkable memory and computational efficiency.

In the remaining chapters, this thesis focused on the other big issue concerning learning models for 3D reasoning: the lack of 3D data itself. To this end, we presented several approaches to address different types of data scarcity. When 3D data exists, but it is unlabeled, we proposed a self-supervised task based on approximate convex decomposition. We showed that when few labels are available, our proposed self-supervised learning task significantly outperforms the state-of-the-art in unsupervised shape classification and few-shot part segmentation. For shape generation, the lack of 3D supervision increases the importance of the priors induced by the different models and representations. We analyzed the priors induced by neural networks while parametrizing manifolds and showed how those can be applied to 2D and 3D manifold reconstruction. We further demonstrated how understanding these priors can improve learning-based tasks by proposing regularizations and convolutional architectures that encode these priors more efficiently. We also investigated the priors induced by convolutional architectures while generating regular occupancy grids. More importantly, we showed how those volumetric representations can be coupled with differentiable projection operators to enable learning directly from posed images. However, the most challenging setup arises when no 3D data is available and there is no pose information. In this scenario, we only have access to images without any object identification. This problem was tackled by employing a modified generative adversarial network which generates 3D shapes and projects them to 2D using the aforementioned projection operators. We demonstrated how this framework can lead to learning 3D generative models without access to 3D shape, object or keypoint annotation.

Future Work

Models and techniques for 3D shape understanding and generation experienced remarkable progress in the last few years. The next paragraphs discuss some interesting research challenges that follow the contributions of this thesis and the current state of the field as a whole.

Learning from real images requires more than geometry. The projection modules developed in our research can be seen as simple differentiable renderers. The ultimate goal is not only be able to learn 3D from silhouettes but from photorealistic images. Computer graphics research has developed a solid toolset for synthesizing realistic images. Combining those techniques with good 3D shape representations will lead to better models, capable of learning not only 3D geometry but material properties, like color, textures and BRDFs, within a single framework. Models like these have many applications in image and 3D editing and are paramount to aid the creation of photorealistic content and 3D manipulation of 2D images. Perhaps more importantly, techniques capable of reasoning about various components of the image formation process can lead to better understanding of the inductive biases needed to perform more efficient learning in various scenarios. In other words, how can one design models that induce good priors for other scene components besides geometry, like materials and illumination?

Models for Dynamic 3D Data. The main motivation of my research comes from the intuition that reasoning with images is suboptimal when interacting with the real tridimensional world. However, for many applications, considering a static 3D world is also a crude approximation. After significant progress in developing models for static objects, a natural next step is to think about problems concerning dynamic 3D scenes. This requires rethinking many decisions concerning architectures and representations for 3D data – a solution that simply adds an extra dimension is clearly not the best one. More than that, foundational work creating datasets for these types of problems is required. Models that are capable of reasoning about motion in 3D already have a clear impact in autonomous navigation. Models that can generate moving data in 3D will likely not only have a similar impact in creative applications, but may also lead to efficient 3D video representations.

Full-Scene Reconstruction. Techniques that yield the best single-view reconstruction results are usually applied in images containing a single object. While a lot of progress has been made in this setup, it is time to move on to more interesting and realistic scenarios. A natural extension is to build upon consolidated object detection pipelines and utilize single-object reconstruction models in the process of reconstructing full scenes containing multiple objects. However, such model should not only be able to reconstruct 3D from single-object proposals but also utilize contextual scene information to guide its reconstruction. Whereas seminal work has been done in this area, there is still a significant gap between the quality of reconstructing single objects when compared to full scenes. Closing this gap requires investigating efficient 3D representations and building datasets with full 3D annotation. Reconstructing high-quality full 3D scenes from as few images as possible (maybe just one) is a key component for developing the next generation of robots and augmented reality applications.

Differentiable Programming for Computer Aided Design. Using differentiable rendering to obtain supervisory signal for learning 3D shapes has shown remarkable success. The main intuition behind those approaches is that, since we know how to simulate the image formation process, one can devise differentiable approximations and insert them into the training loop. However, there are many other domains where such approaches can be applied. For example, if one can differentiably simulate agent behavior and interaction with objects, the signal can be used to guide many design tasks. More than that: artists and game designers often rely on graph-based descriptions of materials, shaders and game logic. Implementing differentiable approximations to these workflows would allow learning from data more efficiently and recovering interpretable latent representations, crucial for allowing meaningful manipulation by practitioners. These differentiable engines have the potential to become the cornerstone of accelerated content creation workflows for games, visual effects and 3D design in general.

APPENDIX A

MULTIRESOLUTION TREE NETWORKS - ADDITIONAL EXPERIMENTS

A.1 Shape segmentation model

As mentioned in the Chapter 2, MRTNet can also be applied to shape part segmentation. Given an input shape represented as a point cloud, the task is to segment the shape into meaningful parts. For example, a chair shape may be segmented into back, seat, leg parts etc. Here we describe the segmentation model and experimental results. The segmentation model follows a similar network to the MR-VAE, with two main differences:

1. We add skip-connections, similar to what's used in UNets [148] for image segmentations. Specifically, each intermediate tensor in the MR-Encoder is concatenated to the tensor of the same size in the MR-Decoder.

2. The output point cloud dimensionality is changed to 50 (instead of 3), i.e. each point is described by a part classification score for each of the 50 total possible parts covering the 16 object categories in ShapeNet.

To spatially sort the input point cloud, we use the RPtree, which according to our experiments leads to slight improvement for segmentation than using KDtree. Specifically, the splitting axes are precomputed random vectors sampled uniformly over the unit sphere. The same set of splitting axes are used for all shapes. In other words, it's almost the same as the vanilla KDtree except the axes are chosen as random vectors instead of xyz. This ensures a consistent ordering and relationship between neighboring points, which facilitate a dense classification task like part segmentation.

#instances			2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
	mean class	mean object	air-planes	bag	cap	car	chair	ear-phone	guitar	knife	lamp	laptop	motor-cycle	mug	pistol	rocket	skate	table
MRTNet	79.3	83.0	81.0	76.7	87.0	73.8	89.1	67.6	90.6	85.4	80.6	95.1	64.4	91.8	79.7	57.0	69.1	80.6
KDNet [96]	77.4	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
PointNet [169]	80.4	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
3DCNN [169]	74.9	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
Yi, 2016 [216]	79.0	81.4	81.0	78.4	77.7	75.7	87.6	61.9	92.0	85.4	82.5	95.7	70.6	91.9	85.9	53.1	69.8	75.3

Table A.1: Shape segmentation results. Numbers reported here are the mean intersection over union (mIoU) scores. The table shows comparisons between methods that use 3D position information without normal information.

Experiments. We trained our segmentation model described above on the ShapeNet part segmentation benchmark, which contains groundtruth part labels for 16,881 shapes covering 16 different categories annotaded with 50 parts in total. Each object has 2 to 6 parts. Since this dataset is already represented as point clouds, it is not necessary to perform any additional point sampling. However, each shape may contain a varying number of points, so for each shape we randomly duplicate existing points until we have 4096 points. We follow the evaluation protocol used in prior work [96, 169], where labels that do not appear in a particular category are ignored during evaluation. In other words, only the predictions corresponding to labels of a particular category are used.

Similar to the classification network, the segmentation network is trained by minimizing a cross-entropy using an Adam optimizer with learning rate 10^{-3} and $\beta = 0.9$. The learning rate is also divided by 2 every 5 training epochs. The first three convolutional layer have 32 filters and the next three have 64. We also apply test-time anisotropic scale augmentation. The scaling factors are sampled uniformly at random from the interval $[0.8, 1.2]$. At inference time, we compute 16 different scaled versions of the point cloud and return the mean classification score for each point.

Figure A.1 shows qualitative results from our part segmentation, and Table A.1 lists the mean intersection over union (mIoU) results for all 16 categories. Our approach produces competitive results in comparison with the state of the art. In particular, it outperforms the recently proposed KDNet by nearly 2% in the mean per category mIoU. However, it lacks behind some other recent works, like PointNet [169].

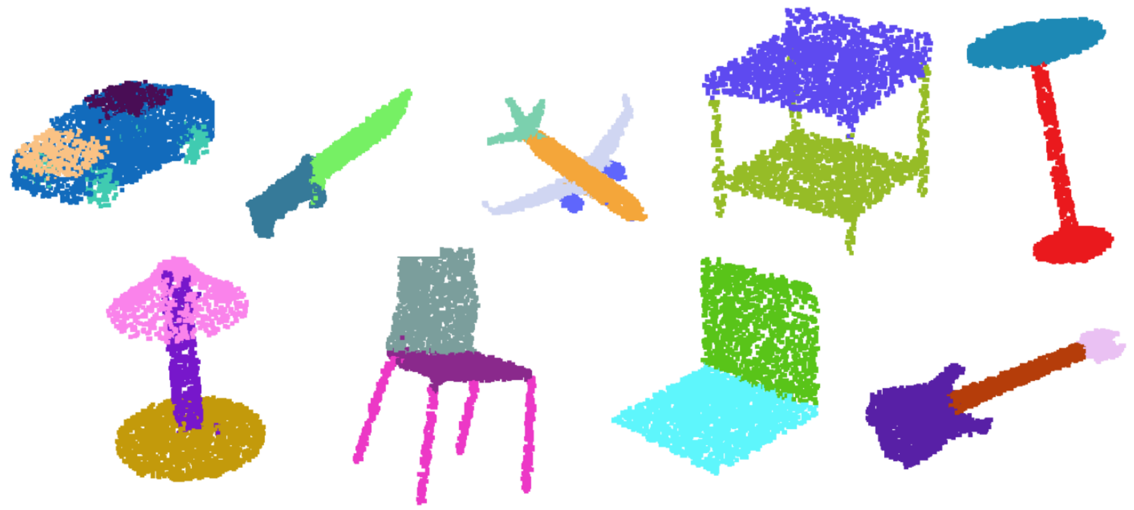


Figure A.1: Qualitative results for the shape segmentation task. Each different segmentation part is shown in a different color.

In comparison to the multiresolution UNets, the single resolution counterpart leads to a drop in about 1%. We believe that the drop here is much smaller in comparison to the classification task because the multiresolution representation is replicating similar benefits to UNet, where information from different scales is added directly to the decoder. In order to verify this hypothesis we trained a segmentation network without the skip connections from UNet. The multiresolution model obtained an accuracy of 79.82%, while the baseline had 76.14%, which is 3% lower.

Increasing filter size in the single resolution network did not yield any benefits. For example, increasing the kernel size from 2 to 8, led to a drop of about 1% in performance. This demonstrates that the multiresolution network adds more than simply increasing the receptive fields.

A.2 Image-to-Shape Inference: Additional Results

In this section, we show additional results of image-to-shape inference using MRT-Net.

Category	MRTNet	Single Res.	Fully Connected
airplane	**0.976** / **0.920**	1.142 / 1.185	1.258 / 1.423
bench	**1.438** / **1.326**	1.616 / 1.666	1.753 / 2.358
cabinet	**1.774** / **1.602**	1.913 / 1.916	1.980 / 2.263
car	**1.395** / **1.303**	1.511 / 1.476	1.583 / 1.668
chair	**1.650** / **1.603**	1.789 / 1.927	1.982 / 2.385
display	**1.815** / **1.901**	2.060 / 2.301	2.185 / 3.029
lamp	**1.944** / **2.089**	1.953 / 2.608	2.163 / 3.400
speaker	**2.165** / **2.121**	2.336 / 2.456	2.374 / 2.913
rifle	**1.029** / **1.028**	1.191 / 1.213	1.291 / 1.402
sofa	**1.768** / **1.756**	1.885 / 2.022	1.985 / 2.421
table	**1.570** / **1.405**	1.689 / 1.562	1.808 / 2.049
telephone	**1.346** / **1.332**	1.637 / 1.677	1.643 / 2.342
watercraft	**1.394** / **1.490**	1.482 / 1.792	1.703 / 2.202
mean	**1.559** / **1.529**	1.708 / 1.831	1.824 / 2.297

Table A.2: Single-image shape inference. Full results of the ablation studies covering all 13 categories. Note that MRTNet is consistently better than both baselines.

Reconstructed meshes. For some applications, such as 3D printing, the output is required to be triangle meshes instead of just point clouds. To do so, we take the image-to-shape inference results (each shape containing 4K points), create a 0.025^3 cube centered at each point, then apply Poisson Surface Reconstruction to create triangle meshes. This is a simple way to create a mesh from a point cloud, without normal estimation (due to the relatively low point count). Figure A.2 shows renderings of the reconsturcted meshes. The input images are Internet photos from Figure 6 in the paper. We rendered each mesh in wireframe mode to reveal the underlying triangle meshes. Some geometric details in the point clouds are necessarily smoothed out due to surface reconsturctions. Nonetheless, the reconsturcted meshes are reasonably faithful to the input images.

Full ablation studies. In the paper, for image-to-shape inference experiments, we presented a summary of ablation studies of MRTNet in comparison with a single-resolution baseline, and a fully-connected baseline. Here we present the full ablation study results, covering all 13 shape categories. Refer to Table A.2. Note that MRTNet is consistently better than both baselines.

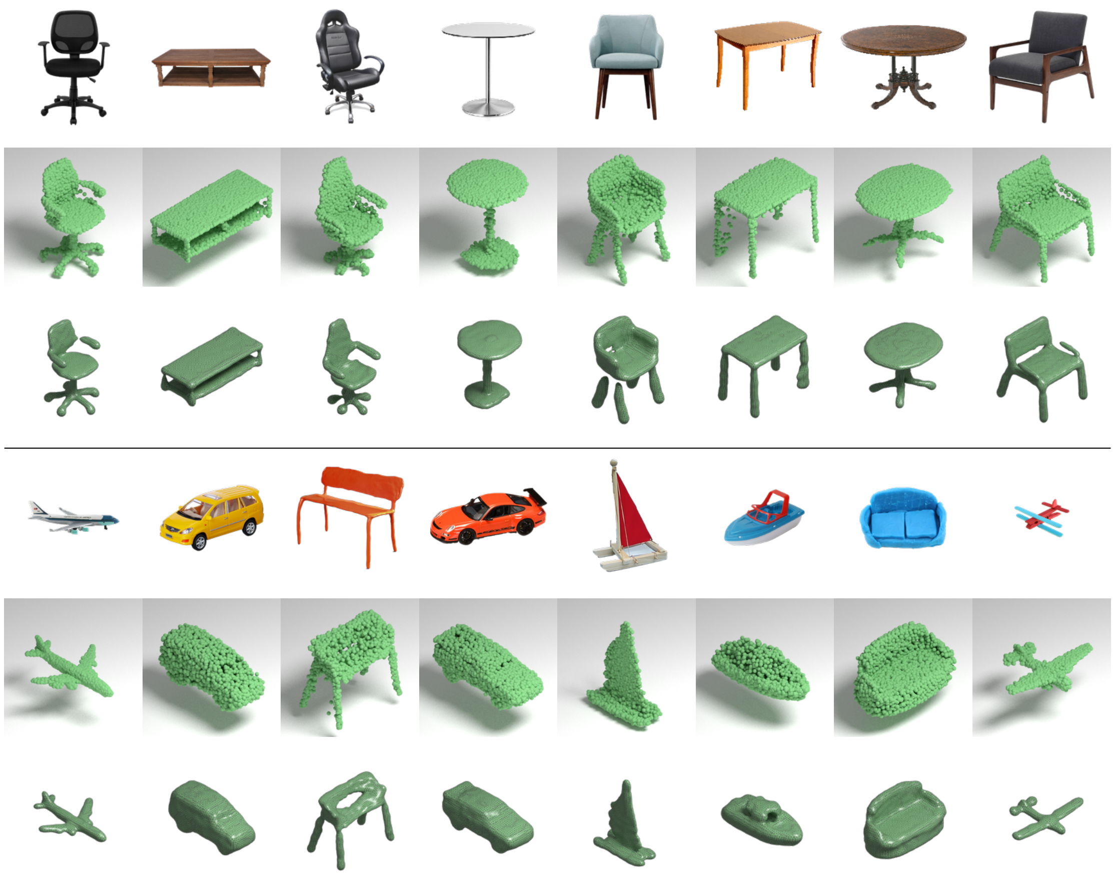


Figure A.2: Reconsturcted meshes from point clouds generated by applying MRTNet on Inernet photos of furnitures and toys. From top-down, the first image in each example is the input photo, the second is the point cloud (each 4K points) generated by MRTNet, the third is a rendering of the reconstructed mesh. We rendered each mesh in wireframe mode to reveal the underlying triangles. Zoom in for details.

A.3 Unsupervised Shape Generation (MR-VAE): Additional Results

In this section, we show additional results of unsupervised shape generation.

Comparison with fully connected decoder. We experimented using fully-connected (FC) decoders as a baseline for MR-VAE. The generated shapes have a much lower quality: see results in Fig. A.3. This is not surprising as there is no prior work that employs these types of decoders alone. Solutions like [49] use FC layers, but as an additional branch to a much deeper convolutional architecture.

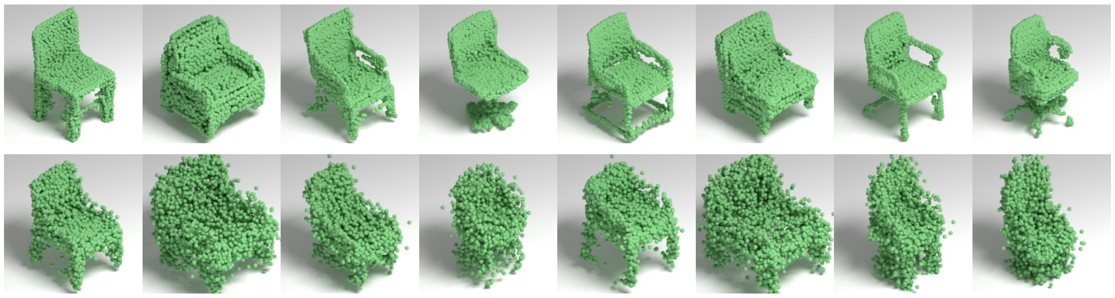


Figure A.3: Chairs generated by randomly sampling the encoding. Top: results from our MR-VAE; Bottom: results by using a fully connected (FC) baseline.

Besides, generating shapes by sampling from a random noise is much harder than reconstructing input shapes. The shapes presented in Fig. A.3 are all **sampled** from a random noise and **not** reconstructions (which would be easy to generate at high quality). Moreover, we are generating shapes with 4K points, which is more difficult for a vanilla decoder, because it tends to generate a lot of misplaced points, as we can see in Fig. A.3 bottom row. Similarly, single resolution decoders produce low-quality samples as seen in Fig. 7 bottom row (in the main paper).

Finally, note that PointNet and Kd-net cannot be used as **decoders** since the former ignores the ordering of points while the later conditions the processing on the splits.

Visualization of MR-VAE encodings using t-SNE. In Figure A.4 we show visualizations of encodings learned by MR-VAE using t-SNE. Specifically, we randomly selected 1000 shapes from the ShapeNet dataset, computed their encodings learned by MR-VAE, then applied t-SNE to compute the 2D coordinate of each encoding, and finally rendered all 1000 shapes on a 2D plane. From the results, we observe that similar shapes tend to stay together, indicating the ability of MR-VAE on learning the latent representations of the shapes.

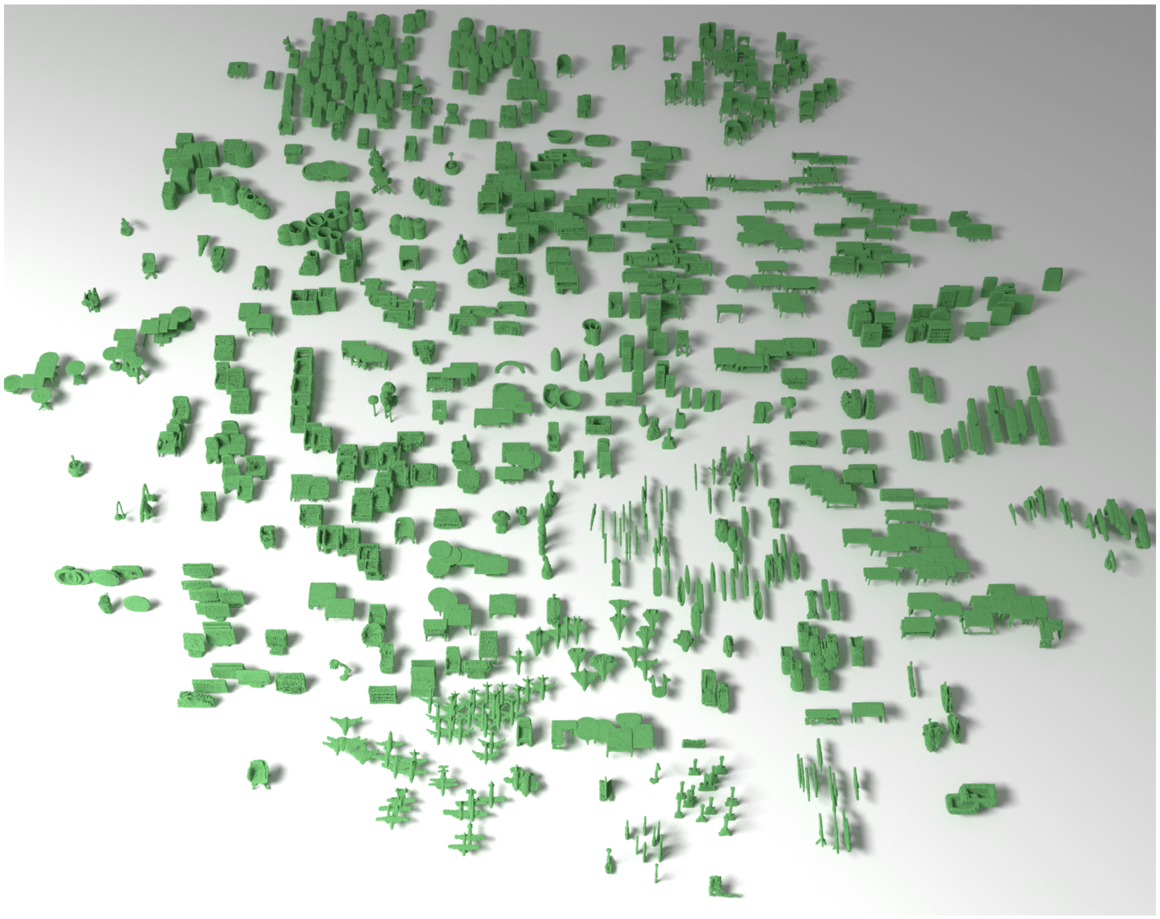


Figure A.4: ShapeNet shapes arranged according to their encodings learned by MR-VAE. 1000 samples from our model trained in the ShapeNet data. The position of the models in the plane is computed after running t-SNE on the latent representation of the shapes. *Zoom in for details.*

APPENDIX B

LEARNING GENERATIVE MODELS OF SHAPE HANDLES - EXTRA RESULTS

B.1 Additional Completion results

In this section, we present additional results for the handle completion task. This task consists of creating a set of handles that represents a plausible shape *and* contains the handles in a given *incomplete* set. Here, we explore the challenging setup where the incomplete set of handles (A) contains a single element. We solve this problem by minimizing Equation 3.9 through gradient descent, which corresponds to finding the latent representation z^* that minimizes the coverage loss $C(z, A)$. Results are presented in Figure B.1. We are able to generate multiple completion proposals (in blue) for each incomplete set of handles (in orange) by minimizing $C(z, A)$ starting from different values of z, initially sampled from a standard gaussian distribution $\mathcal{N}(0, I)$. As we can see in Figure B.1, the "completed" set of handles corresponds to plausible shapes while approximating the elements in the incomplete set.

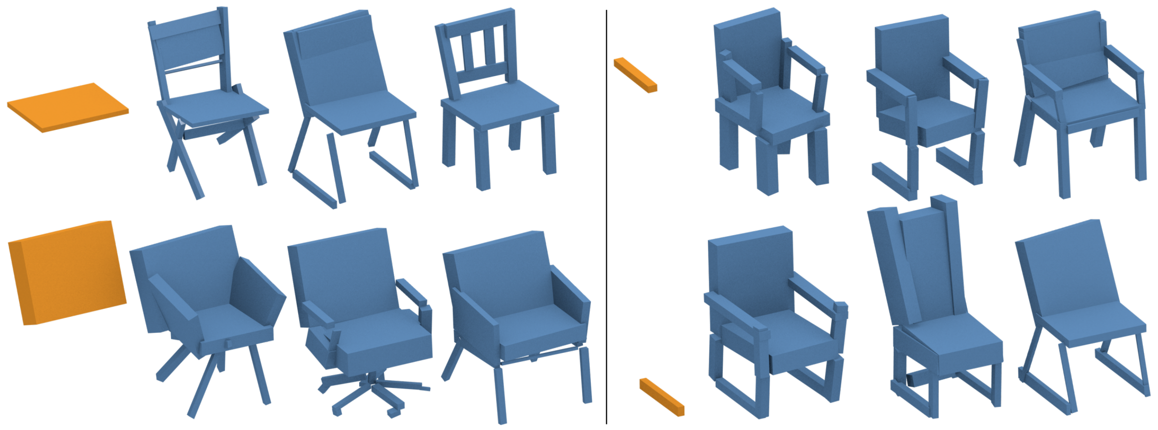


Figure B.1: Additional completion examples. Given an incomplete set of handles (orange cuboids), we solve the optimization problem described in Equation 8 (main paper) using gradient descent. Using different starting points for z^*, our model is capable of computing multiple plausible results that represent a complete shape and closely approximates the incomplete set of handles.

APPENDIX C

DEEP MANIFOLD PRIOR - CONVOLUTIONAL PARAMETRIZATIONS AND ADDITIONAL ANALYSIS

C.1 Convolutional Parametrizations

In the main paper, we experimented with fully connected architectures for representing manifold parametrizations. However, parametrizations represented by convolutional architectures also induce a prior useful for manifold reconstruction tasks. In this section, we show experiments with denoising and single-view reconstruction. We start by defining a `ConvBlock`, which consists of a bilinear upsampling layer followed by a 2D-conv, batch normalization [81] and Leaky ReLU activation (slope=0.2). Every convolutional layer uses filter size 3×3, stride 1 and the number of filters is exactly half the number of its input channels. In other words, at every `ConvBlock`, the output tensor spatially doubles the size of its input tensor, but only has half the number of channels. This pattern follows throughout the whole network, except for the last layer, where the output layer always have 3 channels, representing the (x, y, z) point coordinates.

C.1.1 Denoising

The denoising experiments follow the same procedure described in the main paper, except for the network architecture. Instead of using a fully connected model, we employ a network with 3 `ConvBlock`s, starting from an input tensor with shape $4 \times 4 \times 512$ whose values are drawn from a standard gaussian distribution. The output of each parametrization is a tensor with shape $32 \times 32 \times 3$, which we can treat as a point cloud with 1024 and use Chamfer distance in the same way as described in

Section 5. We also use the position of the points in the output tensor to define the local neighborhood utilized in the stretch regularization. Results are presented in Figure C.1. As we can see, convolutional parametrizations also induce a useful prior for manifold reconstructions and, similarly to the other parametrizations, it is significantly better than the baselines. Quantitatively, using convolutional parametrizations in the denoising yields slightly worse results than using fully connected networks – in terms of Chamfer distance, 4.58×10^{-4} vs. 4.48×10^{-4}.

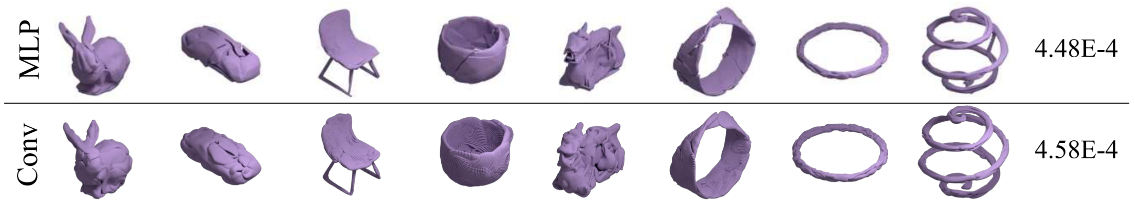


Figure C.1: Comparison of Conv and MLP networks for denoising. Average error across shapes to the right. Both models use 8 parametrizations and stretch regularization. Zoom for details.

C.1.2 Single-view Reconstruction

In this subsection we present quantitative and qualitative results for single-view image-to-shape using convolutional paramterizations. We also train a convolutional decoder with stretch regularization on the single-view reconstruction benchmark [35]. This follows the same experimental setup as previous papers [35,49,55,68]. However, unlike AtlasNet [68], our network is trained in one stage, without the need to train the decoder in an auto-encoder setting before fine-tuning it with an image CNN in a second step. We used Adam optimizer [93] with learning rate of 10^{-3}. The model is trained for 40 epochs and the learning rate is divided by 2 every 5 epochs. We use ResNet-18 as image encoder and 32 convolutional parameterizations. Even though we use more parameterizations than AtlasNet, the total number of parameters is smaller (see Table C.3. The evaluation results per category are presented in Table C.1. Compared to MRTNet, our model performs better in 12 out of 13 categories. Compared

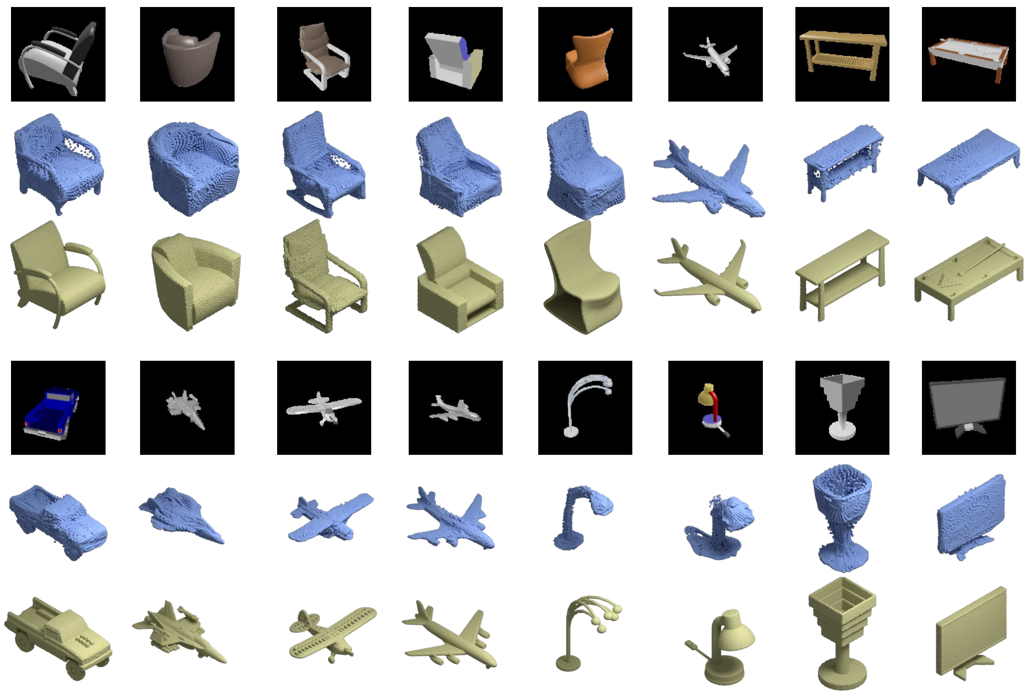


Figure C.2: Image-to-shape reconstruction results from the test set. The images shown are the input (black background), our results (32K points, rendered blue), and ground truth (rendered in light green). Qualitatively, our method is able to generate high-resolution point clouds faithfully capturing fine geometric details such as the chair legs, arms, airplane engines, monitor stands etc.

to AtlasNet, our method is better or ties (the firearm category) in 7 out of 13 categories. Overall our approach outperforms AtlasNet in per-category mean by 0.21, a relative improvement of 4.4%. Also note that our model outperforms AtlasNet mainly in categories with a large number of examples (tables, cars, airplanes, chairs). As a result, if average over instances, our method has a per-instance mean of 4.0, vs. 4.38 by AtlasNet – a relatively improvement of 8.7%.

	pla.	ben.	cab.	car	cha.	mon.	lam.	spe.	fir.	cou.	tab.	cel.	wat.	**mean**
AtlasNet [68]	2.17	**3.39**	**3.93**	3.40	4.56	**5.05**	12.24	8.79	**2.15**	**4.58**	4.15	**3.25**	3.93	4.74
MRTNet [55]	2.25	3.68	4.73	**2.55**	4.06	6.07	11.15	8.84	2.25	4.98	4.45	3.72	3.64	4.80
Ours (32 dec.)	**2.06**	3.40	4.46	2.60	**3.76**	5.94	**10.66**	**8.38**	**2.15**	4.64	**3.96**	3.45	**3.40**	**4.53**

Table C.1: Quantitative results for single-view image-to-shape reconstruction. The table reports Chamfer distance metric (scaled by 10^3) computed per category, and the mean of all categories. For each method 4K points were used to compute the distance.

Architecture	mean/cat.	mean/inst.
MRTNet	4.80	4.26
1 dec./vgg16/4k	4.85	4.30
1 dec./res18/16k	4.75	4.22
32 dec./res18/32k	**4.53**	**4.00**

Table C.2: Architecture variations and evaluation results. The table reports per-category mean and per-instance mean for MRTNet, and three variations of our methods: single decoder with 4K output points, 16K output points, and 32 decoders with 32 output points. For all cases, the Chamfer distance is calculated using 4K sample points, and results are scaled by 10^3.

Ablation studies. Table C.2 shows a quantitative comparison between a few architectural variations. We start by analyzing a variation of our network that generates the same number of points (using a single decoder) as MRTNet (4K points) and the same image encoder (vgg-16). The performance of this variation is 0.05 worse than MRTNet, but it has an order of magnitude less parameters than MRTNet. Another variation is to still use a single decoder but generate a higher-resolution point cloud (16K points). This variation results in improved Chamfer distance, by 0.1, than the first variation, indicating that the increased resolution does improve reconstruction accuracy. Again, even when the number of generated points is higher than 4K, our evaluation is done by randomly selecting 4K points, for fair comparison. The last row in the table is our default setting (32 decoders outputting a total of 32K points). The number of network parameters are reported in Table C.3. Even though the number of points our network generates is 8 times that of MRTNet, its size is only about 1/6 of MRTNet, since our network does not need to represent multiple resolutions at each layer. Compared to AtlasNet, our network is about 1/3 of its size, due to the efficiency of using a fully convolutional architecture. Despite using a much smaller number of parameters, our network outperforms MRTNet (in terms of Chamfer distance metric) by 0.27, and AtlasNet by 0.21.

Method	#parameters
AtlasNet	42.6M
MRTNet	81.6M
Ours (1 dec.)	2.49M
Ours (1 residual dec.)	5.79M
Ours (32 dec.)	14.5M

Table C.3: Comparing the # of network parameters.

Qualitative Results. Figure C.2 shows image-to-shape reconstruction results for images from the test dataset. Overall our method is able to accurately capture fine geometric details such as the chair legs, arms, airplane engines, monitor stands etc. The number of points (32K) is considerably higher than previous work (e.g. 1K by [49] and 4K by [55]). Some specific shapes, such as lamps and jet fighters, present significant challenges for the network as the input images do not contain all the visual details. Nonetheless our method is able to produce a reasonable approximation.

Test on real images. The test set images are synthetically rendered and as such they look similar to the training images. To evaluate our method on real images we use photos downloaded from the Internet, as shown in [55]. They are processed by removing the background so only the foreground object remains. Figure C.3 shows the results. The top row in the figure shows furniture objects, which demonstrate that even though the network is trained using synthetic images rendered with artificial lighting and materials, the model is able to generalize well to real shading, lighting, and materials. The second row shows additional objects where the shading is considerably different from training images. In particular, the last image (desktop computer) is in a category that the training has never seen. Nonetheless the reconstructed shape is reasonable.

Shape correspondence. Once trained, our network learns to generate shapes with corresponding structures. We demonstrate this with the following experiment. First,

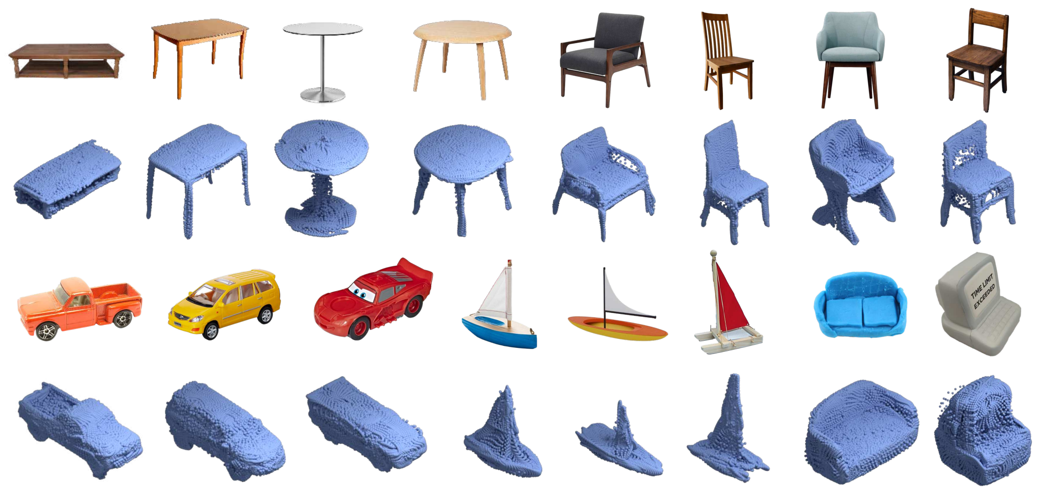


Figure C.3: Image-to-shape reconstruction results on Internet photos. We test our method on real photos downloaded from the Internet and the results are rendered in blue. The test images here are considerably different from the training set. Our method achieves reasonable results with accurate geometric details. The last image (computer) represents a category that has not been seen during training.

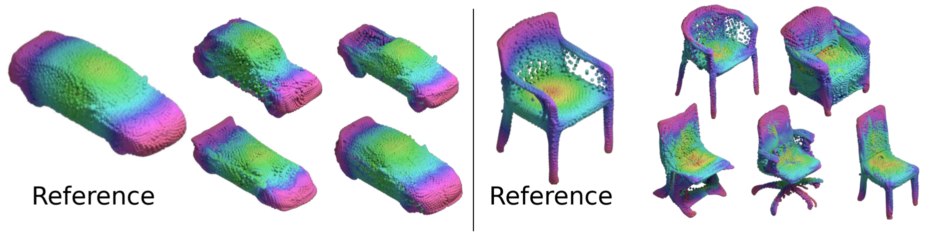


Figure C.4: Visualizing Shape Correspondences. Our network learns approximate shape correspondences even though the training is not supervised with such information. The shapes shown here are generated by 32 decoders.

we randomly select a point cloud generated by our network and call it a reference shape. Then, we assign every point in the reference shape a color, where the hue is computed based on the point's distance to the center of gravity of the object. Then this color assignment is propagated to the other point clouds, such that a point at index (i, j) in the output tensor is assigned the same color as the point on the reference shape at the same index. The resulting colorized point clouds are shown in Figure C.4. Similar color indicates similar index range in the output tensor. Note that even though the network is not trained explicitly with point correspondences as supervisory signal, it learns to generate corresponding parts in the same regions of the output tensor, as can be seen around the tips of the chairs' arms, legs and seats.

C.2 Limiting distribution for the curvature

We start by parameterizing the derivative of a space curve $\dot{x} = \cos(f(t))$ and $\dot{y} = \sin(f(t))$ where f is a neural network. From the standard analysis we know that $f(t)$ converges to a Gaussian with mean μ and kernel $k(\cdot, \cdot)$. Without loss of generality we can assume that the mean μ is such that $\cos(\mu) \neq 0$ and $\sin(\mu) \neq 0$. This can be achieved by adding a fixed bias term μ to the output of the last layer. To compute the limiting distribution of $\ddot{x}$ and $\dot{y}$ we apply the first order delta method to obtain:

$$\dot{x} \to \mathcal{N}(\cos(\mu), \sigma^2 \sin^2(\mu)), \tag{C.1}$$

$$\dot{y} \to \mathcal{N}(\sin(\mu), \sigma^2 \cos^2(\mu)). \tag{C.2}$$

Note we can only apply the first order delta method when the derivatives are not zero. Hence we assumed that μ is set to be a quantity which has this property. Otherwise we need the second-order delta method and the resulting distribution would be χ^2 for one of the derivatives.

Since the derivative is a linear operator it follows that $\ddot{x}$ and $\ddot{y}$ are also GPs. The curvature formula for a arc-length parameterized space curve is $\kappa^2 = \ddot{x}^2 + \ddot{y}^2$. From this it follows that κ^2 is a χ^2 random variable.

Graph parameterization. We also analyze the case where the curve is the graph of a one-dimensional function, i.e., $x = x$,$y = f(x)$. In this case the curvature can be written as $\kappa = \ddot{f}/((1 + \dot{f}^2)^{\frac{3}{2}})$. Once again all the derivatives $\dot{f}$ and $\ddot{f}$ are Gaussian random variables. Assume that $(\dot{f}, \ddot{f})$ are distributed according to $N(0, \Sigma)$.

Here $\Sigma = [\sigma_{\dot{f},\dot{f}}, \sigma_{\dot{f}\ddot{f}}; \sigma_{\dot{f}\ddot{f}}, \sigma_{\ddot{f}\ddot{f}}]$ denoting the joint covariance distribution. Applying the delta method with $g(a,b) = b/(1+a^2)^{3/2}$, we get that k is distributed as a Gaussian random variable $N(0, \nabla g^T \Sigma \nabla g)$. Since $\nabla g(a,b)|_{0,0} = [0,1]$, we have $k \rightarrow N(0, \sigma_{\ddot{f}\ddot{f}})$.

BIBLIOGRAPHY

[1] `https://clippingmagic.com/`.

[2] Achlioptas, Panos, Diamanti, Olga, Mitliagkas, Ioannis, and Guibas, Leonidas J. Learning Representations and Generative Models For 3D Point Clouds. In *International Conference on Machine Learning* (2018).

[3] Alexa, Marc, Behr, Johannes, Cohen-Or, Daniel, Fleishman, Shachar, Levin, David, and Silva, Claudio T. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics 9*, 1 (2003), 3–15.

[4] Allen, Brett, Curless, Brian, and Popović, Zoran. The space of human body shapes: reconstruction and parameterization from range scans. In *ACM transactions on graphics (TOG)* (2003), vol. 22, ACM, pp. 587–594.

[5] Andriluka, Mykhaylo, Roth, Stefan, and Schiele, Bernt. Monocular 3D pose estimation and tracking by detection. In *Computer Vision and Pattern Recognition (CVPR)* (2010), IEEE.

[6] Arthur, David, and Vassilvitskii, Sergei. k-means++: The advantages of careful seeding. Tech. rep., 2007.

[7] Attene, Marco, Falcidieno, Bianca, and Spagnuolo, Michela. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (2006), 181–193.

[8] Au, Oscar Kin-Chung, Zheng, Youyi, Chen, Menglin, Xu, Pengfei, and Tai, Chiew-Lan. Mesh Segmentation with Concavity-Aware Fields. *IEEE Trans. Visual. Comput. Graphics 18*, 7 (Jul 2011), 1125–1134.

[9] Barron, Jonathan T, and Malik, Jitendra. Shape, illumination, and reflectance from shading. *Transactions of Pattern Analysis and Machine Intelligence (PAMI)* (2015).

[10] Barrow, Harry, and Tenenbaum, J. Recovering intrinsic scene characteristics. *Comput. Vis. Syst., A Hanson & E. Riseman (Eds.)* (1978), 3–26.

[11] Bernard, Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.* (Jul 1984).

[12] Besl, Paul J, and McKay, Neil D. Method for registration of 3-d shapes. In *Robotics-DL tentative* (1992), International Society for Optics and Photonics, pp. 586–606.

[13] Blanz, Volker, and Vetter, Thomas. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 187–194.

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

[15] Bowers, John, Wang, Rui, Wei, Li-Yi, and Maletz, David. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph. 29*, 6 (2010), 166:1–166:10.

[16] Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, Eckstein, Jonathan, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning 3*, 1 (2011), 1–122.

[17] Brock, André, Lim, Theodore, Ritchie, James M., and Weston, Nick. Generative and discriminative voxel modeling with convolutional neural networks.

[18] Bronstein, Alexander M, Bronstein, Michael M, Kimmel, Ron, Mahmoudi, Mona, and Sapiro, Guillermo. A gromov-hausdorff framework with diffusion geometry for topologically-robust non-rigid shape matching. *International Journal of Computer Vision 89*, 2 (2010), 266–286.

[19] Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. Spectral networks and locally connected networks on graphs.

[20] Buades, Antoni, Coll, Bartomeu, and Morel, J-M. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition (CVPR)* (2005).

[21] Caron, Mathilde, Bojanowski, Piotr, Joulin, Armand, and Douze, Matthijs. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 132–149.

[22] Caron, Mathilde, Bojanowski, Piotr, Mairal, Julien, and Joulin, Armand. Unsupervised pre-training of image features on non-curated data. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 2959–2968.

[23] Cashman, Thomas J, and Fitzgibbon, Andrew W. What shape are dolphins? building 3d morphable models from 2d images. *IEEE transactions on pattern analysis and machine intelligence 35*, 1 (2013), 232–244.

[24] Chang, Angel X, Funkhouser, Thomas, Guibas, Leonidas, Hanrahan, Pat, Huang, Qixing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015).

[25] Chang, JH Rick, Li, Chun-Liang, Poczos, Barnabas, Kumar, BVK Vijaya, and Sankaranarayanan, Aswin C. One network to solve them all—solving linear inverse problems using deep projection models. *arXiv preprint* (2017).

[26] Chazelle, Bernard, and Dobkin, David Paul. *Optimal Convex Decompositions.* No. C in Machine Intelligence and Pattern Recognition. 1985, pp. 63–133.

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

[28] Chen, Liang-Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915* (2016).

[29] Chen, Yang, and Medioni, Gérard. Object modelling by registration of multiple range images. *Image and vision computing 10*, 3 (1992), 145–155.

[30] Chen, Zhiqin, Yin, Kangxue, Fisher, Matthew, Chaudhuri, Siddhartha, and Zhang, Hao. Bae-net: branched autoencoder for shape co-segmentation. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 8490–8499.

[31] Chen, Zhiqin, and Zhang, Hao. Learning implicit fields for generative shape modeling. In *The IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[32] Cheng, Zezhou, Gadelha, Matheus, Maji, Subhransu, and Sheldon, Daniel. A Bayesian Perspective on the Deep Image Prior. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[33] Cho, Youngmin, and Saul, Lawrence K. Kernel methods for deep learning. In *Advances in neural information processing systems* (2009), pp. 342–350.

[34] Chopra, Sumit, Hadsell, Raia, and LeCun, Yann. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005), vol. 1, IEEE, pp. 539–546.

[35] Choy, Christopher B, Xu, Danfei, Gwak, JunYoung, Chen, Kevin, and Savarese, Silvio. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *European Conference on Computer Vision* (2016).

[36] Cocosco, Chris A, Kollokian, Vasken, Kwan, Remi K-S, Pike, G Bruce, and Evans, Alan C. Brainweb: Online interface to a 3D MRI simulated brain database. In *NeuroImage* (1997).

[37] Cohen-Steiner, David, Alliez, Pierre, and Desbrun, Mathieu. Variational shape approximation. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM.

[38] Cover, Thomas M, and Thomas, Joy A. *Elements of information theory.* John Wiley & Sons, 2012.

[39] Dabov, Kostadin, Foi, Alessandro, Katkovnik, Vladimir, and Egiazarian, Karen. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on image processing 16*, 8 (2007), 2080–2095.

[40] Dasgupta, Sanjoy, and Freund, Yoav. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing* (2008), ACM, pp. 537–546.

[41] Dinh, Laurent, Krueger, David, and Bengio, Yoshua. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* (2014).

[42] Doersch, Carl, Gupta, Abhinav, and Efros, Alexei A. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1422–1430.

[43] Donahue, Jeff, and Simonyan, Karen. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems* (2019), pp. 10541–10551.

[44] Dong, Chao, Loy, Chen Change, He, Kaiming, and Tang, Xiaoou. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision (ECCV)* (2014).

[45] Dosovitskiy, Alexey, Tobias Springenberg, Jost, and Brox, Thomas. Learning to generate chairs with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[46] Dutilleux, Pierre. An implementation of the "algorithme à trous" to compute the wavelet transform. In *Wavelets*. Springer, 1990, pp. 298–304.

[47] Eigen, David, and Fergus, Rob. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *International Conference on Computer Vision (ICCV)* (2015).

[48] Eigen, David, Puhrsch, Christian, and Fergus, Rob. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems (NIPS)* (2014).

[49] Fan, Haoqiang, Su, Hao, and Guibas, Leonidas J. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *Computer Vision and Pattern Recognition (CVPR)* (2017).

[50] Gadelha, Matheus, Gori, Giorgio, Ceylan, Duygu, Mech, Radomir, Carr, Nathan, Boubekeur, Tamy, Wang, Rui, and Maji, Subhransu. Learning generative models of shape handles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

[51] Gadelha, Matheus, Maji, Subhransu, and Wang, Rui. 3d shape generation using spatially ordered point clouds. In *British Machine Vision Conference (BMVC)* (2017).

[52] Gadelha, Matheus, Maji, Subhransu, and Wang, Rui. Unsupervised 3D Shape Induction from 2D Views of Multiple Objects. In *International Conference on 3D Vision (3DV)* (2017).

[53] Gadelha, Matheus, Rai, Aartika, Maji, Subhransu, and Wang, Rui. Inferring 3d shapes from image collections using adversarial networks. *To appear at International Journal of Computer Vision (IJCV)* (2020).

[54] Gadelha, Matheus, RoyChowdhury, Aruni, Sharma, Gopal, Kalogerakis, Evangelos, Cao, Liangliang, Learned-Miller, Erik, Wang, Rui, and Maji, Subhransu. Label-efficient learning on point clouds using approximate convex decompositions. In *European Conference on Computer Vision (ECCV)* (2020).

[55] Gadelha, Matheus, Wang, Rui, and Maji, Subhransu. Multiresolution Tree Networks for 3D Point Cloud Processing. In *ECCV* (2018).

[56] Gadelha, Matheus, Wang, Rui, and Maji, Subhransu. Shape reconstruction using differentiable projections and deep priors. In *International Conference on Computer Vision (ICCV)* (2019).

[57] Gal, Ran, Sorkine, Olga, Mitra, Niloy J., and Cohen-Or, Daniel. iwires: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (Siggraph) 28*, 3 (2009), #33, 1–10.

[58] Genova, Kyle, Cole, Forrester, Vlasic, Daniel, Sarna, Aaron, Freeman, William T, and Funkhouser, Thomas. Learning shape templates with structured implicit functions. In *International Conference on Computer Vision* (2019).

[59] Ghosh, Mukulika, Amato, Nancy M., Lu, Yanyan, and Lien, Jyh-Ming. Fast approximate convex decomposition using relative concavity. *Compututer Aided Deisgn. 45* (Feb 2013), 494–504.

[60] Gidaris, Spyros, Singh, Praveer, and Komodakis, Nikos. Unsupervised representation learning by predicting image rotations. In *ICLR* (2018).

[61] Gionis, Aristides, Indyk, Piotr, Motwani, Rajeev, et al. Similarity search in high dimensions via hashing. In *VLDB* (1999), vol. 99, pp. 518–529.

[62] Girdhar, R., Fouhey, D.F., Rodriguez, M., and Gupta, A. Learning a predictable and generative vector representation for objects. In *ECCV* (2016).

[63] Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)* (2014).

[64] Gori, Giorgio, Sheffer, Alla, Vining, Nicholas, Rosales, Enrique, Carr, Nathan, and Ju, Tao. Flowrep: Descriptive curve networks for free-form design shapes. *ACM Transaction on Graphics 36*, 4 (2017).

[65] Goyal, Priya, Mahajan, Dhruv, Gupta, Abhinav, and Misra, Ishan. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235* (2019).

[66] Graham, Benjamin, and van der Maaten, Laurens. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307* (2017).

[67] Gretton, Arthur, Borgwardt, Karsten M, Rasch, Malte, Schölkopf, Bernhard, and Smola, Alex J. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems (NIPS)* (2006).

[68] Groueix, Thibault, Fisher, Matthew, Kim, Vladimir G., Russell, Bryan, and Aubry, Mathieu. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2018).

[69] Hadsell, Raia, Chopra, Sumit, and LeCun, Yann. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (2006), vol. 2, IEEE, pp. 1735–1742.

[70] Häne, Christian, Tulsiani, Shubham, and Malik, Jitendra. Hierarchical surface prediction for 3d object reconstruction. In *International Conference on 3D Vision (3DV)* (2017).

[71] Hartley, Richard, and Zisserman, Andrew. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[72] Hassani, Kaveh, and Haley, Mike. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 8160–8171.

[73] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision* (2014), Springer, pp. 346–361.

[74] Henderson, Paul, and Ferrari, Vittorio. Learning to Generate and Reconstruct 3D Meshes with only 2D Supervision. In *British Machine Vision Conference (BMVC)* (2018).

[75] Hoffman, Donald D, and Richards, Whitman. Parts of recognition.

[76] Hoiem, Derek, Efros, Alexei A, and Hebert, Martial. Geometric context from a single image. In *International Conference on Computer Vision (ICCV)* (2005).

[77] Huang, Haibin, Kalogerakis, Evangelos, Chaudhuri, Siddhartha, Ceylan, Duygu, Kim, Vladimir G., and Yumer, Ersin. Learning local shape descriptors from part correspondences with multiview convolutional networks. *ACM Transactions on Graphics 37*, 1 (2018).

[78] Huang, Jian, Yagel, Roni, Filippov, Vassily, and Kurzion, Yair. An accurate method for voxelizing polygon meshes. *IEEE Symposium on Volume Visualization* (Oct 1998).

[79] Huang, Jing, and You, Suya. Point cloud labeling using 3d convolutional neural network. In *ICPR* (2016), pp. 2670–2675.

[80] Huang, Qi-Xing, and Guibas, Leonidas. Consistent shape maps via semidefinite programming. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 177–186.

[81] Ioffe, Sergey, and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (2015), ICML'15.

[82] Jaderberg, Max, Simonyan, Karen, Zisserman, Andrew, and kavukcuoglu, koray. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems (NIPS)* (2015).

[83] Ji, Zhongping, Liu, Ligang, and Wang, Yigang. B-mesh: A modeling system for base meshes of 3d articulated shapes. *Computer Graphics Forum 29*, 7 (2010), 2169–2177.

[84] Jiang, Huaizu, Larsson, Gustav, Maire Greg Shakhnarovich, Michael, and Learned-Miller, Erik. Self-supervised relative depth learning for urban scene understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–35.

[85] Kaick, Oliver Van, Fish, Noa, Kleiman, Yanir, Asafi, Shmuel, and Cohen-OR, Daniel. Shape segmentation by approximate convexity analysis. *ACM Trans. Graph. 34*, 1 (2014).

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

[87] Kanazawa, Angjoo, Tulsiani, Shubham, Efros, Alexei A., and Malik, Jitendra. Learning category-specific mesh reconstruction from image collections. In *ECCV* (2018).

[88] Kar, Abhishek, Tulsiani, Shubham, Carreira, Jo?o, and Malik, Jitendra. Category-specific object reconstruction from a single image. In *Computer Vision and Pattern Recognition (CVPR)* (2015).

[89] Kato, Hiroharu, Ushiku, Yoshitaka, and Harada, Tatsuya. Neural 3d mesh renderer. In *Computer Vision and Pattern Recognition (CVPR)* (2018).

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

[91] Kazhdan, Michael, and Hoppe, Hugues. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG) 32*, 3 (2013), 29.

[92] Ke, Tsung-Wei, Maire, Michael, and Yu, Stella X. Multigrid neural architectures. In *CVPR* (2017).

[93] Kingma, Diederik P, and Ba, Jimmy. ADAM: a method for stochastic optimization. In *International Conference on Learning Representations* (2014).

[94] Kingma, Diederik P., and Welling, Max. Auto-encoding variational bayes. *CoRR abs/1312.6114* (2013).

[95] Kingma, Durk P, and Dhariwal, Prafulla. GLOW: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems* (2018), pp. 10236–10245.

[96] Klokov, Roman, and Lempitsky, Victor. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV* (2017).

[97] Klokov, Roman, and Lempitsky, Victor. Escape from cells: Deep Kd-Networks for the recognition of 3D point cloud models. In *Proc. ICCV* (2017).

[98] Kolesnikov, Alexander, Zhai, Xiaohua, and Beyer, Lucas. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005* (2019).

[99] Kulkarni, Tejas D, Whitney, William F, Kohli, Pushmeet, and Tenenbaum, Josh. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems (NIPS)* (2015).

[100] Land, Edwin H, and McCann, John J. Lightness and retinex theory. *JOSA 61*, 1 (1971), 1–11.

[101] Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Learning representations for automatic colorization. In *European Conference on Computer Vision* (2016), Springer, pp. 577–593.

[102] Laurentini, Aldo. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on pattern analysis and machine intelligence 16*, 2 (1994), 150–162.

[103] Li, Jiaxin, Chen, Ben M, and Hee Lee, Gim. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 9397–9406.

[104] Li, Jun, Xu, Kai, Chaudhuri, Siddhartha, Yumer, Ersin, Zhang, Hao, and Guibas, Leonidas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017) 36*, 4 (2017), to appear.

[105] Li, Tzu-Mao, Aittala, Miika, Durand, Frédo, and Lehtinen, Jaakko. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graph (SIGGRAPH Asia)* (2018).

[106] Li, Yangyan, Pirk, Soeren, Su, Hao, Qi, Charles R., and Guibas, Leonidas J. Fpnn: Field probing neural networks for 3d data. In *NIPS* (2016).

[107] Lien, Jyh-Ming, and Amato, Nancy M. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling* (2007), SPM '07.

[108] Lin, Chen-Hsuan, Kong, Chen, and Lucey, Simon. Learning efficient point cloud generation for dense 3d object reconstruction. In *AAAI Conference on Artificial Intelligence (AAAI)* (2018).

[109] Lin, Tsung-Yi, Dollár, Piotr, Girshick, Ross, He, Kaiming, Hariharan, Bharath, and Belongie, Serge. Feature pyramid networks for object detection. In *CVPR* (2017).

[110] Liu, Hsueh-Ti Derek, Tao, Michael, and Jacobson, Alec. Paparazzi: Surface editing by way of multi-view image processing. *ACM Transactions on Graphcs.* (2018).

[111] Lun, Zhaoliang, Gadelha, Matheus, Kalogerakis, Evangelos, Maji, Subhransu, and Wang, Rui. 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks. *International Conference on 3D Vision (3DV)* (2017), 67–77.

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

[113] Maggioni, Matteo, Katkovnik, Vladimir, Egiazarian, Karen, and Foi, Alessandro. Nonlocal transform-domain filter for volumetric data denoising and reconstruction. *IEEE transactions on image processing 22*, 1 (2013), 119–133.

[114] Mamou, Khaled. Volumetric approximate convex decomposition. In *Game Engine Gems 3*, Eric Lengyel, Ed. A K Peters / CRC Press, 2016, ch. 12, pp. 141–158.

[115] Masci, Jonathan, Boscaini, Davide, Bronstein, Michael M., and Vandergheynst, Pierre. Geodesic convolutional neural networks on riemannian manifolds.

[116] Maturana, Daniel, and Scherer, Sebastian. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS* (2015), pp. 922–928.

[117] Meagher, Donald. Geometric modeling using octree encoding. *Computer graphics and image processing 19*, 2 (1982), 129–147.

[118] Mehra, Ravish, Zhou, Qingnan, Long, Jeremy, Sheffer, Alla, Gooch, Amy, and Mitra, Niloy J. Abstraction of man-made shapes. In *ACM SIGGRAPH Asia 2009 Papers* (2009), SIGGRAPH Asia '09, ACM.

[119] Mescheder, Lars, Oechsle, Michael, Niemeyer, Michael, Nowozin, Sebastian, and Geiger, Andreas. Occupancy networks: Learning 3D reconstruction in function space. In *The IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[120] Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K. Automatic grasp planning using shape primitives. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)* (2003), vol. 2, pp. 1824–1829 vol.2.

[121] Mo, Kaichun, Guerrero, Paul, Yi, Li, Su, Hao, Wonka, Peter, Mitra, Niloy, and Guibas, Leonidas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Transactions on Graphics (TOG), Siggraph Asia 2019 38*, 6 (2019), Article 242.

[122] Mo, Kaichun, Zhu, Shilin, Chang, Angel X., Yi, Li, Tripathi, Subarna, Guibas, Leonidas J., and Su, Hao. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[123] Müllner, Daniel, et al. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software 53*, 9 (2013), 1–18.

[124] Muralikrishnan, Sanjeev, Kim, Vladimir G., and Chaudhuri, Siddhartha. Tags2Parts: Discovering semantic regions from shape tags. In *Proc. CVPR* (2018), IEEE.

[125] Nalbach, Oliver, Arabadzhiyska, Elena, Mehta, Dushyant, Seidel, Hans-Peter, and Ritschel, Tobias. Deep shading: Convolutional neural networks for screen-space shading. *arXiv preprint arXiv:1603.06078* (2016).

[126] Neal, Radford M. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

[127] Nehari, Zeev. *Conformal mapping*. Courier Corporation, 2012.

[128] Nguyen-Phuoc, Thu, Li, Chuan, Balaban, Stephen, and Yang, Yong-Liang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *Advances in Neural Information Processing Systems 31* (2018).

[129] Nguyen-Phuoc, Thu, Li, Chuan, Theis, Lucas, Richardt, Christian, and Yang, Yong-Liang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *International Conference on Computer Vision (ICCV)* (2019).

[130] Niu, Chengjie, Li, Jun, and Xu, Kai. Im2struct: Recovering 3d shape structure from a single rgb image. In *Computer Vision and Pattern Regognition (CVPR)* (2018).

[131] Noroozi, Mehdi, and Favaro, Paolo. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision* (2016), Springer, pp. 69–84.

[132] Noroozi, Mehdi, Vinjimoor, Ananth, Favaro, Paolo, and Pirsiavash, Hamed. Boosting self-supervised learning via knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 9359–9367.

[133] Odena, Augustus, Dumoulin, Vincent, and Olah, Chris. Deconvolution and checkerboard artifacts. *Distill* (2016).

[134] Öztireli, A Cengiz, Guennebaud, Gael, and Gross, Markus. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 493–501.

[135] Park, Jeong Joon, Florence, Peter, Straub, Julian, Newcombe, Richard, and Lovegrove, Steven. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[136] Paschalidou, Despoina, Ulusoy, Ali Osman, and Geiger, Andreas. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[137] Pathak, Deepak, Girshick, Ross, Dollár, Piotr, Darrell, Trevor, and Hariharan, Bharath. Learning features by watching objects move. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 2701–2710.

[138] Pathak, Deepak, Krahenbuhl, Philipp, Donahue, Jeff, Darrell, Trevor, and Efros, Alexei A. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2536–2544.

[139] Prasad, Mukta, Fitzgibbon, Andrew, Zisserman, Andrew, and Van Gool, Luc. Finding nemo: Deformable object class modelling using curve matching. In *Computer Vision and Pattern Recognition (CVPR)* (2010).

[140] 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).

[141] Qi, Charles R., Yi, Li, Su, Hao, and Guibas, Leonidas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NIPS* (2017).

[142] Qi, Charles Ruizhongtai, Su, Hao, Nießner, Matthias, Dai, Angela, Yan, Mengyuan, and Guibas, Leonidas. Volumetric and multi-view cnns for object classification on 3d data. In *Computer Vision and Pattern Recognition (CVPR)* (2016).

[143] Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[144] Radon, J. On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging 5*, 4 (Dec 1986), 170–176.

[145] Rezende, Danilo Jimenez, Eslami, SM, Mohamed, Shakir, Battaglia, Peter, Jaderberg, Max, and Heess, Nicolas. Unsupervised learning of 3D structure from images. In *Advances in Neural Information Processing Systems (NIPS)* (2016).

[146] Richter, Stephan R., and Roth, Stefan. Matryoshka Networks: Predicting 3D Geometry via Nested Shape Layers. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2018).

[147] Riegler, Gernot, Ulusoys, Ali Osman, and Geiger, Andreas. Octnet: Learning deep 3D representations at high resolutions. In *Proc. CVPR* (2017).

[148] Ronneberger, O., P.Fischer, and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (2015), LNCS, pp. 234–241.

[149] Roweis, Sam T, and Saul, Lawrence K. Nonlinear dimensionality reduction by locally linear embedding. *science 290*, 5500 (2000), 2323–2326.

[150] Rudin, Leonid I, Osher, Stanley, and Fatemi, Emad. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena 60*, 1-4 (1992), 259–268.

[151] Salimans, Tim, Goodfellow, Ian J., Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. *CoRR abs/1606.03498* (2016).

[152] Sánchez, Jorge, Perronnin, Florent, Mensink, Thomas, and Verbeek, Jakob. Image classification with the fisher vector: Theory and practice. *International journal of computer vision 105*, 3 (2013), 222–245.

[153] Savarese, Silvio, and Fei-Fei, Li. 3d generic object categorization, localization and pose estimation. In *International Conference on Computer Vision (ICCV)* (2007).

[154] Saxe, Andrew M, Koh, Pang Wei, Chen, Zhenghao, Bhand, Maneesh, Suresh, Bipin, and Ng, Andrew Y. On random weights and unsupervised feature learning. In *ICML* (2011), vol. 2, p. 6.

[155] Saxena, Ashutosh, Chung, Sung H, and Ng, A. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems (NIPS)* (2005).

[156] Schwing, Alexander G, and Urtasun, Raquel. Efficient exact inference for 3d indoor scene understanding. In *European Conference on Computer Vision (ECCV)* (2012).

[157] Sharma, Abhishek, Grau, Oliver, and Fritz, Mario. Vconv-dae: Deep volumetric shape learning without object labels. In *European Conference on Computer Vision* (2016), Springer, pp. 236–250.

[158] Sharma, Gopal, Kalogerakis, Evangelos, and Maji, Subhransu. Learning point embeddings from shape repositories for few-shot segmentation. *CoRR abs/1910.01269* (2019).

[159] Shepard, Donald. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference* (1968), ACM, pp. 517–524.

[160] Shepp, Lawrence A, and Logan, Benjamin F. The Fourier reconstruction of a head section. *IEEE Transactions on nuclear science 21*, 3 (1974), 21–43.

[161] Shi, Jianbo, and Malik, Jitendra. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence 22*, 8 (2000), 888–905.

[162] Simonovsky, Martin, and Komodakis, Nikos. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR* (2017).

[163] Simonyan, Karen, and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition.

[164] Sitzmann, Vincent, Thies, Justus, Heide, Felix, Nießner, Matthias, Wetzstein, Gordon, and Zollhöfer, Michael. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[165] Solak, Ercan, Murray-Smith, Roderick, Leithead, William E, Leith, Douglas J, and Rasmussen, Carl E. Derivative observations in gaussian process models of dynamic systems. In *Advances in neural information processing systems* (2003).

[166] Soltani, Amir Arsalan, Huang, Haibin, Wu, Jiajun, Kulkarni, Tejas, and Tenenbaum, Joshua. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *CVPR* (2017).

[167] Su, Hang, Jampani, Varun, Su, Deqing, Maji, Subhransu, Kalogerakis, Evangelos, Yang, Ming-Hsuan, and Kautz, Jan. SPLATNet: Sparse lattice networks for point cloud processing. *arXiv preprint arXiv:1802.08275* (2018).

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

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

[170] Su, Hao, Qi, Charles R, Li, Yangyan, and Guibas, Leonidas J. Render for CNN: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *International Conference on Computer Vision (ICCV)* (2015).

[171] Su, Jong-Chyi, Maji, Subhransu, and Hariharan, Bharath. When does self-supervision improve few-shot learning? *arXiv preprint arXiv:1910.03560* (2019).

[172] Sussmann, Héctor J. Orbits of families of vector fields and integrability of distributions. *Transactions of the American Mathematical Society 180* (1973), 171–188.

[173] Tatarchenko, Maxim, Dosovitskiy, Alexey, and Brox, Thomas. Multi-view 3D models from single images with a convolutional network. In *European Conference on Computer Vision (ECCV)* (2016).

[174] Tatarchenko, Maxim, Dosovitskiy, Alexey, and Brox, Thomas. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *arXiv preprint arXiv:1703.09438* (2017).

[175] Tatarchenko, Maxim, Park, Jaesik, Koltun, Vladlen, and Zhou., Qian-Yi. Tangent convolutions for dense prediction in 3D. *CVPR* (2018).

[176] Tenenbaum, Joshua B, De Silva, Vin, and Langford, John C. A global geometric framework for nonlinear dimensionality reduction. *science 290*, 5500 (2000), 2319–2323.

[177] Thabet, Ali, Alwassel, Humam, and Ghanem, Bernard. MortonNet: Self-Supervised Learning of Local Features in 3D Point Clouds. *arXiv* (Mar 2019).

[178] Thiery, Jean-Marc, Guy, Emilie, and Boubekeur, Tamy. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia 2013) 32*, 6 (2013), Art. No. 178.

[179] Thiery, Jean-Marc, Guy, Émilie, Boubekeur, Tamy, and Eisemann, Elmar. Animated mesh approximation with sphere-meshes. *ACM Trans. Graph.* (2016), 30:1–30:13.

[180] Tipping, Michael E, and Bishop, Christopher M. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61*, 3 (1999), 611–622.

[181] Tkach, Anastasia, Pauly, Mark, and Tagliasacchi, Andrea. Sphere-meshes for real-time hand modeling and tracking. *ACM Trans. Graph. 35*, 6 (2016).

[182] Tomasi, Carlo, and Manduchi, Roberto. Bilateral filtering for gray and color images. In *International Conference on Computer Vision (ICCV)* (1998).

[183] Trinh, Trieu H, Luong, Minh-Thang, and Le, Quoc V. Selfie: Self-supervised pretraining for image embedding. *arXiv preprint arXiv:1906.02940* (2019).

[184] Tulsiani, Shubham, Carreira, Joao, and Malik, Jitendra. Pose induction for novel object categories. In *International Conference on Computer Vision (ICCV)* (2015).

[185] Tulsiani, Shubham, Efros, Alexei A., and Malik, Jitendra. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Computer Vision and Pattern Regognition (CVPR)* (2018).

[186] Tulsiani, Shubham, Su, Hao, Guibas, Leonidas J., Efros, Alexei A., and Malik, Jitendra. Learning shape abstractions by assembling volumetric primitives. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[187] Tulsiani, Shubham, Zhou, Tinghui, Efros, Alexei A., and Malik, Jitendra. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Computer Vision and Pattern Regognition (CVPR)* (2017).

[188] Ulyanov, Dmitry, Vedaldi, Andrea, and Lempitsky, Victor. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018).

[189] Van Kaick, Oliver, Zhang, Hao, Hamarneh, Ghassan, and Cohen-Or, Daniel. A survey on shape correspondence. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 1681–1707.

[190] Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research 11*, Dec (2010), 3371–3408.

[191] Vinh, Nguyen Xuan, Epps, Julien, and Bailey, James. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research 11*, Oct (2010), 2837–2854.

[192] Von Luxburg, Ulrike. A tutorial on spectral clustering. *Statistics and computing 17*, 4 (2007), 395–416.

[193] Wang, Nanyang, Zhang, Yinda, Li, Zhuwen, Fu, Yanwei, Liu, Wei, and Jiang, Yu-Gang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV* (2018).

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

[195] Wang, Peng-Shuai, Sun, Chun-Yu, Liu, Yang, and Tong, Xin. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Trans. Graph. 37*, 6 (2018).

[196] Wang, Xiaolong, and Gupta, Abhinav. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 2794–2802.

[197] Wang, Xiaolong, He, Kaiming, and Gupta, Abhinav. Transitive invariance for self-supervised visual representation learning. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 1329–1338.

[198] Wang, Yue, Sun, Yongbin, Liu, Ziwei, Sarma, Sanjay E, Bronstein, Michael M, and Solomon, Justin M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG) 38*, 5 (2019), 1–12.

[199] Weller, René. A Brief Overview of Collision Detection. *SpringerLink* (2013), 9–46.

[200] Welling, Max, and Teh, Yee W. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning (ICML)* (2011).

[201] Williams, Christopher KI. Computing with infinite networks. In *Advances in neural information processing systems* (1997), pp. 295–301.

[202] Williams, Francis, Schneider, Teseo, Silva, Claudio, Zorin, Denis, Bruna, Joan, and Panozzo, Daniele. Deep geometric prior for surface reconstruction. In *The IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[203] Woodham, Robert J. Photometric method for determining surface orientation from multiple images. *Optical engineering 19*, 1 (1980), 191139–191139.

[204] 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.

[205] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition* (2015).

[206] Wu, Zhirong, Song, Shuran, Khosla, Aditya, Yu, Fisher, Zhang, Linguang, Tang, Xiaoou, and Xiao, Jianxiong. 3d shapenets: A deep representation for volumetric shapes. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[207] Xian, Chuhua, Lin, Hongwei, and Gao, Shuming. Automatic cage generation by improved obbs for mesh deformation. *The Visual Computer 28*, 1 (2012), 21–33.

[208] Xie, Junyuan, Xu, Linli, and Chen, Enhong. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems (NIPS)* (2012).

[209] Xie, Qizhe, Hovy, Eduard, Luong, Minh-Thang, and Le, Quoc V. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252* (2019).

[210] Xu, Li, Lu, Cewu, Xu, Yi, and Jia, Jiaya. Image smoothing via L0 gradient minimization. *ACM Transactions on Graphics (TOG) 30*, 6 (2011), 174.

[211] Yan, Xinchen, Yang, Jimei, Yumer, Ersin, Guo, Yijie, and Lee, Honglak. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems* (2016).

[212] Yang, Guandao, Huang, Xun, Hao, Zekun, Liu, Ming-Yu, Belongie, Serge, and Hariharan, Bharath. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 4541–4550.

[213] Yang, Yan, Sun, Jian, Li, Huibin, and Xu, Zongben. Deep ADMM-Net for Compressive Sensing MRI. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[214] Yang, Yaoqing, Feng, Chen, Shen, Yiru, and Tian, Dong. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 206–215.

[215] Yi, Li, Guibas, Leonidas, Hertzmann, Aaron, Kim, Vladimir G., Su, Hao, and Yumer, Ersin. Learning hierarchical shape segmentation and labeling from online repositories. *ACM Trans. Graph. 36* (July 2017).

[216] 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).

[217] Yi, Li, Su, Hao, Guo, Xingwen, and Guibas, Leonidas. SyncSpecCNN: synchronized spectral cnn for 3d shape segmentation. In *CVPR* (2017).

[218] Yu, Fisher, and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[219] Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Colorful image colorization. In *European conference on computer vision* (2016), Springer, pp. 649–666.

[220] Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1058–1067.

[221] Zhao, Yongheng, Birdal, Tolga, Deng, Haowen, and Tombari, Federico. 3d point capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 1009–1018.

[222] Zhou, Tinghui, Tulsiani, Shubham, Sun, Weilun, Malik, Jitendra, and Efros, Alexei A. View synthesis by appearance flow. In *European Conference on Computer Vision (ECCV)* (2016).

[223] Zhou, Yang, Yin, Kangxue, Huang, Hui, Zhang, Hao, Gong, Minglun, and Cohen-Or, Daniel. Generalized cylinder decomposition. *ACM Trans. Graph. 34*, 6 (2015).

[224] Zhou, Yi, Barnes, Connelly, Jingwan, Lu, Jimei, Yang, and Hao, Li. On the continuity of rotation representations in neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[225] Zhou Ren, Junsong Yuan, Chunyuan Li, and Wenyu Liu. Minimum near-convex decomposition for robust shape representation. In *2011 International Conference on Computer Vision* (Nov 2011).

[226] Zhu, Chenyang, Xu, Kai, Chaudhuri, Siddhartha, Yi, Li, Guibas, Leonidas J., and Zhang, Hao. Cosegnet: Deep co-segmentation of 3d shapes with group consistency loss. *CoRR abs/1903.10297* (2019).

[227] Zoran, Daniel, and Weiss, Yair. From learning models of natural image patches to whole image restoration. In *International Conference on Computer Vision (ICCV)* (2011).