UNIVERSITY OF CALIFORNIA,
IRVINE

Geometric Algorithms for Deep Point Networks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Nitin Agarwal

Dissertation Committee:
Professor Gopi Meenakshisundaram, Chair
Professor Charless Fowlkes
Assistant Professor Shuang Zhao

2020

DEDICATION

To my grandparents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I first wish to thank my PhD advisor, Professor Gopi Meenakshisundaram for being a wonderful mentor over the past six years. I am deeply indebted to him for always making the time to discuss my work and for his insightful comments on my research. He has always been understanding and supportive, even when I decided to switch my research interest from medical imaging to deep learning. In addition to my research, he has constantly helped me navigate my way though graduate school, internships, and life in general.

I thank my committee members, Professors Charless Fowlkes and Shuang Zhao for supporting me along the way and for their comments on ways to improve my work. I would like to also acknowledge the support of my collaborators: Professors Niloy Mitra, Sung-Eui Yoon and Xiangmin Xu with whom I have had the pleasure to work with in various capacities during the last six years. I am also grateful for all the support given by Professor Aditi Majumder, especially during the initial years of my PhD. I also acknowledge the other faculty (Dr. Raymond Klefstad and Dr. Mustafa Ibrahim for whom I TAed) and the staff (Holly Byrnes, Mary Carrillo and Sholeh Satari) in the ICS Department for their assistance throughout my time here.

I shared a special bond with my colleagues in the iGravi and the Vision lab at UCI, especially Jia, Mahdi, Hao, Mehdi, Yu, Zahra, Ali, Cheng, Samia, Zhanhang, Jessica, Isabela, Andy and Shu and I am grateful to them for numerous things from sharing their GPUs to keeping me company in the lab when I was working till late. Coming from Seattle, I never thought I could like any other US city as much, but my friends - Dhrub, Ani, Primal, Roberto, Sabur, Biswadeep, Rohit, Gaurav with whom I went hiking, played tennis, and discovered Irvine, made it possible. My interactions with them helped me survive the graduate program.

No part of this journey would have been possible without the love of my family. I would like to thank my parents and my brother for their belief in me and for their encouragement all along the way. I am especially grateful to my wife Bijetri, who tolerated my late nights, crazy deadlines, incessant grumblings for six years. It has been and will always be a comfort to have her by my side.

CURRICULUM VITAE

Nitin Agarwal

EDUCATION

Doctor of Philosophy in Computer Science **2014-2020**
University of California, Irvine *Irvine, California*

Masters of Science in Bioengineering **2011-2013**
University of Washington, Seattle *Seattle, Washington*

Bachelor in Engineering in Electronics & Instrumentation **2010**
Birla Institute of Technology and Science, Pilani *Rajasthan, India*

SELECT PUBLICATIONS

1. **Nitin Agarwal**, M. Gopi."GAM: Guided and Augmented Meshing for Deep Point Networks." In International Conference on 3D Vision (3DV), 2020. (under review)

2. **Nitin Agarwal**, Sung-Eui Yoon, M. Gopi. "Learning Embedding of 3D models with Quadric Loss." In British Machine Vision Conference (BMVC), 2019.

3. **Nitin Agarwal**, Xiangmin Xu, M. Gopi. "Geometry Processing of Conventionally Produced Mouse Brain Slice Images." In Journal of Neuroscience Methods, 2018.

4. **Nitin Agarwal**, Xiangmin Xu, M. Gopi. "Automatic Detection of Histological Artifacts in Mouse Brain Slice Images". In International Conference on Medical Image Computing and Computer Assisted Intervention, Workshop on Medical Computer Vision: Algorithms for Big Data (MICCAI), 2016.

5. **Nitin Agarwal**, Xiangmin Xu, Gopi Meenakshisundaram. "Robust Registration of Mouse Brain Slices with Severe Histological Artifacts." In Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP), 2016.

DATASET

1. **Nitin Agarwal**, Andrew Self, M.Gopi. "AntHQ: A High Quality 3D Mesh Dataset with Artistic Embellishments."

2. **Nitin Agarwal**, Lujia Chen, Sina Sobhani, Xiangmin Xu, M.Gopi. "Annotated 3D Mouse Brain Model".

ABSTRACT OF THE DISSERTATION

Geometric Algorithms for Deep Point Networks

By

Nitin Agarwal

Doctor of Philosophy in Computer Science

University of California, Irvine, 2020

Professor Gopi Meenakshisundaram, Chair

Point networks have recently enjoyed a lot of success due to the significant growth in 3D data and the development of novel point network architectures focusing on new applications. In this dissertation, I show that the performance of existing point networks can be improved by using insights from classical geometry processing algorithms. I demonstrate this first by proposing a new point-surface loss function called Quadric loss, which preserves sharp features such as edges and corners of 3D shapes. Inspired by the classical quadric simplification, Quadric loss minimizes the quadric error between the reconstructed points and the input surface. I show that combining Quadric loss with other popular point based loss functions can achieve better reconstruction results than existing approaches. Next, I propose a new meshing algorithm called Guided and Augmented Meshing, GAM, which generates a surface for the output points of a point network using a mesh prior. GAM decouples the geometry from the topology by making the point network solely responsible for geometry and the mesh prior responsible for topology. I show the benefits of such a disentanglement for single-view shape prediction and fair evaluation of deep point networks. Finally, I also present a novel 3D mesh dataset which was curated during this research, along with its several promising future applications.

Chapter 1

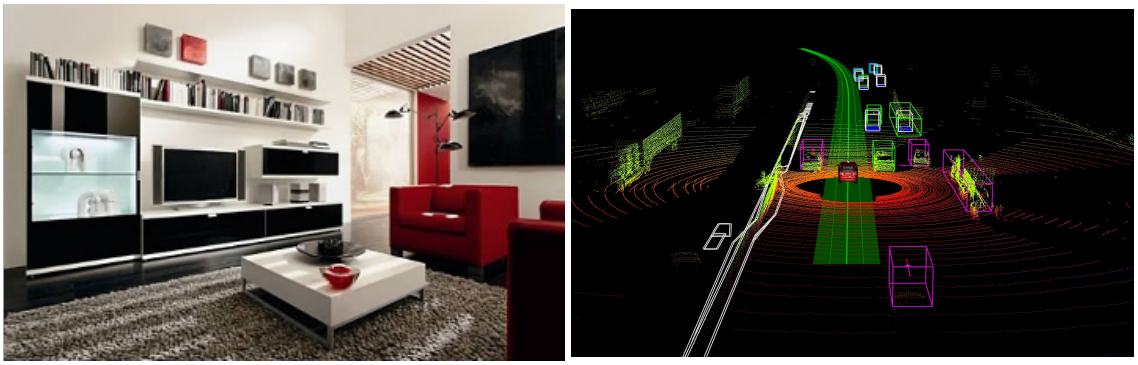
Introduction

The fundamental objective of this dissertation is to demonstrate that the performance of deep point networks can be improved by borrowing insights from classical geometry processing algorithms. In this chapter, we provide an overview of the dissertation, starting with a motivation of the broader area of 3D deep learning and categorization of the task space in 3D computer vision. We then discuss the various representation for 3D shapes, including their advantages and shortcomings, before focusing on deep point networks. We conclude this chapter by providing a road map to this thesis.

1.1 3D Deep Learning

We, humans live in a three-dimensional world. Recognizing objects around us and being able to interact with them is essential for our survival and sustenance. For example, understanding the shape and size of 3D objects and performing actions with them, such as lifting a coffee mug, sitting on a chair, speaking with our mobile phones and writing on a notebook are crucial parts of our lives.

For several decades, 3D visual computing has predominantly focused on single 3D models or small model collections where hand-crafted geometric algorithms have stood out. However, the development of new technologies and reconstruction algorithms, over the past few years has drastically increased the amount of accessible 3D data. Motivated by the far-reaching impact of dataset efforts such as the WordNet [23] and ImageNET [20], the whole computer vision community has made significant progress in collecting and curating large scale 3D dataset, both for single objects (e.g. ShapeNet [13], ObjectNet3D [90], ModelNET [88], ABC [46]) and 3D scenes (e.g. NYU Depth Dataset [72], SceneNN [37], ScanNet [18], MatterPort [12]). Such efforts compel us to redefine 3D visual computing from the perspective of big 3D data and to develop novel learning based geometric algorithms.



Indoor Scene Outdoor Scene

Figure 1.1: Problem tasks can vary depending on the type of 3D scene. For static indoor scenes predicting scene layout and understanding object relationships might seem important while for outdoor scenes object detection and tracking could take precedence.

1.1.1 Task Space in 3D Computer Vision

Problems in 3D computer vision can be categorized in many ways. One could classify them according to where the 3D data is accessed or produced in the algorithm, what kind of object representations are the algorithms using etc. However, in this dissertation, we categorize 3D computer vision tasks as either object-centric or scene-centric.

Object centric. Object centric tasks refer to problems where the analysis is done on the whole 3D shape or a region of the shape. Examples of major tasks include object classification, segmentation, correspondence estimation, shape abstraction, 3D reconstruction of single objects.

Scene centric. In scene centric tasks, the analysis is done on the entire 3D scene, which usually comprises of several 3D objects. These 3D scenes can either be indoor or outdoor (Figure 1.1). Depending on the type of 3D scene, the tasks can include object detection and classification, tracking, predicting scene layout, predicting object functionality, computing relationship between objects.

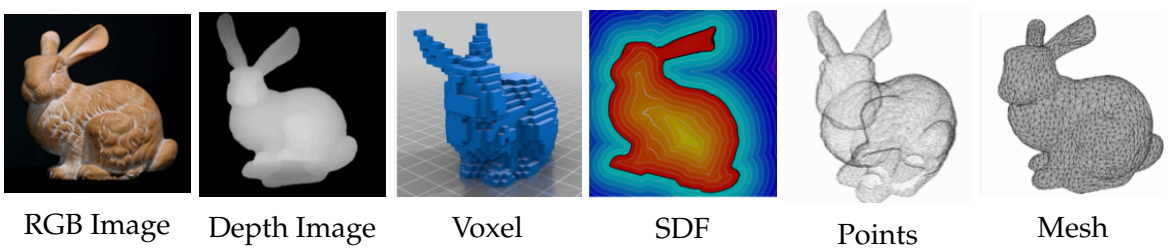


Figure 1.2: Popular 3D shape representations.

Although most of the algorithms discussed in this dissertation can be scaled to scene centric tasks, we primarily focus at object centric tasks.

1.1.2 3D Representation

Among all the digital representations we have for real physical objects, 3D is arguably the most expressive encoding. 3D objects can exist in a variety of representations like collection of RGB images, RGB-D images, volumetric grids, signed distance functions (SDF), point clouds and polygonal meshes (Figure 1.2). Each of these representations facilitate different application scenarios. Additionally, each type of data format has its specific properties, which can pose challenges to the design of learning algorithms.

For example, one advantage of structured representation like images and voxels is that basic operations like convolution and pooling are well defined. Although voxel based representations are memory intensive and suffer from poor resolution [88], there have been works which try to address these issues [82, 69]. On the other hand, unstructured representations like points and meshes are light weight and can represent high resolution models, but are not friendly to deep learning. However, with the recent success of point and mesh architectures [65, 66, 80, 59], convolution operations have been extended to these irregular structures enabling a host of new applications. Another popular representation that has lately gained attention is the signed distance function (SDF), where

instead of explicitly encoding the object geometry, we only encode the distance to the surface. Although SDFs can represent high resolution models and are friendly to convolution, they typically require iso-surface extraction for mesh generation using algorithms like marching cubes [55].

1.2 Roadmap

As mentioned earlier, in this dissertation we develop algorithms for deep point networks, specifically focusing on object centric applications.

The dissertation is organized as follows. We first provide a brief background on deep point networks including commonly used architectures, design choices and applications in Chapter 2. In Chapter 3, we present a novel point-surface loss function which captures sharp features in 3D shapes. Next, we describe an algorithm to mesh the output points of a deep point network using a mesh prior in Chapter 4. We explore its various applications and utility to points networks. Finally, in Chapter 5 we introduce a new high quality mesh dataset and discuss its applications before concluding in Chapter 6.

Chapter 2

Deep Point Networks

In this chapter, we first motivate and give background on deep point networks. We then discuss few challenges and design choices that one needs to take into consideration while design point networks. We then conclude this chapter by describing some common applications of deep point networks.

2.1 Motivation & Background

As discussed in the previous chapter, 3D data can usually be represented with different formats, including depth images, point clouds, meshes, volumetric grids and signed distance functions. As a commonly used format, point cloud representation preserves the original geometric information in 3D space without any discretization. Further, it is also the output format for a majority of 3D range scanners (e.g LiDAR, Kinect) and reconstruction algorithms, making it the preferred representation for many scene understanding related applications such as autonomous driving and robotics.

Recently, deep learning on point clouds has been attracting more and more attention, especially in the last five years. Several publicly available datasets are also released, such as ModelNet [88], ShapeNet [13], PartNet [57], ScanNet [18], ApolloCar3D [13], and the KITTI Vision Benchmark Suite [26]. These datasets have further boosted the research of deep learning on 3D point clouds, with an increasingly number of methods being proposed to address various problems related to point cloud processing, including 3D shape classification, 3D object detection and tracking, 3D point cloud segmentation, 3D point cloud registration, 6-DOF pose estimation, and 3D reconstruction to name a few.

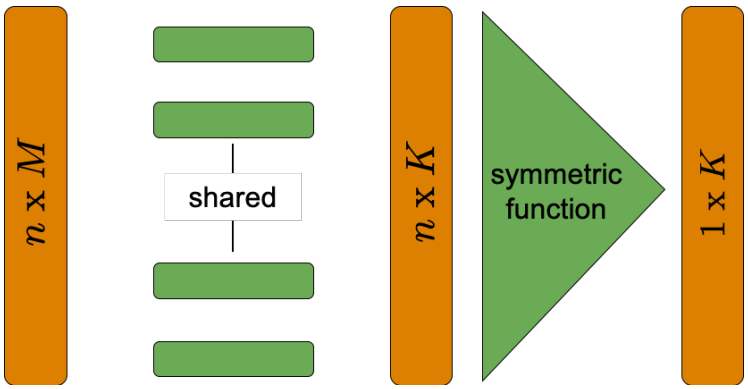


Figure 2.1: A PointNet architecture where MLPs convert high dimensional point features $\mathbb{R}^M$ into a $\mathbb{R}^K$. These features are subsequently aggregated into a K dimensional feature vector using symmetric functions such as max pooling.

2.2 Challenges & Design Choices

Although there has been tremendous amount of success in the applications of deep neural networks for image classification, segmentation and detection, extending these approaches to unstructured data such as point clouds is challenging. Below we discuss few challenges and design choice which arise in a point network architecture for any application.

2.2.1 Symmetric Functions

The first and foremost challenge in designing a network which can consume unordered point clouds is making the network invariant to point ordering. Unlike images, point clouds have an irregular structure making it difficult to extend basic operations like convolution and pooling. Although there are works which transform points clouds into more structured representation like voxels and collection of images, these unnecessarily increase the memory footprint while also introducing quantization artifacts that can obscure natural invariances of the data.

In order to make the network invariant to point ordering, an easy strategy could be to sort the input points. However, in high dimensional space there does not exist an ordering that is stable with respect to point permutation. For if such an ordering exists, it defines a mapping which preserves spatial proximity as the dimension reduces, which is difficult to achieve in a general case. Another approach could be to train a recurrent neural network (RNN) with multiple permutations of the input hoping that the network learns to be invariant to the point ordering. Although this may seem plausible, in practice this is difficult to scale to thousands of input points, which is quite common for point sets.

A common strategy which most works adopt is to first apply a set of functions $\{h_1, h_2, ..., h_n\}$ with shared parameters to each point separately and then aggregate those high-dimensional features through symmetric functions such as max pooling as shown in Figure 2.1 and Equation 2.1.

$$f(\{x_1, x_2, ..., x_n\}) \approx g(h_1(x_1), h_2(x_2), ...h_n(x_n)), \tag{2.1}$$

where $f \in \mathbb{R}^K$ is a feature vector for the entire pointset, $h : \mathbb{R}^M \rightarrow \mathbb{R}^K$ is approximated by multi-layer perceptron (MLP) and $g : (\mathbb{R}^K \text{x } .... \text{ x} \mathbb{R}^K) \rightarrow \mathbb{R}^K$ is a symmetric function. Although there exists other symmetric functions such as mean, minimum, multiplication, etc., maximum and sum are the most common and widely used.

2.2.2 Local vs Global Features

Unlike meshes, point clouds do not have any connectivity information associated with them. Unless the connectivity is borrowed from the ground truth mesh from which the point clouds were sampled, there are two approaches for carrying out convolutions on

point clouds. Depending on the application, convolution can either be performed on individual points or it can be performed on groups of points.

Point Features. The most simple pointnet architecture is where several MLPs which share weights are learnt across multiple layers to embed each point in a high-dimensional feature vector. These MLPs essentially learn local point features which then serve as building blocks for designing more complex architectures.

Global Features. Applications such as point cloud segmentation [85] and classification [65] require an understanding of the entire shape. Hence, networks for these applications first learn individual point features and later aggregate them using symmetric functions (section 2.2.1) to obtain a single global feature vector. These feature vectors may then pass through fully-connected layers before providing a final prediction. Quite often, these global features are concatenated with individual points or local features to further improve the networks performance.

Local Features. Designing point networks which require local neighbourhood information can be difficult. For example, networks which predict local geometric properties such as normals and curvatures [32] need to obtain features that are local to the point. However, computing an appropriate neighbourhood for performing convolution can itself be tricky, especially if the point set comes with non-uniform density in different areas.

Depending on the density and distribution of the input points, different strategies can be adopted for grouping points around a local neighbourhood to learn local features. If the points in the input are uniformly distributed, a simple strategy such as k nearest neighbours [85] or even computing a fixed number of closest points within a specific radius [32] can give good result. However, the networks performance can drastically drop if the same strategy is adopted for input that has non-uniform distribution of points. As

illustrated in Figure 2.2 using any of the above mentioned approaches would yield low performance.

To compute local features on non-uniform input points, two hierarchical methods were proposed by PointNet++ [66]. Multi-scale grouping (MSG) computes feature vectors using multiple scales but at a single resolution level and later concatenates these features into a single feature vector. Computing features at multiple scales enables the network to overcome any discrepancies introduced due to non-uniform density. Contrarily, multi-resolution grouping (MRG) concatenates feature vectors from two levels which represent two different resolutions. This is computationally more efficient as it avoids feature extraction in large scale neighbourhoods at the highest resolution level.

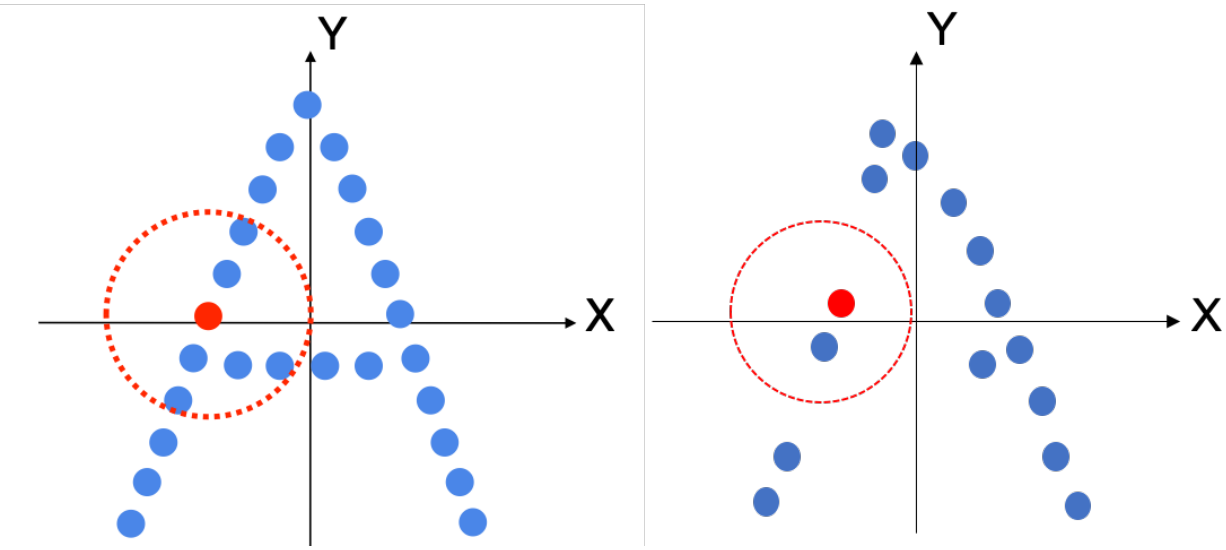


Figure 2.2: Grouping strategies for computing local point features for point networks. While simple strategies such as computing a fixed number of closest points within a specific radius can be used for uniform points distributions, blindly extending such techniques for input with non-uniform point distribution gives poor results.

2.2.3 Invariance to Geometric Transformation

Another obstacle in designing networks for geometric data such as point clouds is for the network to be invariant to any geometric transformation. Unlike 2D images, point clouds can undergo geometric transformations such as translation and rotation. Typically, such transformations should not affect the output result especially for applications like object detection, classification, segmentation, etc. A natural solution to this is to align all the input point clouds to a canonical space before extracting features using point networks. However, this limits the generalization power of the network. An alternative can be to use a spatial transformer [38]. Spatial transformers are mini point networks which predict a 3 x 3 affine transformation that can be directly used to transform the input point coordinates. Further, these spatial transformers are optimized during training and hence are specific to each application and perform better than manual alignment.

2.3 Applications for Point Networks.

PointNet [65] and PointNet++ [66] have been instrumental in the success of deep point networks. They have been the motivation behind many point network architectures. Below we discuss few of these applications. For a comprehensive list, please refer to [33].

Representation Learning refers to learning a compact representation of the input 3D shape. Given a point cloud, an autoencoder is trained in an unsupervised manner to learn an embed which accurately captures the entire shape [1, 2]. Such embeddings have several applications including classification, clustering, shape interpolation, shape analogies etc. Further, depending on the network architecture, training such an autoencoder can also assist computing shape correspondences. For example, networks which learn to deform 2D

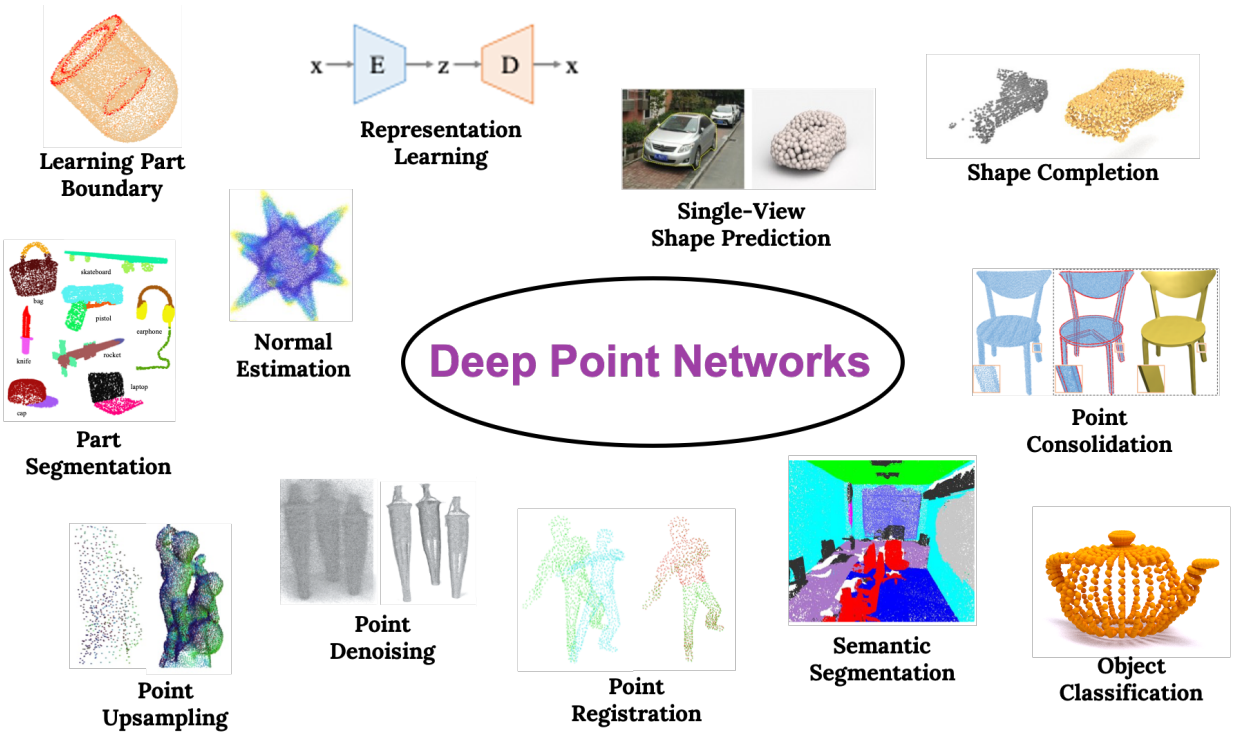


Figure 2.3: Applications of Point Networks

patches to fit the input shape [30, 93], have been shown to learn semantically meaningful correspondences across different shapes.

Single-View Shape Prediction or point-set generation refers to reconstructing a 3D point cloud from a single-view RGB image [22, 52]. This problem is ill-posed and extremely challenging as the network needs to learn and draw inferences about hidden or occluded parts of the 2D image. Surface reconstruction can then be performed on the output point clouds to obtain a final 3D mesh.

Part Segmentation involves predicting a part label for each point in the input point cloud [65, 85]. Point networks focusing on part segmentation are trained in a supervised manner using datasets with ground truth part labels. These networks often concatenate both global and local features to output the final point prediction.

Object Classification is the task of classifying the input point cloud into one of k categories, where k depends on the number of categories in the input dataset [65, 66]. Networks for object classification usually only require global features for predicting the object category.

Normal Estimation refers to computing the normal vector for each point in the input point cloud [32]. Normal vectors are usually computed using surface information available in mesh data. Computing normal vectors using only points is challenging as normal computation is sensitive to point density. Hence, networks which predict these local geometric properties primarily use local features and require sufficiently dense sampling to avoid artifacts due to spatial proximity of points that are geodesically remote.

Point Upsampling as the name suggests involves upsampling a sparse input point cloud [96, 94]. Classical surface reconstruction algorithms like Ball-Pivot algorithm [9] cannot accurately reconstruct surfaces from sparse point clouds as they heavily depend on the point density and distribution.

Given the numerous applications for deep point networks mentioned above, any fundamental contribution towards point cloud processing will prove to be a boon for the 3D deep learning community.

Chapter 3

Learning Shape Representation using Quadric Loss

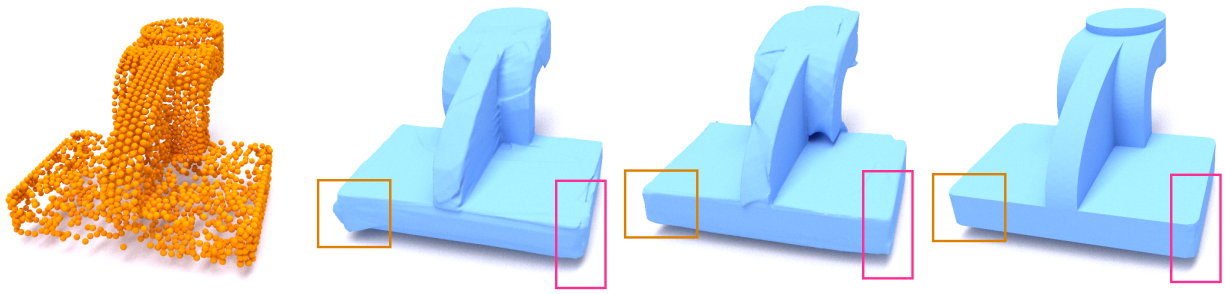
In this chapter, we first motivate the need for sharp features in 3D shapes and briefly describe the loss functions available today for 3D reconstruction. We then introduce quadric loss and discuss its geometric interpretation. Finally, we show experimental results highlighting the advantages and shortcomings of quadric loss before concluding the chapter.

3.1 Motivation

Following the tremendous success in image classification and detection, deep learning based techniques have been widely extended to 3D data, opening up numerous 3D applications such as 3D object classification, segmentation, shape representation and correspondence finding to name a few. In this chapter we focus on shape representation, particularly on learning a better embedding or shape representation of 3D models using an auto encoder.

Early 3D deep learning techniques use 2D and 3D convolution modules to design their network architectures. Recent techniques extend such convolution modules to handle irregular representations such as points [65, 66] and meshes [53, 19, 78, 68]. Together with these architectures, different loss functions have been proposed for 3D reconstruction. At a high level, they can be classified as being between two points (e.g., L1, Earth Mover Distance [22]) or between a point and a surface (e.g., surface loss [95]). Among these loss functions, Chamfer loss [95] has been widely used for reconstructing 3D models.

While these loss functions work well in maintaining the overall structure of the 3D model, they do not preserve high-frequency information such as edges and corners. To address this issue, we propose a novel loss function, Quadric loss, for preserving such detailed structures. Inspired by mesh simplification techniques, Quadric loss is defined as the sum of squared distances between a reconstructed point and planes defined by triangles



(a) Input Point Cloud (b) Chamfer (c) Chamfer+Quadric (d) Original Mesh

Figure 3.1: (a) Input point cloud reconstructed using an auto-encoder network with (b) Chamfer loss alone and (c) Chamfer + Quadric loss. Reconstructed meshes are generated using Poisson surface reconstruction on output point cloud.

incident to its corresponding point in the input mesh. Intuitively, the Quadric loss penalizes the displacement of points along the normal direction of those planes, maintaining sharp edges and corners as shown in Figure 3.1.

To demonstrate the benefits of Quadric loss, we conduct experiments with 3D CAD models, and compare various loss functions both qualitatively and quantitatively. Overall, we find that the combination of Chamfer and our Quadric loss shows the best result, since Chamfer loss maintains the overall structure and point distribution, while the Quadric loss preserves sharp features.

The main contributions of this chapter are as follows:

- We propose a new point-surface loss named *Quadric loss*, which preserves sharp features such as corners and edges in the reconstructed models. It is fast, easy to compute and is fully differentiable. It does not introduce any hyperparameters and can be used with most existing point or mesh based architectures without modification.
- We evaluate our loss function extensively and also provide its geometric interpretation.

- We compare our Quadric loss with other point-surface loss functions and the popular Chamfer loss and discuss in detail the merit and demerit of each.

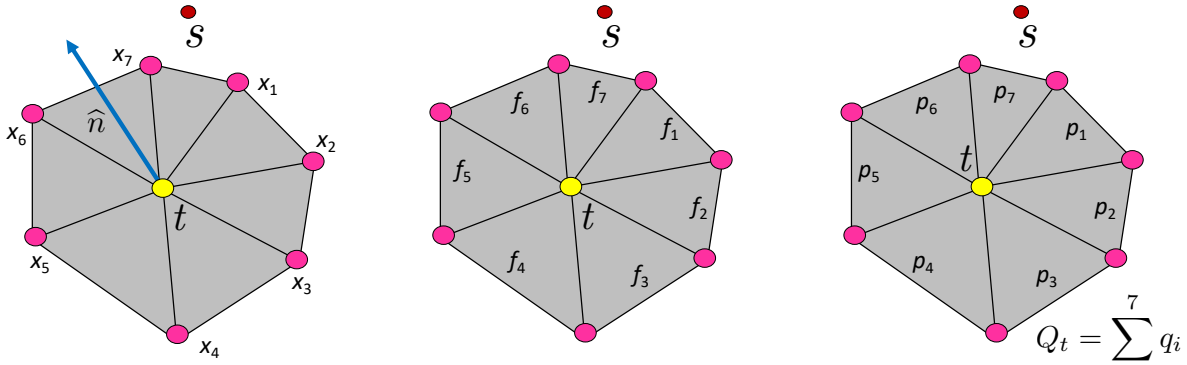
3.2 Related Work

3.2.1 Learning Shape Representation

There is a rich literature for learning compact 3D shape representations using deep learning techniques. Prior works [75, 50, 27, 87] have used image and voxel based representations of 3D models to learn a discriminative representation for the task of 3D object recognition, classification and generation. Although their structured representations facilitate the use of traditional 2D and 3D convolution, they are not readily available for handling complex and high resolution models. On the other hand, part-based approaches [49, 58, 89] can produce shapes with complex structures, but the level of detail is restricted to the components and primitives used.

Recently, convolution has been extended to more unstructured representations like 3D point datasets and meshes. PointNet [65] and PointNet++ [66] have been widely used as an encoder to achieve superior performance on various tasks such as object classification [65, 85], segmentation [65, 85], point set generation [1, 93], shape correspondence [31] etc. Mesh based networks have also been used to learn embeddings for shape completion [53, 19] and shape deformation [78, 68].

Since point and mesh based representations, when compared to voxel-based representations, are light-weight, flexible in terms of reconstructing complex models and scale well to high resolution models, we propose a loss function which can be used by such networks to further enhance the embedding and reconstruction quality of 3D models.



$$L_{normal} = \sum_{i=1}^{7} |(s - x_i) \cdot \hat{n}| \qquad L_{surface} = min_{i=[1,7]} \Phi(s, f_i) \qquad L_{quad} = s^T Q_t s$$

(a) Normal Loss (b) Surface Loss (c) Quadric Loss

Figure 3.2: Computation of point-surface losses. Let the the reconstructed point s correspond to the point t in the input mesh. (a) Normal loss computes the inner product between the edge formed by s and x_i and the ground truth normal vector $\hat{n}$ at t; (b) Surface loss computes the point-triangle distance Φ between s and f, where f represents a triangle and *not a plane*, and takes the minimum of them with different triangles; (c) Quadric loss (our contribution) computes the sum of the square of the distance between s and each of the plane p ($p = [a, b, c, d]^T$) formed by the triangle incident at t using the quadric matrix q_i which is computed as $q_i = p_i p_i^T$. Please see Eq. 3.4 for more details.

3.2.2 3D Reconstruction Losses

Losses commonly used with point and mesh based networks for 3D reconstruction can be broadly classified into two categories - between two points or between a point and a surface.

Point based Loss: Point based loss functions compute the dissimilarity between two pointset distributions. Let S_1 and S_2 be the input and output point distributions. Losses like L1 [19] and L2 [31, 53] require both one-to-one correspondence and the cardinality of the two pointsets to be the same (Equation 3.1).

$$\mathcal{L}_{L1}(S_1, S_2) = \sum_{\substack{x \in S_1 \\ y \in S_2}} |x - y| \qquad \text{and} \qquad \mathcal{L}_{L2}(S_1, S_2) = \sum_{\substack{x \in S_1 \\ y \in S_2}} ||x - y||_2 \tag{3.1}$$

Earth movers distance (EMD) or Wasserstein metric [22] is similar to these losses as it requires the input cardinality between pointsets to be the same. It solves an optimization problem where it computes a bijection ϕ between the two pointsets (Equation 3.2).

$$\mathcal{L}_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x_1 \in S_1} ||x - \phi(x)||_2^2 \tag{3.2}$$

However, a major drawback of EMD is that it is both memory and compute intensive, hence is usually approximated [22]. Chamfer distance (CD) [22, 81, 30], which has become a standard for reconstructing 3D objects, computes the shortest distance of each point in one pointset to the other pointset (Equation 3.3).

$$\mathcal{L}_{CD}(S_1, S_2) = \sum_{x \in S_1} ||x - y||_2^2 + \sum_{y \in S_2} ||x - y||_2^2 \tag{3.3}$$

This distance is computed in both directions. It does not require the cardinality of the input points to be the same nor does it require any one-to-one correspondence. Although CD works well at recovering the overall structure, it does not preserve sharp features like corners and edges, and often results in collapse of smaller structures [19].

Point-Surface based Loss: Point-surface based loss functions try to minimize the distance between the output reconstructed point and the input surface. Yu et al. [95] propose *surface-loss* (point-triangle), which computes the minimum of the shortest distances between an output point and each triangle in a subset of triangles defining the input mesh (Figure 3.2b). Similar to *surface-loss*, Yu et al. [95] also propose *edge-loss*, which requires

the edges in the input model to be manually annotated. Wang et al. [81] propose *normal-loss* to incorporate higher-order features in their reconstruction. It minimizes the inner product of the edge formed from the output point and the neighbours of the corresponding input point with its normal vector. In other words, it requires the edges between the output point and the neighbours of the corresponding input point to be orthogonal to the ground truth normal vector (Figure 3.2a).

Departing from these prior approaches, we propose a new point-surface based loss function named *quadric loss*, which encourages sharp corners and edges to be preserved in the output reconstruction (Figure 3.1). Unlike edge-loss, quadric loss does not require the edges to be annotated in the input models. Quadric loss minimizes the distances between the output point and the planes defined by the triangles incident to its corresponding point in the input mesh (Figure 3.2c). It is fast and easy to compute as oppose to surface loss for which one needs to consider all the seven cases, as the point which minimizes the point-triangle distance can be on the 3 edges, 3 vertices or inside the triangle [95]. Quadric loss is also differentiable making it amiable for training via back propagation.

3.3 Quadric Loss

Quadric error metric was originally proposed for mesh simplification [70, 25], i.e, the task of reducing of a mesh with a high polygon count to a low polygon count while preserving as much visual geometric detail as possible. Quadric error computes the squared distance between a point and a plane in $\mathbb{R}^3$. Inspired by this quadric error metric, we propose *quadric loss*, a point-surface loss function, which penalizes the reconstructed points in the normal direction, thereby preserving sharp edges and corners in the output reconstruction as shown in Figure 3.1.

3.3.1 Background

Let a point s be represented in homogeneous coordinates $[x, y, z, 1]^T$, and a plane p be represented as $[a, b, c, d]^T$ where $a^2 + b^2 + c^2 = 1$. The distance of s from p is given by $ax + by + cz + d$, which can be computed as $p^T s$. The square of the distance of s from p is given by

$$(p^T s)^2 = (p^T s)(p^T s) = s^T (pp^T) s = s^T Q_p s, \tag{3.4}$$

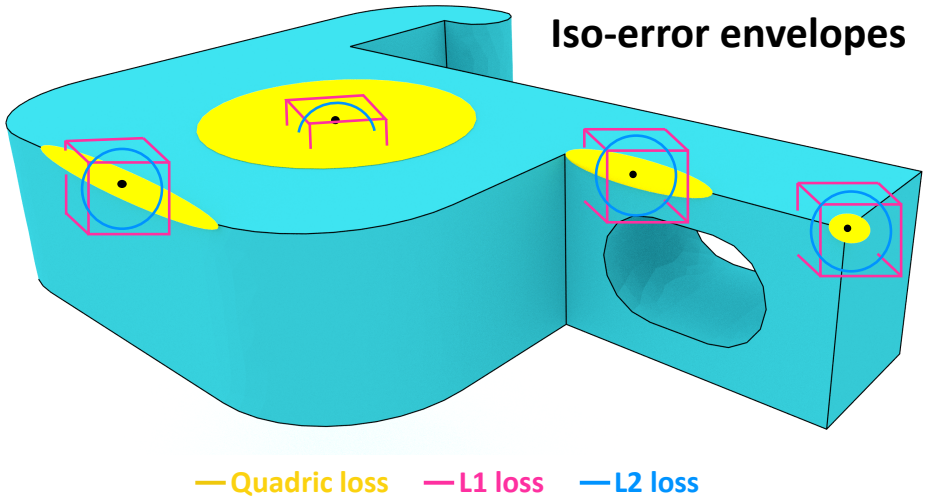


Figure 3.3: Geometric Interpretation of quadric loss. Quadric loss is an *ellipsoidal loss* and it penalizes the reconstructed points more in the normal direction. Here we show the iso-error envelope of Quadric, L1 and L2. For illustration purposes, we draw iso-errors in 2D on few points (yellow) on the input surface. Points lying on flat planes would ideally have ellipsoids with 0 minor axis and ∞ major axes lengths, i.e the reconstructed points can be placed anywhere on the plane. Note that the ellipsoid for points on sharp features like corners is very small compared to L1 and L2, ensuring the reconstructed points to preserve such features.

where Q_p is a symmetric matrix called the quadric matrix [25], determined only by the plane and not by the point. Given a set of planes $p_1, p_2, ...p_k$, the sum of the squared distance of s from this set of planes is given by

$$\Delta(s) = \sum s^T Q_i s = s^T (\sum Q_i) s = s^T Q s, \quad where \quad Q = \sum Q_i. \tag{3.5}$$

It should be clear that in a mesh, the quadric error of a vertex s from the planes defined by the triangles incident on s is zero.

3.3.2 Efficient Computation

Given an input mesh $\mathcal{M}$ with $\mathcal{V}_{in} \in \mathbb{R}^{N\text{x}3}$ vertices and a set of reconstructed points $\mathcal{V}_{out} \in \mathbb{R}^{N\text{x}3}$, let s be a reconstructed point corresponding to input vertex t. We want s to be on all the triangles incident on t just as t is on those planes in the input mesh. So the quadric error of s from the planes defined by the triangles incident on its corresponding point t, namely $s^T Q s$, has to be minimized. We call $s^T Q s$ as the quadric loss, which we compute between $\mathcal{V}_{in}$ and $\mathcal{V}_{out}$ as the following:

$$\mathcal{L}_{quad}(\mathcal{V}_{in}, \mathcal{V}_{out}) = \frac{1}{N} \sum_{\substack{s \in \mathcal{V}_{out} \\ t \in \mathcal{V}_{in}}} s^T Q_t s \tag{3.6}$$

3.3.3 Geometric Interpretation

The iso-value surfaces $s^T Q s$ defined by the quadric matrix Q at the input vertex t, represents a family of ellipsoids centered at t, for which one of the three axes corresponds to the normal vector of the surface at t. The length of the other two axes are inversely

proportional to the curvature of the surface in those directions. For example, in a planar region, the length of the ellipsoidal axes is infinity along the plane and zero along the normal vector direction. In other words, the reconstructed point can be anywhere on the plane, but any displacement along the normal vector direction will introduce more quadric loss. For vertices along a sharp, straight edge of a 3D model, the quadric error ellipsoid will have infinite length along the edge and zero length for the other two axes. In other words, the reconstructed point can be placed anywhere along the straight edge for the quadric error to be still zero, but any displacement away from the edge will incur a loss. A similar argument holds for a pointed corner of a 3D model. The quadric ellipsoid will be very small, restricting the freedom of placement of the reconstructed point as shown in Figure 3.3. Hence, unlike Chamfer and L2 loss which are *spherical losses* - points equidistant from the input vertex have equal loss, quadric loss is an *ellipsoidal loss* which penalizes displacement of points more in the normal direction.

3.4 Experiments

In this section, we present the results of training an auto-encoder with various point-surface loss functions. Specifically, we compare our quadric loss with surface loss [95] and normal loss [81] for the task of shape reconstruction. We analyze the reconstruction results both qualitatively and quantitatively, and also compare our proposed loss with the popular Chamfer loss.

3.4.1 Dataset

To train the autoencoder, we use the recently published ABC dataset [46]. Although this dataset contains more than 1 million high quality CAD models of mechanical parts,

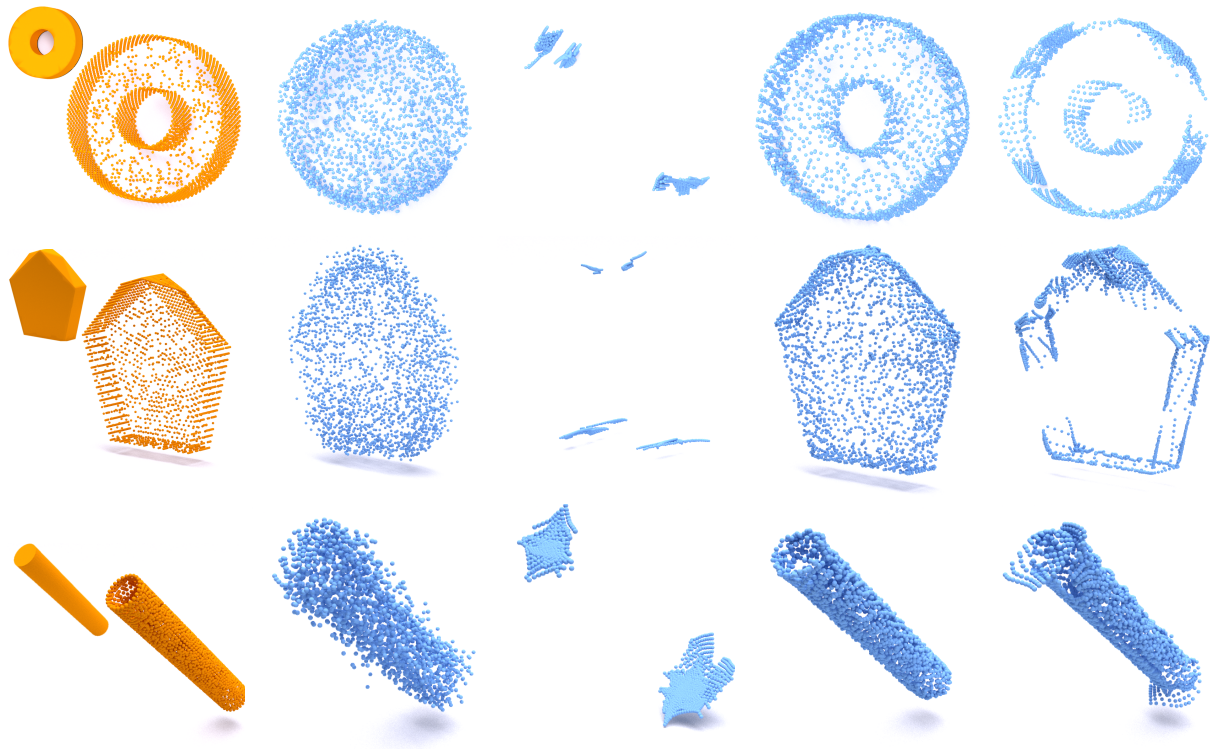


Figure 3.4: Effect of Point-Surface loss. Reconstruction results (2500 points) on example 3D models from the test set with different loss functions. In comparison to Chamfer focusing on preserving the input point distribution, our quadric loss encourages points to be on edges and corners. On flat planes (like the top and bottom faces of the cylinders in the bottom row) reconstructed points minimize the quadric error by lying on the plane, but can be outside the ground truth model. Such artifacts can be avoided by the combination of quadric and Chamfer loss (top row of Figure 3.5).

we randomly selected 5000 CAD models for our experiment. The reason of using ABC dataset over other 3D shape repositories like ShapeNet [13] and ModelNet40 [88] is the presence of sharp edges and corners, which are commonly found in mechanical parts (Figure 3.5). As some of the models comprised of multiple disconnected components, we separated each model into its connected components such that each model has a single mesh. This increased our dataset size to 8064 models. We also simplified the models using Q-slim [25] to reduce the vertex count to 2500 vertices, and centered and normalized them to a unit sphere. We randomly split the data to get a distribution of 90% for training and 10% for testing.

3.4.2 Network & Implementation Details

Although Quadric Loss can potentially be used with any point or mesh based network, we use an auto-encoder based network and analyze the reconstruction quality during shape reconstruction. We use the encoder from Dynamic Graph CNN (DGCNN) [85], which performs convolution over k-nearest neighbours in the feature space at every layer and is currently the state of the art for point cloud analysis. Specifically, we use the classification architecture without the spatial transformer and the fully connected layers to encode a point cloud of 2500 vertices into a latent vector dimension of 1024.

For the decoder we use AtlasNet [30], which takes in the 1024 embedding from the DGCNN encoder and generates an output surface using N learnt parameterizations. We follow the same training strategy as AtlasNet, which is to sample the learned parameterizations at every training step to avoid over-fitting. For all the experiments in this paper, we use this auto-encoder architecture with k = 20, N = 25 and an output point cloud size of 2500.

In order to compare the three point-surface loss functions, we train 4 networks - one with CD + surface loss, one with CD + normal loss, one with CD + quadric loss and one with CD alone. To compute the three losses (surface, normal and quadric), we use the correspondences found from Chamfer distance. For all the experiments we use Adam [44] optimizer with a batch size of 16. The learning rate was set to 0.001 for all losses except the networks trained with quadric loss for which we found a slower learning rate of 0.0001 to be most effective. All learning rates were multiplied by 0.8 every 100 epochs. For a fair comparison we train all the networks to the same number of epochs and we also ensure that the total loss in each network is an equal contribution of both the loss functions by weighting the terms appropriately. All the code was implemented in Pytorch and training was performed on NVIDIA TITAN Xp GPU.

Losses	CD		Metro	
	median	max	median	max
Normal loss	397.09	1750.6	10.65	28.38
Surface loss	21.86	398.85	6.11	24.93
Quadric loss	9.44	217.5	3.18	20.80
Chamfer loss	**1.97**	40.87	3.13	19.08
Normal + Chamfer loss	2.97	39.83	3.38	19.21
Surface + Chamfer loss	2.23	37.04	3.16	18.87
Quadric + Chamfer loss	2.21	**36.78**	**2.96**	**18.80**

Table 3.1: 3D reconstruction results on models from the test set. We compare different loss functions using Chamfer distance (CD), computed on 2500 points, multiplied by 10^3 and Metro error [17], multiplied by 10. Among all four losses, Chamfer loss best preserves the overall structure and point distribution which is reflected in its low CD and Metro values. Quadric loss preserves sharp edges and corners (Fig. 3.4) but has a higher CD when compared to Chamfer loss. Combining quadric with Chamfer achieves best results.

3.4.3 Evaluation Metric

To evaluate the quality of the reconstructed shapes, we compare it with the ground truth shapes using two criteria. First, we compare the Chamfer distance (CD) [22] between the input and output point clouds. CD alone is a necessary but not a sufficient condition for a good reconstruction; CD can be minimized by assigning just one point in one point cloud to a cluster of points in the other point cloud. Hence, we also compare the Metro error between the input and output meshes using the publicly available software [17]. Simply put, it computes the Euclidean distance between two meshes by sampling points on them. We report the maximum distance between the two meshes because outliers dictate the visual quality and fidelity of the reconstructed mesh.

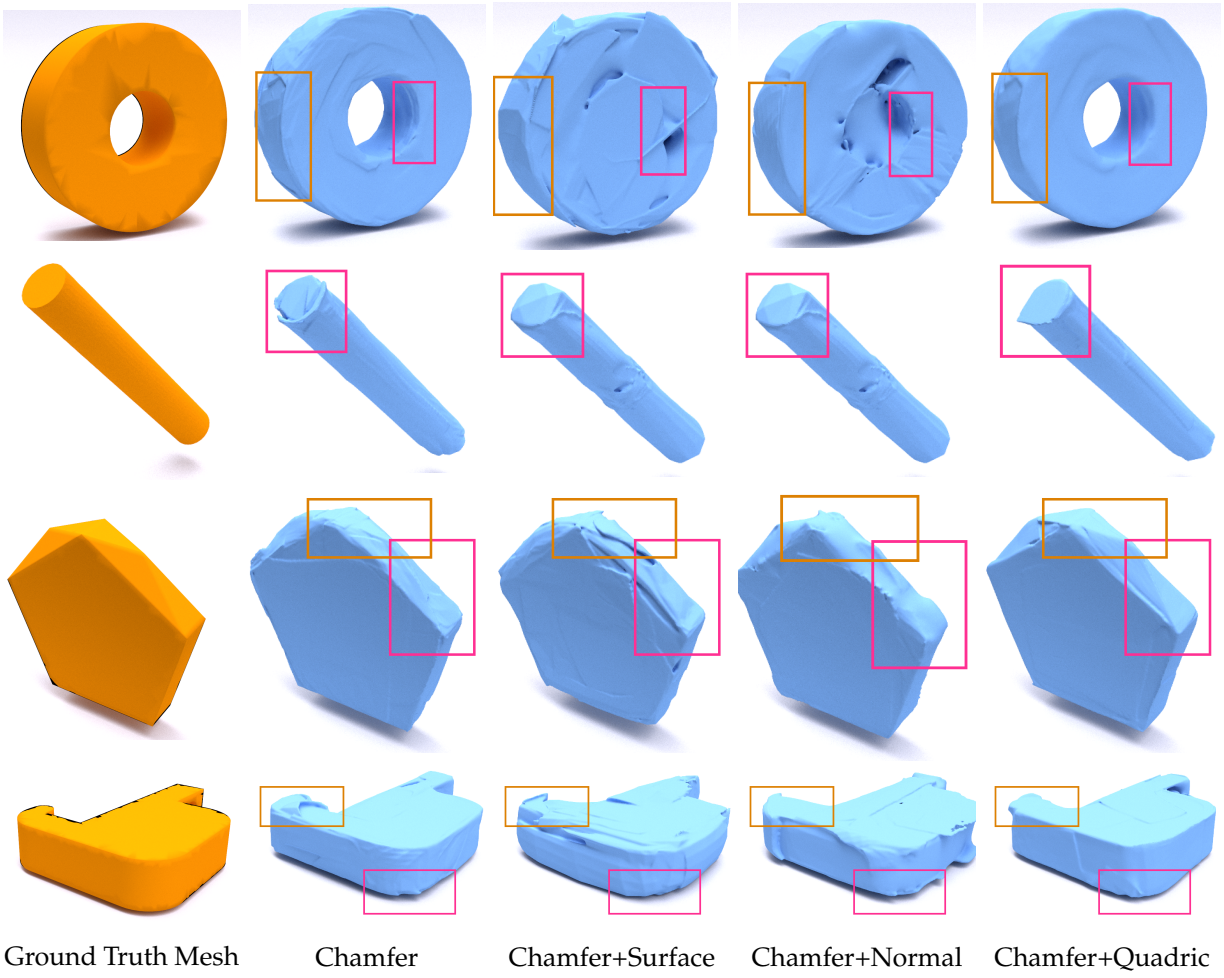


Figure 3.5: Reconstruction results of 3D models from the test set. To obtain a mesh from the reconstructed point clouds, we follow a similar procedure as AtlasNet [30]. i.e. we shoot rays at the model from infinity to obtain a dense sample of points followed by Poisson surface reconstruction (PSR) [42]. Chamfer loss when added to surface, normal and quadric loss improves the reconstruction result as compared to them individually. Note, sharp edges and corners are achieved with quadric and Chamfer togather.

3.4.4 Shape Reconstruction

We evaluate the learnt embedding by analyzing the reconstruction quality of the 3D models. We report the quantitative results in Table 3.1 where the results are from computing the median and the maximum values of all models in the test set.

Reconstruction without Chamfer: In order to study the effect of various point-surface losses, we train the auto encoder with surface, normal, and quadric losses independently

without chamfer loss. We report the qualitative results in Figure 3.4. As compared to surface and normal loss, quadric loss alone reconstructs the models much better. Through our extensive experiments we find training of normal loss also to be much difficult as compared to surface and quadric loss. i.e. it does not converge. Quadric loss in comparison to surface loss preserves the sharp features better while surface loss is able to achieve better point distribution. As surface loss computes the closest triangle to the reconstructed point and minimizes that distance, it is difficult for it to reconstruct sharp features like edges and corners.

As quadric loss encourages more reconstructed points to lie along the edges and corners, it has a higher CD than network trained with Chamfer loss alone. Also, notice the small difference in Metro error between Chamfer and quadric loss. This is because Metro error is computed by sampling the meshes and not the points (like CD). Also, quadric loss with AtlasNet decoder is able to reconstruct the patches (N learnt parameterizations) close to the input surface. This demonstrates that Metro error does not care about the point distribution as long as the output mesh surface is close to the input mesh surface. Hence, a good reconstruction should preserve both the point distribution (low CD) and overall structure (low Metro).

Reconstruction with Chamfer: Chamfer loss when added improves reconstructions based on surface, normal, or quadric losses (Figure 3.5). Quadric with chamfer achieves the best reconstruction results overall. Addition of quadric to Chamfer loss further reduces the maximum CD from 40.87 to 36.78. This is because as Chamfer loss tries to preserve the point distribution, quadric loss tries to preserve sharp features like edges and corners. Hence, models reconstructed using both quadric and Chamfer enjoy best of both worlds - sharp features and good point distribution.

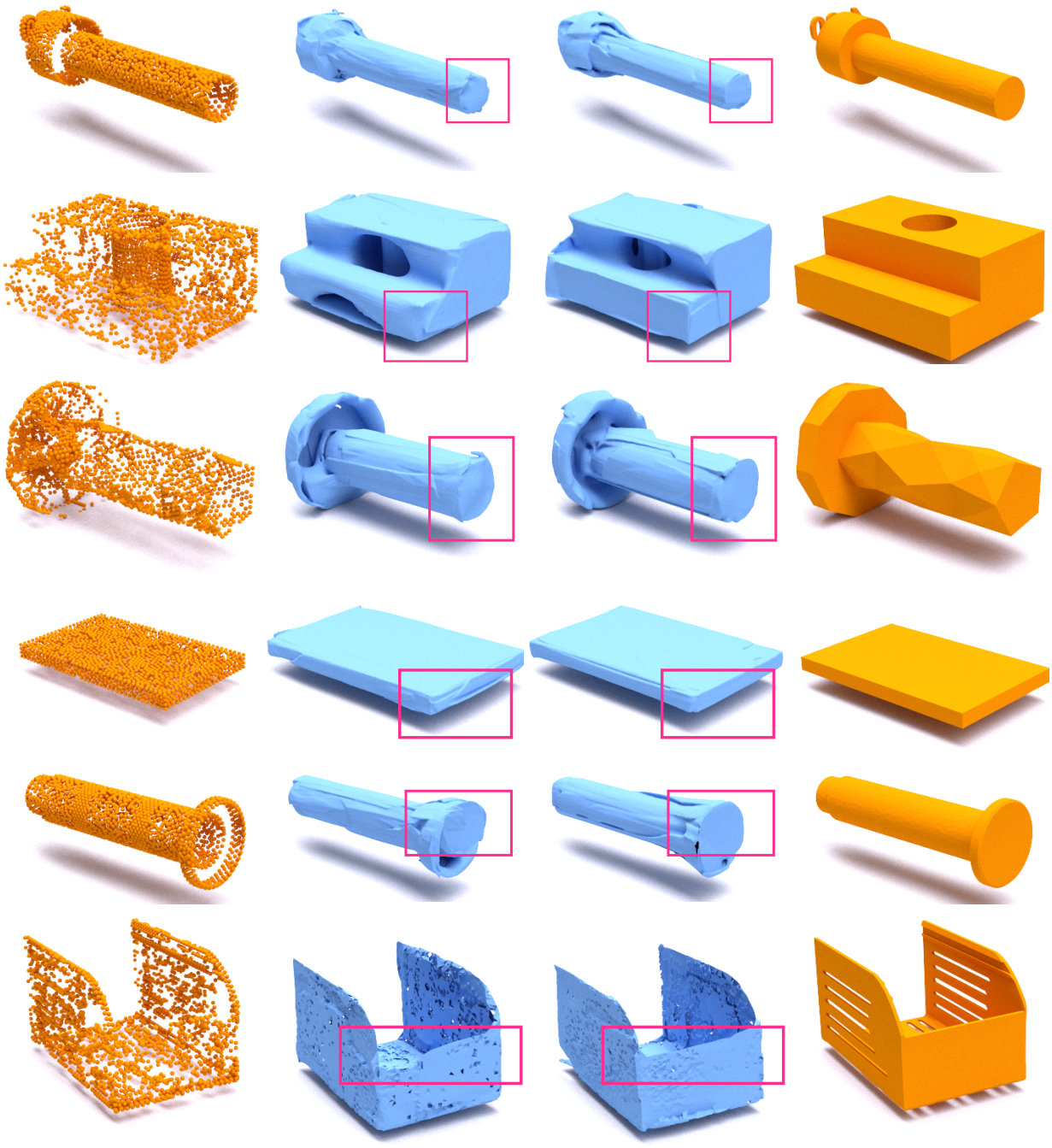


Figure 3.6: More qualitative comparison results.

3.4.5 More Qualitative Results

We also provide more qualitative results of using Quadric loss along with Chamfer loss for reconstructing 3D models. As Quadric loss is an ellipsoidal loss which does not care

about point distribution and only tries to preserve sharp features like edges and corners, it needs to be accompanied by a spherical loss like Chamfer loss to complement it. In Figure 3.6 we show the reconstruction result of models from the test set with Chamfer loss alone and Chamfer with Quadric loss.

3.5 Discussion

In this chapter we propose a new point-surface loss function, named quadric loss, which penalizes the displacement of points in the normal direction thereby preserving sharp features like edges and corners in the reconstructed models. Quadric loss is easy to compute, fully differentiable and can be integrated into most point and mesh based architectures. Quadric loss can also successfully reconstruct models having no sharp features. However, as quadric loss is an *ellipsoidal loss*, it cannot preserve the input point distribution. For points on the planar faces of a surface, since the quadric loss is zero anywhere on the plane, the reconstructed points may lie outside the extents of the planar face. Hence, quadric loss should always be accompanied with a *spherical loss* like chamfer loss which preserves the input point distribution. Note that Chamfer has its own weakness; its value could be minimized by assigning one point to a cluster of points. Depending on the application, these two losses could be weighted appropriately. Since Chamfer and quadric loss functions complement each other, combining these two loss functions achieve better embedding than using any one of them.

Chapter 4

Surface Reconstruction using GAM

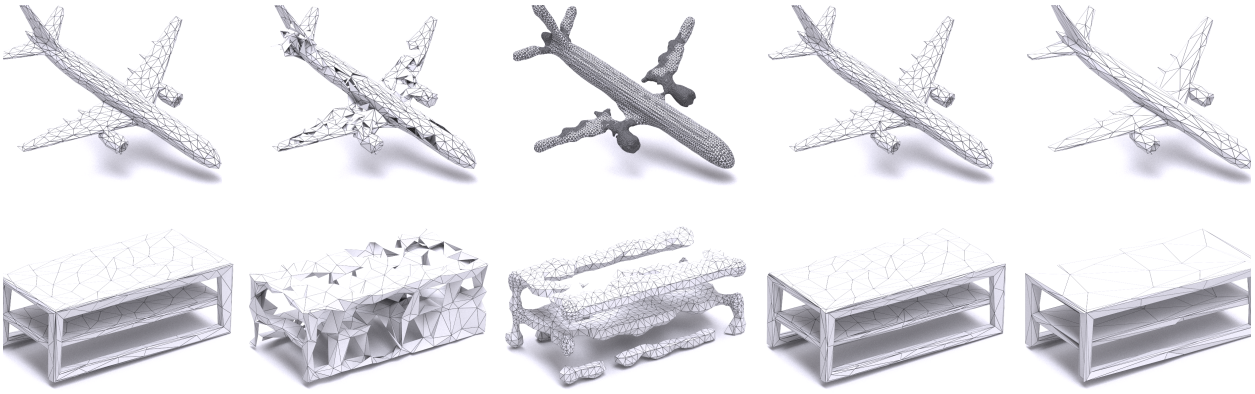
In this chapter, we first motivate and give some background on reconstructing a surface for the output points of a point network. We then introduce the proposed surface reconstruction method, GAM and analyze its various properties. Finally, we show the advantages of GAM for single-view shape prediction and fair evaluation of point network before discussing the limitation and concluding the chapter.

4.1 Motivation & Introduction

Meshes are a natural choice for representing 3D shapes as they can describe complex topologies and surface details while being memory efficient. They are also essential to applications like rendering, simulation, shape analysis and 3D printing to name a few. In this work, we focus on reconstructing surface meshes for point sets which are the output of point reconstruction networks.

Generating surfaces from point clouds has a rich history in computer graphics. Loosely speaking, surface reconstruction requires computing a surface for a set of input points such that those points lie on the reconstructed surface [7, 8, 29]. Such points usually come from 3D range scan data which inherently have noise [8]. For this reason most reconstruction algorithms tend to generate an approximate surface where the input points are not the vertices of the final mesh [42, 43, 3, 61].

However, with the recent success of deep point networks [65, 66], many learning based geometric algorithms have started to output 3D point clouds [22, 1, 52]. Reconstructing a surface connecting all these output points would not only help in visualizing and evaluating these point networks but could also assist in training them end-to-end for 3D surface prediction.



(a) Ground Truth (b) BPA (c) SPR (d) Ours (e) Ours(30%pts)

Figure 4.1: Surface Reconstruction. Surfaces reconstructed using all the original vertices with (b) BPA (c) SPR and (d) our method. We reconstruct the same mesh as the input mesh and preserve the geometry and topology even with 30% of the original points. Please zoom in for details.

Keeping these ideas in mind, in this chapter we propose *GAM - guided and augmented meshing*, an automatic meshing algorithm for point clouds ("reconstructed/output points") which are typically the output of point based networks. GAM guarantees correct geometry and topology of the final surface while preserving as much geometric features as maintained by the output points of the network. Furthermore, GAM can be used both in post-processing to mesh the output points or to train the point network to directly optimize the vertex positions of the final 3D mesh.

The key insight of GAM is to use a mesh prior with similar topology as the ground truth shape for reconstructing a surface for the output points. In our method, the output mesh is *guided* by the mesh prior through a process of *augmenting* the mesh prior vertices with the output points, and iteratively removing all the mesh prior vertices while retaining only its topology. This process results in an output mesh that has only the output points but with the topology of the mesh prior. Hence, unlike traditional surface reconstruction problems where the objective is to reconstruct an approximating surface using *only* a point cloud, our goal is to use both a mesh prior and the output points to generate a surface which faithfully preserves the topology and the output geometric features. We

demonstrate this through single-view 3D reconstruction (SVR) where we use meshes from implicit networks as mesh priors, to generate surfaces for the output points of a point reconstruction network. By combining the outputs of these two networks, we show that the result not only performs better than the individual reconstruction methods, but also outperforms priors works (Figure 4.11). Further, while most SVR methods fail to exploit the mesh representation and generate surfaces with uniform distribution of vertices, even in low curvature and planar regions, we show that by training point reconstruction networks with GAM, we can optimize the vertex position to generate adaptive meshes [74] with arbitrary topology.

In addition to SVR, we show the advantages of GAM for evaluating point networks. Unlike existing surface reconstruction methods like Ball-Pivot Algorithm (BPA) [9] and Screened-Poisson Reconstruction (SPR) [43], GAM is insensitive to both density and distribution of points. This makes GAM an ideal candidate for meshing the output points from a variety of point networks [84, 54, 2]. Using the ground truth shapes as mesh priors, we show that surfaces reconstructed by GAM are more reliable and should also be used to analyze the performance of point reconstruction networks rather than solely relying on the output points.

4.2 Related Work

4.2.1 Surface Generation for Deep Networks

One straightforward way of reconstructing a surface for the output points from a deep network is for the network to implicitly encode the point ordering such that the same input connectivity can be used for the output points. Such networks are typically mesh based networks [53, 78, 68, 77] which use L1 or L2 reconstruction loss between the input

and output points. These methods require the number of points in the input and output to be the same. Other methods use a fixed template mesh which is deformed to match the ground truth 3D shape and later the connectivity of the template is used as a surrogate for the output points [30, 81, 39, 24, 74]. Although these methods achieve impressive results, they limit the topology of the reconstructed mesh to that of the template. Further, extending these approaches to existing point networks by enforcing one-to-one correspondence (i.e. via L1 or L2 loss) reduces the network's performance.

Point based networks, which do not enforce one-to-one correspondences between the input and output points, use analytical surface reconstruction techniques like BPA and SPR to reconstruct surfaces [96, 95, 30, 67]. While BPA does not introduce new points, making it an ideal choice for surface generation, it cannot guarantee correct topology. It is extremely sensitive to the point distribution and radius of the pivoting ball and can even produce results without connecting all the points (Figure 4.1b). On the other hand, SPR is an interpolating technique which is robust to noise but cannot guarantee correct geometry as it fails to capture sharp features (Figure 4.1c). Further, SPR requires accurate normals, which if asked from the network, diminishes the network's performance [30]. Although other reconstruction [11, 29, 61] and remeshing [10] techniques have been proposed, they are more complex and non-trivial to be incorporated inside point networks.

We propose a simple surface generation algorithm that can be used both to post-process the output points of point networks and can be incorporated inside these point network to train them end-to-end for surface prediction. Our method guarantees correct geometry and topology by connecting all the output points and does not require any additional information such as normals. It only uses a mesh prior as a guide which depending on the application, can either be generated using other reconstruction methods or is already present. Furthermore, our method does not introduce any additional points and is insensitive to both the number and distribution of output points. It even works on non-

manifold meshes, a common feature in popular 3D shape repositories like ShapeNET [13].

4.2.2 Single View Reconstruction

A variety of representations have been explored for predicting 3D shapes from a single view image.

Voxels & Points. Methods which reconstruct voxels [15, 87, 92], while intuitive, are often limited by voxel resolution, resulting in missing details. Although techniques like Octree [69, 82] help scale to higher resolutions, they are complex and usually suffer from voxel-based discretization artifacts. Points are popular alternatives as they are light weight, flexible and can describe fine geometry [22, 52, 93]. However, they require meshing [9, 43] as a post-processing step to generate the actual 3D surface. Further, these meshing methods often require heavy parameter tuning to reconstruct surfaces with correct topology and geometry.

Fixed Topology Methods. A few methods [81, 74] use graph convolutions to directly predict triangle meshes. They use template models for 3D supervision which constraints the topology of the final mesh to that of the template (usually genus zero). Impressive 3D shape reconstruction has also been realized without 3D supervision [41, 39, 40]. However, they still rely on fixed or category-specific templates restricting the final mesh topology.

Arbitrary Topology Methods. There have been efforts to generate meshes with complex topologies. Pan et al. [62] propose a topology modification network which progressively removes faces that have high error from a zero genus template. Methods which represent shapes using multiple, possibly overlapping patches [30, 83] can also reconstruct models with arbitrary topologies. However, these intersecting, overlapping patches with open

boundaries are not suitable for downstream applications. Further, there are works which formulate SVR as a two-stage problem - shape retrieval and deformation [64, 48, 47]. While such methods can also produce meshes with arbitrary topology, they are limited to the diversity of shapes they can reconstruct by the models in the repository.

Implicit Methods. Recently, signed distance functions [63, 91, 14] have also been used for 3D shapes. Although these methods can represent meshes with arbitrary topology, they fail to capture fine details and reconstruct smooth meshes. They optimize auxiliary losses defined on intermediate representations and require an additional post-processing step like marching cubes [55] for mesh extraction. Further, they generate dense uniform points even in low curvature and planar regions and thus do not exploit the advantages of a mesh representation.

Hybrid Methods. Hybrid methods combine the benefits of two or more representations. Liao et al. [51] combine voxel and mesh representation by proposing a differentiable marching cubes algorithm to convert the output of a volume decoder to a mesh and optimized the network using geometric losses. Gkioxari et al. [28] improved upon this by first obtaining a coarse voxel prediction with correct topology and then refining it using graph convolutions to recover details. Tang et al. [79] proposed to combine all three - point, voxel and mesh representations for SVR where the skeletal points predicted from the RGB image is converted to a coarse volume using voxelization before being refined using a series of 3D and graph convolutions to produce an output mesh.

Our Approach. We propose a surface reconstruction technique that combines the benefits of point and implicit representation. We combine the output of point networks which are good at geometry (does not have topology), and meshes from implicit networks which are good at topology (maybe bad at geometry) to reconstruct meshes with both high fidelity and correct topology. Hence, unlike previous hybrid techniques which first obtain a coarse mesh from a voxel predictor and later refine that mesh to get details, GAM dis-

entangles geometry from topology, by making the point network solely responsible for geometry and the mesh prior responsible for topology. Further, by training point networks with GAM, we optimize the vertex positions to generate adaptive meshes with arbitrary topology.

4.3 Guided and Augmented Meshing

4.3.1 Overview

Our goal is to compute a surface for the output points from a point network using a mesh prior which coarsely resembles the ground truth shape and has correct topology. Depending on the application, such mesh priors can either be obtained using other reconstruction methods (e.g implicit methods) or are already present (e.g ground truth meshes). We pro-

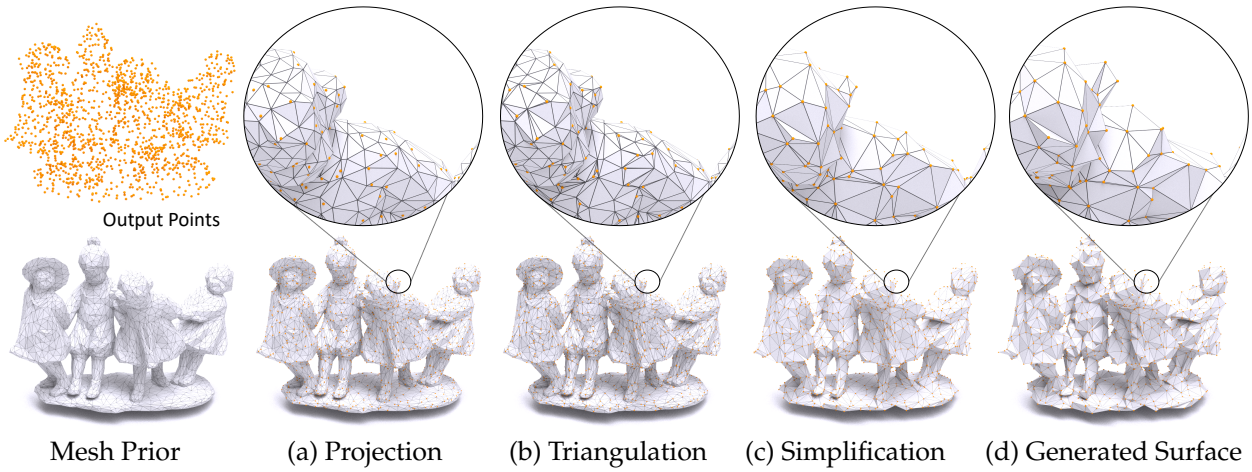


Figure 4.2: Surface reconstruction using GAM. Our method uses both the output points (orange) from a point network and a mesh prior to generate a surface for the output points. For the sake of illustration, here we use the ground truth mesh as priors for GAM. (a) Projection of output points on the mesh surface is followed by (b) Triangulation to ensure that they are incorporated into the mesh prior. (c) Simplification via sequential edge collapse removes all the original points. (d) Unprojection of the points recovers the final surface for the output point set. Please zoom in for details.

pose to use these mesh priors to guide the reconstruction of an output mesh with the output points. Such a reconstruction is more accurate in terms of topology and geometry than an off-the-shelf surface reconstruction algorithm like BPA or SPR. If the output points from a point network are exactly the same as the vertices of the ground truth mesh, we want the reconstructed mesh also to be exactly the same as the ground truth mesh. Figure 4.1d shows such a case where using the input vertices we reconstruct the ground truth mesh, whereas other methods do not. For any other output point sets, we would like the reconstructed mesh to best preserve the input geometric features and topology. To this end, we propose a new meshing algorithm called *guided and augmented meshing*, GAM, for which the input is a mesh prior and the reconstructed points from a point network and the output is a triangular mesh connecting those reconstructed points. GAM works with both manifold and non-manifold mesh priors, and with both dense and sparse reconstructed points. Furthermore, it does not require the reconstructed points to have the same cardinality as the mesh prior. These properties make GAM suitable to be used with a wide variety of 3D deep learning point networks.

GAM has two main algorithmic steps. First, the mesh prior is *augmented* by projecting the reconstructed points onto the mesh prior and retriangulating to include these projected points in the mesh prior. This ensures that the topology of the mesh prior is carried over to GAM's output. Second, all the original points in the augmented mesh are removed by collapsing them to the nearest projected points, resulting in a mesh with only the projected points as vertices and with the connectivity that is *guided* by the mesh prior. Finally, the projected points are "unprojected", yet retaining their mesh connectivity to get the final meshing of the reconstructed points as shown in Figure 4.2. We now discuss these two steps in detail.

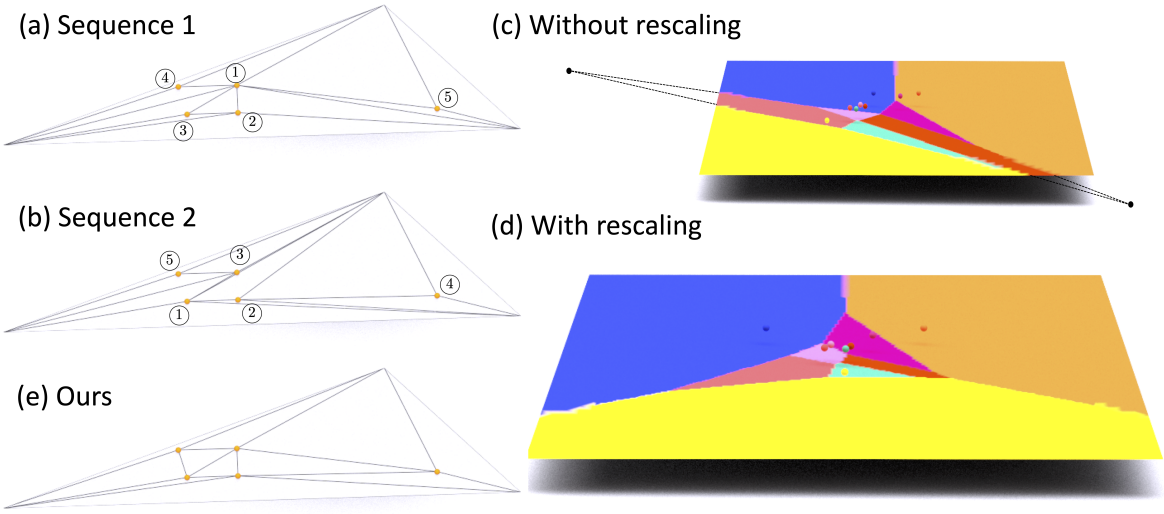


Figure 4.3: Projection. (a-b) Sequential processing of points leads to two different triangulations for the same set of projected points. (c) For a triangle, we map all the projected points and its base vertices to a dense 2D grid. To prevent Voronoi vertices (black points) from lying outside the grid, (d) we rescale the points to an equilateral triangle. (e) We then compute the Voronoi diagram and use it to generate Delaunay triangulation for the projected points.

4.3.2 Projection

GAM induces the topology of the mesh prior into the output mesh by first projecting the reconstructed points onto the mesh prior and retriangulating the mesh. Hence, unlike other surface reconstruction algorithms that starts with no connectivity, GAM starts with the connectivity of the mesh prior and finds the final output connectivity by removing unwanted edges.

Once the points are projected onto the mesh prior, they have to be connected to existing points in the mesh to create a valid triangulation. A valid triangulation could be achieved through sequential projection of each point where if the point projects - (a) inside a triangle, 1-3 split of the triangle is performed, (b) on an edge, 1-2 split is performed on each incident triangle and (c) on an existing input mesh vertex, it is replaced with the projected point. However, the connectivity and quality (sliver triangles) of the resulting triangulation depends on the order of the projected points as shown in Figure 4.3(a-b). We propose

an order-independent triangulation to process all the projected points inside a triangle in parallel using a novel interpretation of Delaunay triangulation. Our method is deterministic and provides consistently high quality triangulation. Further, it is fast and can easily be incorporated inside point networks (section 4.6) unlike other triangulation techniques [6, 71].

Given the vertices of a base triangle and the projected points which lie inside the triangle, we first map all these points to a dense 2D grid. We then compute for each grid point the closest mapped point. We observe that a small neighborhood of grid points that is closest to three different mapped points contains a Voronoi vertex, which in turn indicates a Delaunay triangle connecting the corresponding three closest mapped points. To prevent Voronoi vertices from lying outside the 2D grid, we rescale the base triangle to an equilateral triangle and map all the projected points to their corresponding barycentric coordinates before computing the above mapping as shown in Figure 4.3(c-d). In order to ensure that the triangulation is computed fast, we perform the following optimizations - a) All points that project on the edges of the base triangle are perturbed slightly to fall inside either of the neighbouring triangles. This allows us to process all the triangles independently and in parallel. b) We only process triangles which contain at least one projected point. c) Triangles with only one projected point are directly split into 3 triangles. d) Triangles with more than one projected point are processed using the above projection-to-grid method.

In all cases, the projection operation refers to finding the closest point on the mesh prior, and moving the reconstructed point to that closest point on the mesh. This definition of projection gracefully handles projecting points on a mesh with boundaries as well. Finally, if the input mesh is a non-manifold mesh, all the above operations and definitions work without any modification.

4.3.3 Simplification

Using the mesh generated from the previous step, we next perform simplification where we remove all the original vertices (from mesh prior) through edge collapse operations so that the final mesh contains only the projected vertices. As the order of the edge-collapses determine the final connectivity of the mesh, we sequence the edge collapses based on a cost function.

In order to simplify the mesh we first label all the projected points as 0 and the original points as 1 and select all the edges between the original points and the original and the projected points. Any edge between two projected points is not collapsed. The final point after the collapse of an edge between (a) two original points is, for the sake of simplicity, set to be the mid point of that edge and that mid point is again labelled as original point, and (b) an original and a projected point is set to be the projected point since we cannot move the projected point. Based on these final positions of the points the cost for edge collapse operations is computed as:

$$\Delta(v) = e^{(l_1+l_2)}||v_1 - v_2||_2^2 \tag{4.1}$$

where v_1 and v_2 are the vertices forming an edge and l_1 and l_2 are their corresponding labels. Edges are placed in a priority heap with minimum cost edge on top, and are iteratively removed from the top and collapsed, until the heap is empty. After each edge collapse, the cost to collapse the new edges are computed and they are added to the heap if they are not between two projected points.

Although this works well in practice, there could be few edge collapses that cause triangle flips. In such cases, that edge collapse is skipped, edge removed from the heap, and revisited again after other edge collapses. Note, that an edge collapse that introduces

triangle flips may become a valid edge collapse after other neighborhood edge collapses. However, at the end, all original vertices are removed through edge collapses irrespective of whether it creates flipped triangles or not. Although quadric error [25] could be used as a cost function, we empirically found our weighted edge length cost (Equation 4.1), which favors short edges, reduces the number of triangle flips. Finally, the projected points are unprojected back to their original reconstructed point positions to get a final mesh (Figure 4.2d).

4.4 Analyzing GAM

Before evaluating the performance of GAM on various applications, we first analyze the effects of various components on GAM. Specifically, how does GAM preserve topology of the mesh prior in the output surface (section 4.4.1). As the input to GAM is a mesh prior and output points, we also study the effects of both of these inputs on the output reconstructed surface in sections 4.4.2 & 4.4.3 respectively. And lastly, we also give theoretical bounds on the mesh prior for an accurate surface reconstruction by GAM in section 4.4.4.

4.4.1 How does GAM preserve Geometry & Topology

GAM preserves the topology of the mesh prior as all operations such as local triangulation and edge collapse are topology preserving operations. Under extreme simplification, certain edges, if collapsed, may geometrically (visually) close genus of the model, although in implementation with multi-edge data-structure between vertices, the genus and rest of the topology can be maintained. With sufficient number of output points from the network such extreme simplifications and hence visual change of topology can be prevented. GAM also preserves the geometry as much as possible based on the output

Figure 4.4: Effects of Mesh Prior on GAM. Surfaces generated using the same points (orange) but with three mesh priors of different resolution (shown in inset). GAM creates geometrically similar but topologically different meshes.

quality of points networks. Since these reconstructed points are projected to the closest point in the mesh, GAM does its best to preserve the geometry by imposing the connectivity of the input mesh to the closest reconstructed points.

4.4.2 Effect of Mesh Prior on GAM

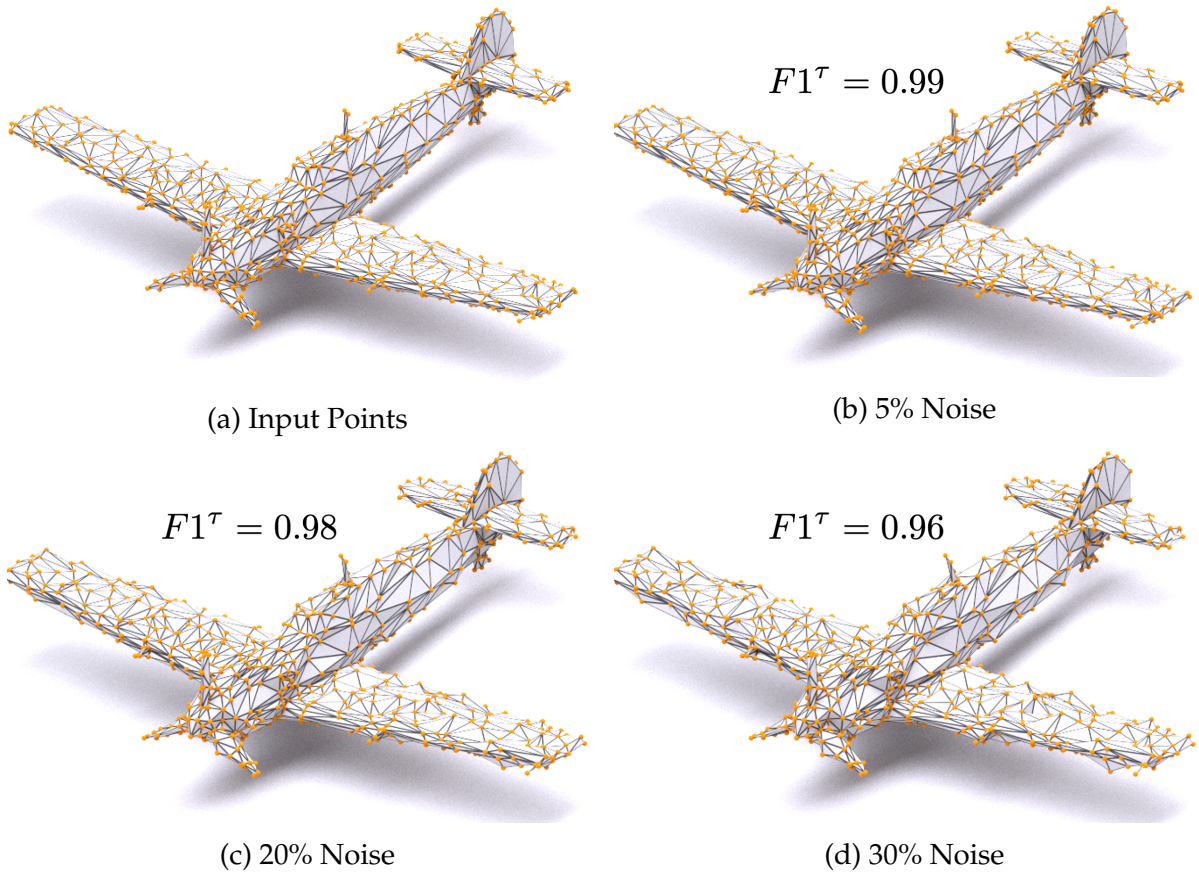
As GAM uses a mesh prior to generate a surface for the output points from a point network, a natural question to ask is what are the prerequisite of a mesh prior. Here we study the effects of the mesh prior on the final reconstructed surface.

We voxelize the ground truth shape at three different resolutions and use the same set of points to reconstruct surfaces using GAM. Specifically, we create three mesh priors whose number of octree leaf nodes are 100, 500 and 4000 respectively and use 2500 points which are sampled on the ground truth mesh. As evident from the Figure 4.4, using the same points, GAM reconstructs geometrically similar shapes whose topology is dictated by the

Figure 4.5: Non-Manifold Mesh Priors. GAM reconstruct accurate surfaces even with non-manifold mesh priors which have multiple connected components. The topology of mesh prior is carried over to the output mesh (see pink arrow).

topology of the mesh prior. This confirms that GAM does not need accurate mesh priors for surface generation. As long as the mesh priors have correct topology and coarsely resemble the original mesh, accurate surfaces will be generated. In our experience, such mesh priors, if not already present, can be easily obtained using existing reconstruction methods (volumetric/implicit networks). We, however, observe that surfaces generated using coarser meshes have different triangulation. This is due to the edge collapse operation during simplification where the cost to collapse an edge depends on the length of the edge. If we allow edge-flip as an operation during simplification, it is possible that we can generate consistent triangulation's independent of the template mesh resolution.

Non-Manifold Mesh Priors. We also show few surfaces generated from GAM using non-manifold mesh priors which contain multiple connected components (Figure 4.5). As GAM borrows the topology of the mesh prior, we observe that the output mesh is disconnected as the mesh prior itself is not a single watertight mesh.



(a) Input Points

(b) 5% Noise

(c) 20% Noise

(d) 30% Noise

Figure 4.6: Effects of Points on GAM. Surfaces generated from GAM when Gaussian noise is added to the mesh vertices. Using the same mesh prior (ground truth mesh), we preserve as much detail as maintained by the points.

4.4.3 Effect of Output Points on GAM

Next, we demonstrate that GAM always reconstructs meshes with correct topology while preserving as much details as maintained by the output points of the network. We show this by injecting noise in the vertices of a mesh and evaluating the surface reconstructed from GAM, which uses the original mesh as prior. Specifically, keeping the mesh prior fixed, we reconstruct surfaces with GAM by adding Gaussian noise in the normal direction with standard deviation 5%, 20% and 30% of the length of the bounding box diagonal to the vertices of the mesh.

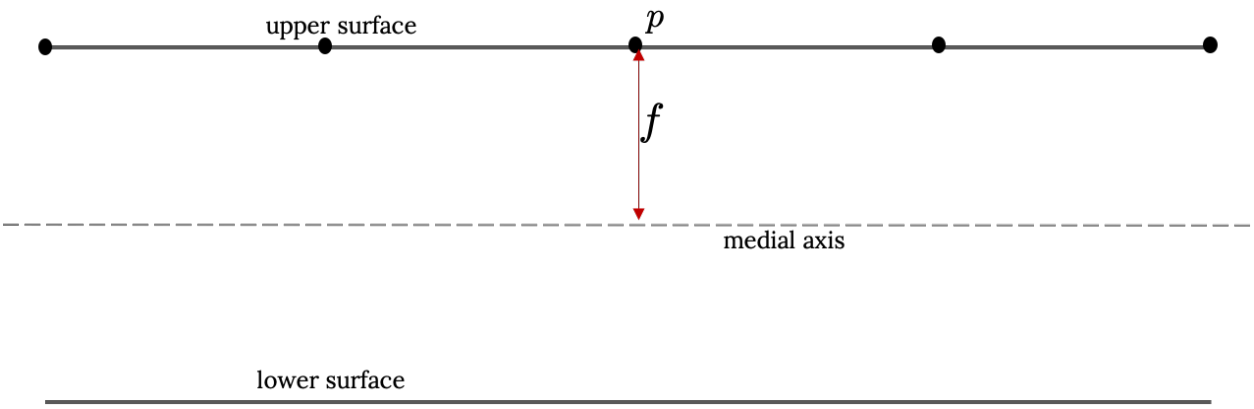


Figure 4.7: A point p sampled on the surface of a mesh is at a distance f from its medial axis.

From Figure 4.6 we observe that adding noise in the normal direction does not affect the output (i.e. triangulation) of GAM, as the output points are projected to the same point on the mesh prior. This suggests that the error in the final reconstruction (low F1 score) is due to the point location and *not* from GAM. We also observe that GAM reconstructs a smooth surface provided the points lie close to the surface of the mesh prior. However, as GAM guarantees to connect all output points, rough surfaces can be generated even if there is noise in only few points. This helps GAM differentiate subtle differences between two point clouds making it an ideal meshing algorithm for evaluating point reconstruction networks (section 4.7.1).

4.4.4 Bounds on the Mesh Prior

While in section 4.4.2 we studied the effects of mesh prior on GAM, here we provide some theoretical bounds on the mesh prior for generating an accurate surface with correct topology through GAM. We first define what is a feature size. We then consider two cases and give theoretical bounds on the mesh prior for each of them. And finally, we conclude by providing a theorem along with a formal proof for the general case.

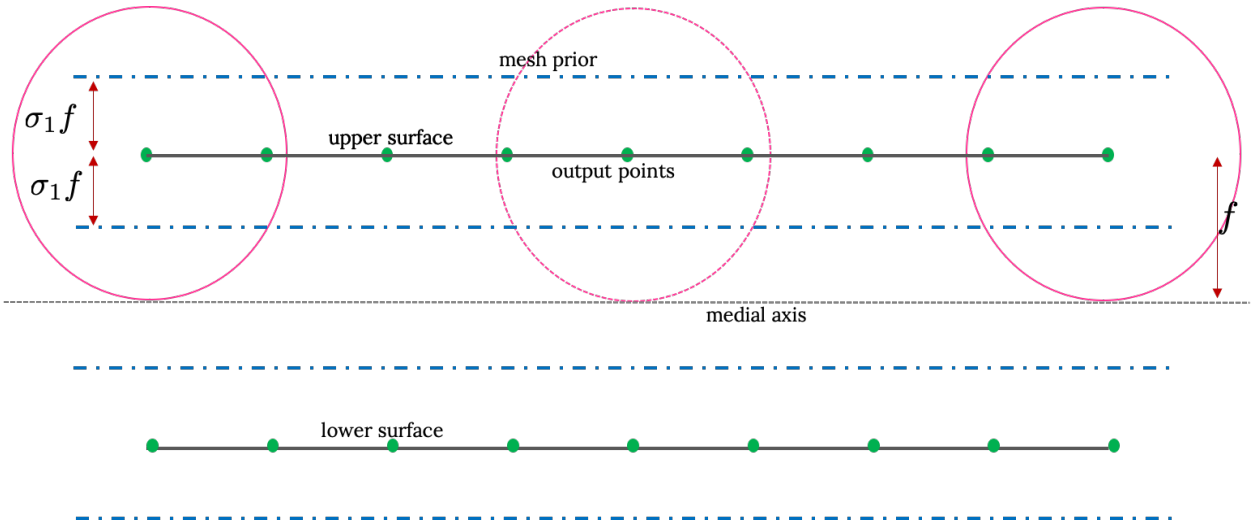


Figure 4.8: Accurate points. If the points from the point network (green) lie accurately on the surface of the ground truth shape, then the mesh prior (blue) needs to lie within the Minkowski (pink) sum of the feature size of all points on the original surface.

Feature Size. For reconstructing a surface, feature size f for a point p on the surface is defined as the Euclidean distance from p to the nearest point on the medial axis [4], which can be represented by the set of Voronoi vertices as shown in Figure 4.7.

Accurate Points. First, we consider the case when the output points from the point network are accurate i.e they lie on the surface of the ground truth shape. In order to prevent the points from projecting onto the lower surface of the mesh prior, the mesh prior needs to lie within the Minkowski sum of the feature size of all points on the original surface as shown in Figure 4.8 and Equation 4.2.

$$\begin{aligned} \sigma_1 f &< (f + f - \sigma_1 f) \\ \sigma_1 f &< (2f - \sigma_1 f) \\ \sigma_1 &< 1 \end{aligned} \tag{4.2}$$

Accurate Mesh Prior. Second, we consider the case when the output points from the point network have noise while the mesh prior is accurate i.e. the same as ground truth. Again,

in order to prevent the points from projecting onto the lower surface of the mesh prior, $\sigma_2 < 1$ as shown in Figure 4.9 and Equation 4.3.

$$\begin{aligned} \sigma_2 f &< (f + f - \sigma_2 f) \\ \sigma_2 f &< (2f - \sigma_2 f) \\ \sigma_2 &< 1 \end{aligned} \tag{4.3}$$

General Case. We now discuss the more general case where there could be error in both the mesh prior and the output points from the point network.

Lemma 1. *In order for GAM to reconstruct surfaces with correct topology, the sum of the error in the mesh prior (σ_1) and the error in the output points (σ_2) needs to be less than 1.*

Proof. In order to reconstruct a surface with correct topology, the upper output points need to be projected onto the upper mesh prior. From Figure 4.10 and Equation 4.4 we can conclude that $\sigma_1 + \sigma_2 < 1$. Hence, the upper bound on the mesh prior is when $\sigma_1 = 0$ i.e. the mesh prior is the same as the ground truth surface and the lower bound is $\sigma_1 < 1 - \sigma_2$.

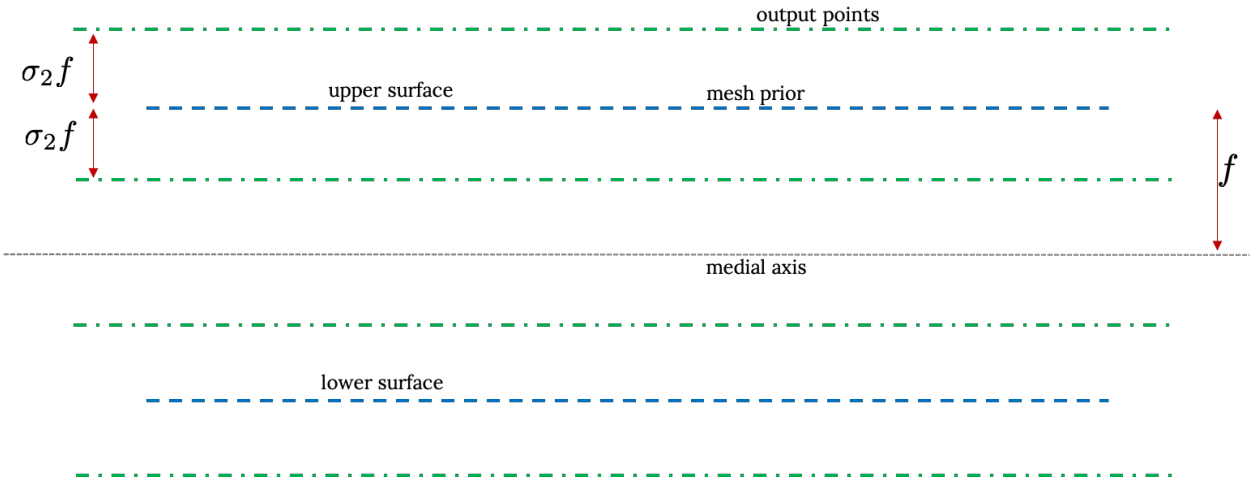


Figure 4.9: Accurate mesh prior. Using the ground truth mesh as the mesh prior (blue), the maximum error in the output points (green) can be σ_2.

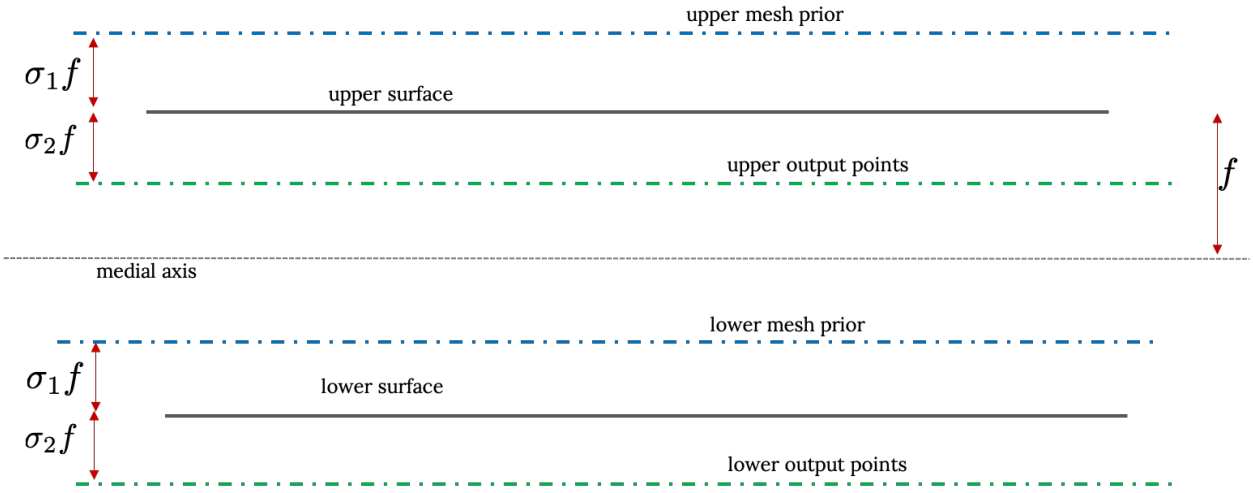


Figure 4.10: General Case where the error is in both the mesh prior (blue) and the output points of the point network (green).

However, for all practical purposes, $\sigma_1 + \sigma_2 < 0.5$ for reconstructing a surface with correct topology using GAM.

$$\begin{aligned} \sigma_1 f + \sigma_2 f &< (f - \sigma_2 f + f - \sigma_1 f) \\ (\sigma_1 + \sigma_2) f &< (2f - (\sigma_1 + \sigma_2) f) \\ 2(\sigma_1 + \sigma_2) f &< 2f \\ \sigma_1 + \sigma_2 &< 1 \end{aligned} \tag{4.4}$$

□

4.5 Single View Reconstruction

We first analyze the effectiveness of GAM for single-view 3D reconstruction where we use GAM in post-processing to combine the outputs of a point network and implicit networks and compare the resulting meshes with previous state-of-the-art methods.

Given a single RGB image of an object, the task is to reconstruct a 3D mesh with correct topology and accurate geometry. To this end we first train a point generation network called PSG$^+$ to output 2000 points. We then use meshes from implicit networks (OccNET [56] and IM-NET [14]) as priors for GAM to generate a surface for the output points.

4.5.1 Data

We use 3D models from 13 categories of ShapeNetCore.v1 [13] and renderings from 3DR2N2 [15] which together have been widely used for SVR [15, 28, 81]. We sample points on the surface of the meshes to train the point networks. We use the same train/test split as [15, 81] (34989 train & 8756 test meshes) and use renderings from all 24 viewpoints for a fair comparison.

4.5.2 Evaluation Metric

Since our goal is to asses the quality of the reconstructed mesh, similar to [28, 74], we sample 10k points uniformly at random from the surfaces of both the predicted and the ground-truth mesh and use it to compute Chamfer distance (CD) [22] and F1$^\tau$ score [45]. F1$^\tau$ score reports the harmonic mean of the precision and recall, which is computed as fraction of predicted or ground truth points within τ distance to points on the other surface [45]. We use the same evaluation protocol as [81] and use a 0.57 mesh scaling factor and $\tau = 10^{-4}$ on the squared Euclidean distance. Lower is better for CD and higher is better for F1 score.

4.5.3 Network & Implementation Details

We use a point network architecture where for the image encoder we use ResNet-18 [34] and for the point decoder 4 fully-connected layers of size 1024, 512, 256, (3x2000). We use ReLU non-linearity and batch normalization on the first three and tanh on the final layer. We train this network using Chamfer loss [22] for 30 epochs with 32 images per batch and Adam optimizer [44] at a learning rate 10^{-4} which decays by 0.8 every 10 epochs. After training, we use GAM as post-processing to combine the output points from our point network (PSG$^+$) and the meshes from implicit networks to generate the final predicted mesh.

4.5.4 Baselines

We compare our results with several state-of-the-art SVR methods. 3DR2N2 [15] and MVD [73] are voxel based methods which reconstruct meshes with arbitrary topology. Implicit networks such as OccNET [56] and IM-NET [14] also generate meshes with arbitrary topology but at high resolution. PSG [22] outputs point predictions. P2M [81] and GEOMetrics [74] deform a template to output a 3D mesh with fixed topology (zero genus). N3MR [41] also deforms a template with a fixed topology but uses a differentiable renderer to train without any 3D supervision. MeshRCNN [28] first predicts voxels with correct topology and then refines it to output a 3D mesh. We compare these methods not only on their mean CD and F1 scores on ShapeNet testset, but also on their topology and number of vertices in their output mesh. For MeshRCNN, we report the performance of their "pretty" model as it generates topologically correct meshes. For PSG, F1 scores were computed directly using the output points [81]. For OccNet and IM-NET, we use the models released by authors and simplify [25] the meshes to approximately 2000 vertices for a fair comparison.

Table 4.1: GAM vs Points/Implicit Networks. F1-scores on ShapeNet testset where we reconstruct meshes using GAM which combines geometry from a point network (PSG$^+$) and topology from implicit networks (OccNET or IM-NET). Such a combination performs better (shown in bold) than either the implicit network (columns 2 & 4) or the point network. Using the ground truth meshes as prior for GAM gives an upper bound for the same output points.

Category	$F1^{\tau}$						$F1^{2\tau}$					
	PSG$^+$	OccNET		IM-NET		GT Mesh	PSG$^+$	OccNET		IM-NET		GT Mesh
		[56]	w/ GAM	[14]	w/ GAM	w/ GAM		[56]	w/ GAM	[14]	w/ GAM	w/ GAM
Plane	82.83	72.24	**85.46**	81.90	**85.76**	88.01	91.70	82.27	**91.99**	89.41	**92.13**	93.68
Bench	64.07	67.67	**73.03**	**75.50**	74.07	76.45	81.86	80.93	**83.66**	**86.41**	84.55	86.06
Cabinet	44.83	67.54	**71.06**	67.54	**70.90**	72.99	70.91	79.90	**84.17**	80.88	**84.26**	86.37
Car	55.48	59.70	**76.02**	63.48	**76.41**	80.45	80.40	73.54	**88.23**	76.36	**88.51**	91.25
Chair	51.61	64.14	**68.29**	67.40	**69.02**	71.98	74.45	77.61	**81.36**	80.93	**82.04**	84.32
Monitor	45.16	58.89	**64.06**	60.75	**63.03**	67.49	70.11	73.11	**78.56**	74.72	**78.26**	81.04
Lamp	48.10	47.84	**56.34**	54.15	**57.19**	60.79	66.02	58.24	**68.45**	66.03	**69.20**	72.45
Speaker	36.67	50.86	**57.69**	56.33	**59.60**	61.50	61.79	63.80	**73.01**	70.41	**74.37**	76.04
Firearm	81.69	72.95	**82.35**	78.36	**83.02**	85.70	91.39	84.36	**89.74**	88.51	**90.28**	92.30
Couch	42.81	60.77	**64.98**	63.34	**66.16**	67.70	68.50	74.08	**79.38**	76.38	**80.10**	81.14
Table	60.92	**71.65**	70.09	**72.12**	69.60	72.08	80.26	**81.26**	82.18	**83.29**	81.84	84.68
Cellphone	63.98	76.78	**79.03**	78.16	**80.26**	81.34	84.49	86.62	**89.38**	87.23	**89.96**	89.97
Watercraft	55.13	46.84	**66.09**	60.93	**65.54**	70.14	76.09	60.70	**79.75**	74.82	**79.34**	83.17
Mean	56.41	62.91	**70.39**	67.68	**70.82**	73.59	76.77	75.11	**82.30**	79.65	**82.68**	84.80

Apart from comparing against previous methods, we also compare our results with few ablated versions of our method. Similar to PSG, we directly evaluate the output points of our point network and report it as PSG$^+$. Further, instead of GAM, we generate surfaces for the same output points using existing meshing methods like BPA and SPR. For SPR, for the normals of the output points, we use the normal vectors of their closest points in the ground truth mesh. For both BPA and SPR, we choose the parameters which gave us the best results and reconstruct the meshes using MeshLab [16].

4.5.5 Comparison with Point & Implicit Networks

Since GAM uses the output of both point and implicit networks, we first compare GAM with these two networks for single-view shape prediction. Table 4.1 shows that combining the geometry from a point network (PSG$^+$) and topology from implicit networks (OccNET/IM-NET) performs better than either the implicit or the point network alone.

Table 4.2: Quantitative Results for SVR. We compare several SVR methods by their output representation, $|V|$ (mean$_{\pm \text{std}}$), CD $(\text{x}10^3)$, F1 score and topology of the output mesh. For [81], † reports the results from their paper and ‡ using the model released by authors. We show that meshes reconstructed from GAM using IM-NET as priors achieves high fidelity and correct topology.

	Representation	$\lvert V\rvert$	CD (↓)	F1$^\tau$(↑)	Topology
N3MR [41]	Mesh	$642_{\pm 0}$	-	33.80	fixed
3DR2N2 [15]	Voxel	-	-	39.01	arbitrary
PSG [22]	Points	$1024_{\pm 0}$	-	48.58	-
P2M [81]†	Mesh	$2466_{\pm 0}$	-	59.72	fixed
OccNET [56]	Implicit Field	$1998_{\pm 8}$	0.825	62.91	arbitrary
MVD [73]	Voxel	-	-	66.39	arbitrary
GEOMetrics [74]	Mesh	$574_{\pm 99}$	-	67.37	fixed
IM-NET [14]	Implicit Field	$1999_{\pm 21}$	0.502	67.68	arbitrary
P2M [81]‡	Mesh	$2466_{\pm 0}$	0.444	68.94	fixed
MeshRCNN [28]	Mesh	$1896_{\pm 928}$	0.397	69.30	arbitrary
PSG$^+$	Points	$2000_{\pm 0}$	0.424	56.41	-
SPR	Mesh	$9069_{\pm 929}$	1.206	59.53	arbitrary
BPA	Mesh	$1871_{\pm 27}$	0.399	70.81	arbitrary
GAM w/ OccNET	Mesh	$2000_{\pm 0}$	0.415	70.39	arbitrary
GAM w/ IM-NET	Mesh	$2000_{\pm 0}$	0.387	**70.82**	arbitrary

Further, we get similar F1 scores using mesh priors from either OccNet or IM-NET. This confirms that GAM does not require accurate mesh priors as long as they coarsely resemble the ground truth shape in terms of both topology and geometry. We also reconstruct surfaces using the ground truth meshes as prior for GAM. This gives us an upper bound and indicates that the performance can further be improved with an implicit network that outputs a more accurate topology.

4.5.6 Comparison with Prior Works

We now compare our results with previous works. In Table 4.2 we show that by combining the geometry from our point network (PSG$^+$) and topology from IM-NET we outperform previous methods both in terms of CD and F1 score using similar number of vertices. Further, when comparing GAM against other meshing methods, we outperform

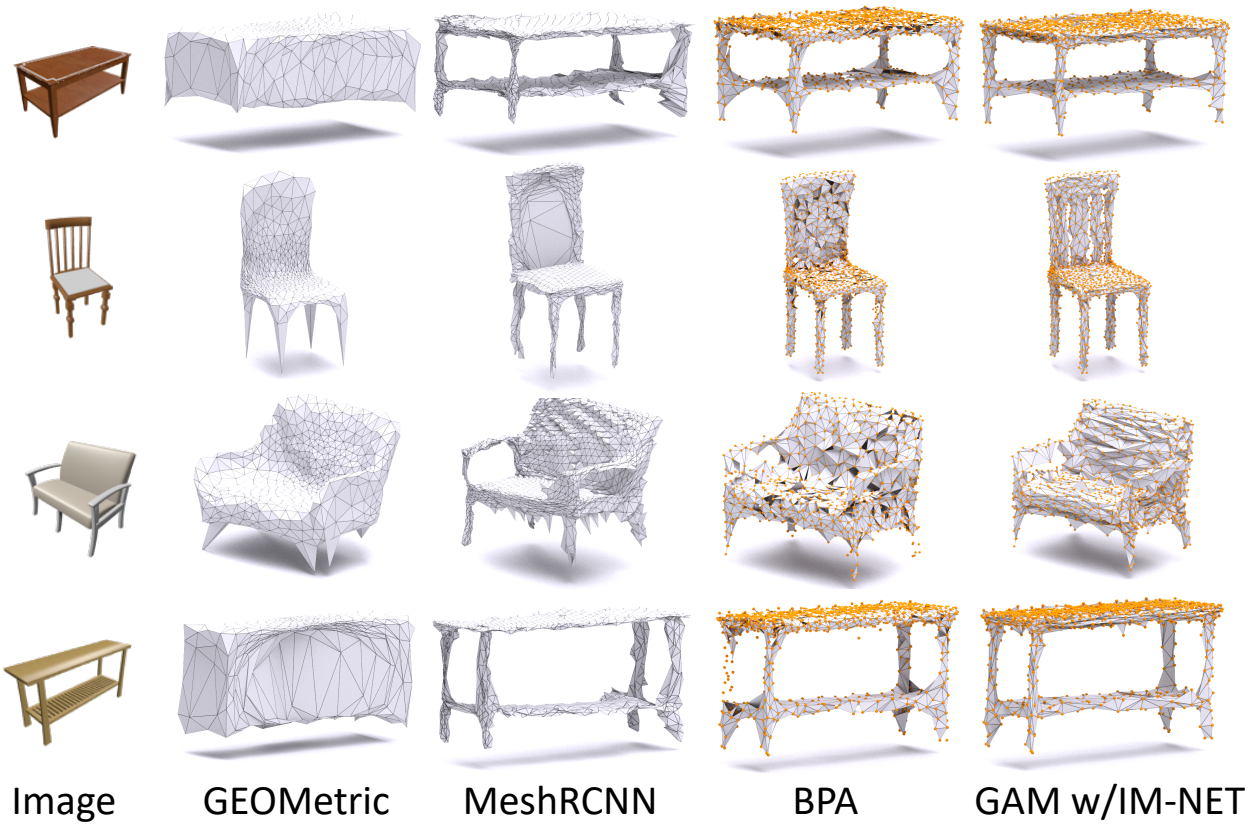


Figure 4.11: Single View Reconstruction Results. Using meshes from IM-NET as priors, GAM reconstructs surfaces for the output points (orange) of a point network (PSG$^+$). Unlike BPA, GAM guarantees to connect all the output points and reconstructs meshes with *both* accurate geometry and correct topology.

SPR by a wide margin but perform similar to BPA. BPA, however, cannot be used for surface reconstruction as it does not guarantee correct topology and often fails to connect all

Figure 4.12: SVR on Natural Images. Single-view reconstruction results on two images from Pix3D [76].

the points as shown in Figure 4.11 and Table 4.2. On the contrary, GAM generates meshes with correct topology and guarantees to connect all the output points.

4.5.7 More Qualitative Results

In Figure 4.13 and 4.14 we provide more qualitative comparisons of surfaces reconstructed from GAM when used in post-processing to combine the output points of our point net-

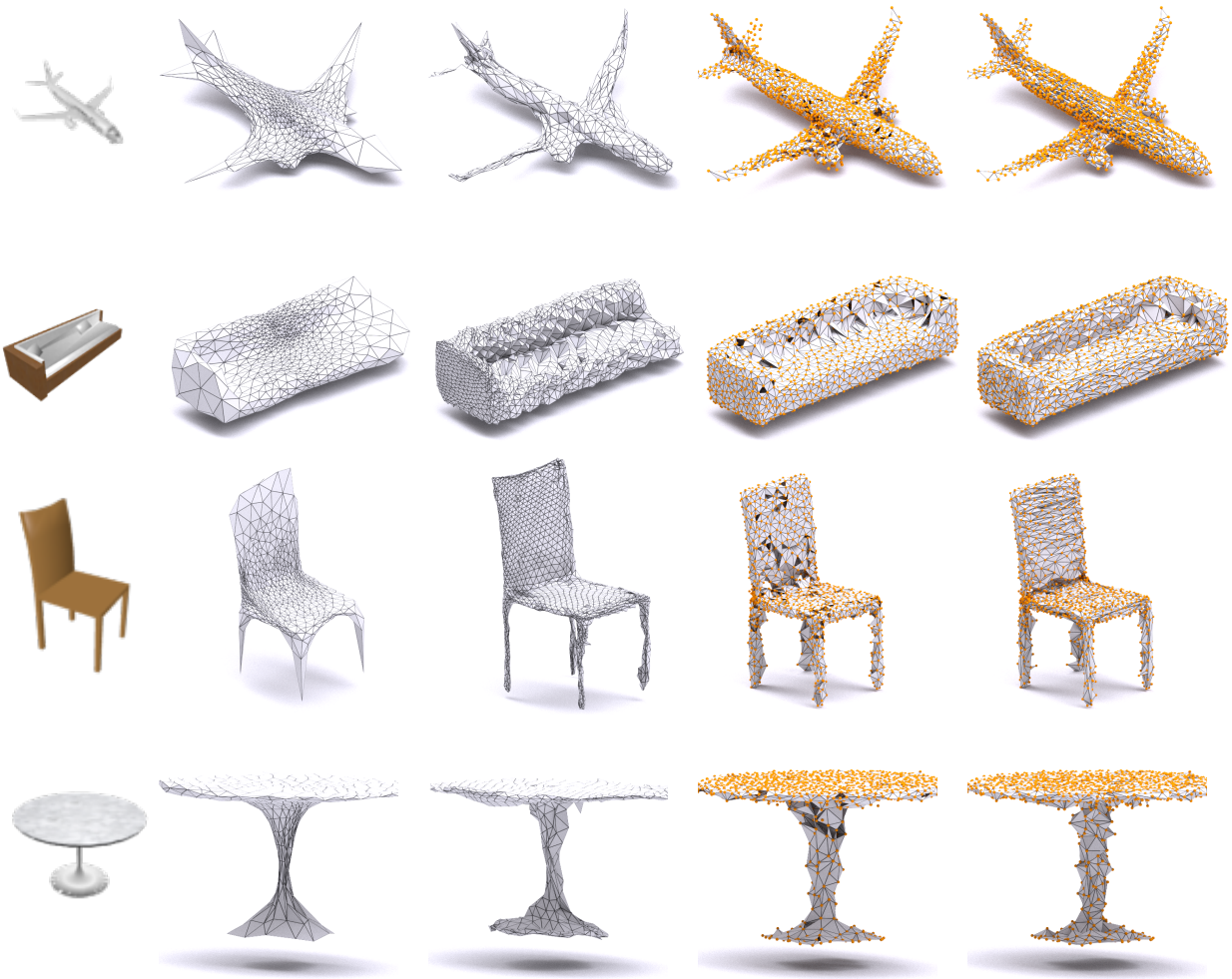


Figure 4.13: Single View Reconstruction. Qualitative comparison of meshes from various SVR approaches. Using meshes from IM-NET as priors, GAM reconstructs accurate surfaces for the output points (orange) of PSG$^+$.

Figure 4.14: More Qualitative results for Single-View Reconstruction.

work (PSG$^+$) and meshes from IM-NET. Unlike GEOMetric which deforms a fixed template, we borrow the topology from IM-NET to reconstruct meshes with correct topology. Furthermore, using the output of point networks which are specifically trained for geometry, we reconstruct meshes with higher fidelity than MeshRCNN using similar number of points (see the wings/engine of the planes and the details around the wheels of the cars in Figure 4.13 & 4.14). Lastly, using BPA to generate a surface for the output points of the point network gives inconsistent results as it fails to connect all the output points.

4.5.8 Results on Natural Images

Having seen reconstruction results on synthetic images, its quite natural to ask how does GAM perform on natural images. We show results in Figure 4.12 using images from Pix3D [76] dataset, which consists of 10069 real-world images and 395 unique 3D models. Specifically, we use GAM to combine the mesh prior and the output points obtained from IM-NET and PSG+ respectively.

4.6 Training Point Networks with GAM

In the previous section we train the point network using Chamfer loss between the ground truth and the reconstructed *points* and used GAM in post processing to generate meshes with uniformly distributed points. Here we show that by incorporating GAM inside the point network, we can reconstruct adaptive meshes with arbitrary topology. Specifically, during training, we generate a surface for the output points using GAM and compute a mesh loss between the ground truth and the output *surface*. By backpropogating this loss through the point network, we regress on the output points such that the output surface is as close as possible to the ground truth surface. Note, GAM does not have any learn-

able parameters and is deterministic where for the same output points, it will generate the same mesh.

We demonstrate this by training two point networks to output 250 points for SVR. We train one network with Chamfer loss on the output points and the other with $\mathcal{L}_{mesh}$ (Equation 4.5) on the meshes reconstructed using GAM. We use implicit meshes as priors for GAM and train both networks on renderings from two categories of ShapeNet.

4.6.1 Data & Implementation Details

To demonstrate the advantages of training point networks with GAM, we train two networks using data from two categories (chair and couch) of ShapeNet [13] where we use one image per shape making a total of 7956 train and 1991 test images. Both networks have the same architecture where for the image encoder we use ResNet-18 [34] and for the point decoder 4 fully-connected layers of size 1024, 512, 256, (3x250). The only difference is that one network is trained with Chamfer loss and the other using a mesh loss on the surfaces generated from GAM. We use ReLU non-linearity and batch normalization on the first three and tanh on the final layer. We use data from two categories (chair and couch) of ShapeNet, specifically one image per shape making a total of 7956 train and 1991 test images. We train both networks with a batch size of 4 images and Adam optimizer with learning rate 10^{-4} for 100 epochs. After training we reconstruct the output meshes for both networks using GAM.

4.6.2 Mesh Loss

Since the point network now directly predicts a 3D surface, we need a loss function which computes an error between the predicted mesh $\mathcal{M}_2$ and the ground truth mesh $\mathcal{M}_1$. Simi-

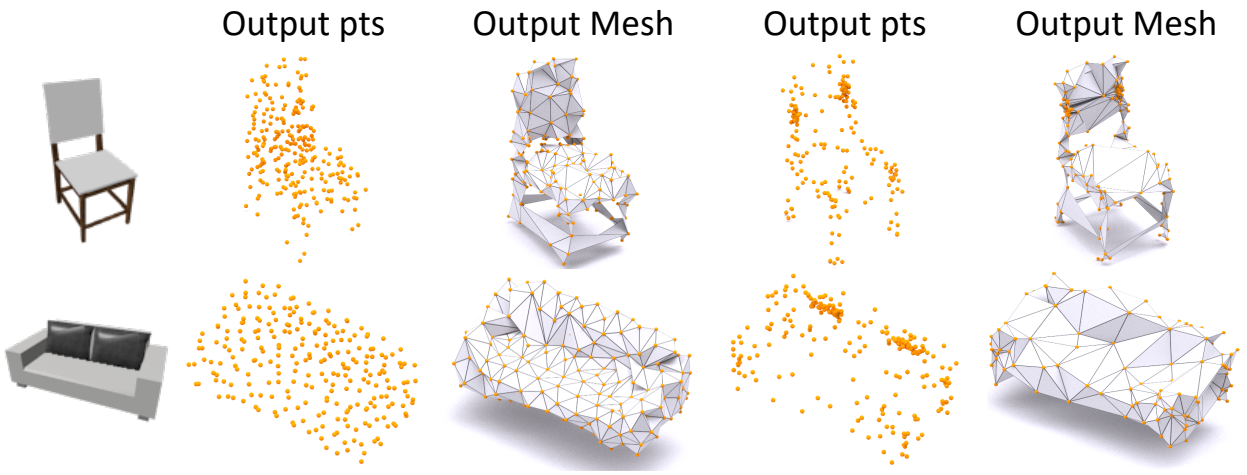
lar to [28, 62], although we can sample points on both meshes and define a point loss (like Chamfer loss) on these resampled points, this would result in the output points being uniformly distributed thereby not fully taking advantage of the mesh representation. We on the other hand only want the predicted surface to lie close to the ground truth surface and are indifferent to the output point distribution. Hence, we define the mesh loss as:

$$\mathcal{L}_{mesh}(\mathcal{M}_1, \mathcal{M}_2) = \sum_{p \in \mathcal{P}} \Phi(p, \hat{\mathcal{Q}}) \; + \; \sum_{q \in \mathcal{Q}} \Phi(q, \hat{\mathcal{P}}) \tag{4.5}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are the ground truth and output points, $\hat{\mathcal{P}}$ and $\hat{\mathcal{Q}}$ are the points sampled on the ground truth mesh $\mathcal{M}_1$ and the predicted mesh $\mathcal{M}_2$ respectively, and $\Phi(a, B) = \underset{b \in B}{min} \; ||a - b||_2^2$. Essentially, $\mathcal{L}_{mesh}$ minimizes both the distance between the ground truth points to the predicted surface for *high coverage* and the distance between the output points to the ground truth surface for *high accuracy*. We approximate both the meshes by densely sampling 10k points on their surfaces using a differentiable mesh sampling strategy [74]. This replaces the expensive point-surface distance computation with the fast point-point distance computation.

4.6.3 Results.

We compare the output meshes from both networks in Figure 4.15. We find training with GAM to perform slightly better in terms of mean F1 score. Further, while Chamfer loss uniformly distributes the points, $\mathcal{L}_{mesh}$ pushes the points towards the edges of the shape as GAM, being invariant to point distribution, can still reconstruct accurate surfaces. Clustering of points on the edges suggests that we could possibly reduce the number of points further, however this goes beyond the focus of this paper and is left as a potential future work.



(a) Image (b) Train with CD (c) Train with Mesh Loss

Figure 4.15: Training with GAM. Training a point network using mesh loss on surfaces reconstructed by GAM allows us to redistribute the points towards the edges of the shape, generating adaptive meshes. On the contrary, training the same network with Chamfer loss on output points generates meshes with uniformly distributed points.

Table 4.3: Quantitative comparison of training a point network with CD vs a mesh loss on surfaces generated from GAM.

$F1^{\tau}$	Train w/ CD	Train w/ $\mathcal{L}_{mesh}$
Chair	44.64	**48.17**
Couch	44.88	**45.33**
Mean	44.76	**46.75**

4.7 Other Applications

The properties of GAM together with point networks allow for several other applications.

4.7.1 Fair Evaluation of Point Networks

Most point based reconstruction networks [22, 52, 1, 2] have access to ground truth meshes, but still evaluate their network by comparing the *output points* with the ground truth points/surfaces. We show that evaluating the output points alone can be misleading and

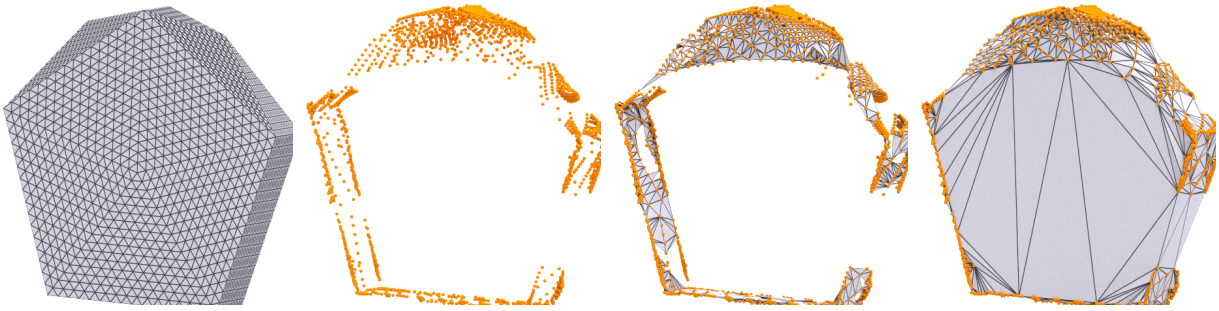
Table 4.4: Evaluating Point Networks for SVR. We compare four point networks with different number of output points by CD $(\text{x}10^3)$ on the output points, F1 scores on the surfaces reconstructed using BPA, SPR & GAM and the mean $\%$ of unreferenced vertices in these reconstructed surfaces. Analyzing the network's performance through the output points alone can often be misleading. Evaluating the surfaces reconstructed by GAM using the output points gives a more accurate assessment.

		$\lvert V\rvert$				$\Delta(\downarrow)$
		2500	2000	1000	500	
Chair	CD (↓)	4.60	5.01	6.76	9.13	98.47%
	$F1^\tau$ (BPA) (↑)	$68.77_{5.9\%}$	$69.08_{5.8\%}$	$65.63_{5.4\%}$	$57.39_{5.4\%}$	19.82%
	$F1^\tau$ (SPR) (↑)	$56.64_{99.9\%}$	$55.30_{99.9\%}$	$50.39_{100\%}$	$44.64_{99.9\%}$	26.88%
	$F1^\tau$ (GAM) (↑)	$\mathbf{72.74}_{0\%}$	$\mathbf{73.32}_{0\%}$	$\mathbf{70.95}_{0\%}$	$\mathbf{69.25}_{0\%}$	**4.79**%
Plane	CD (↓)	2.23	2.36	3.26	4.61	106.7%
	$F1^\tau$ (BPA) (↑)	$86.07_{6.2\%}$	$85.86_{6.2\%}$	$84.84_{5.1\%}$	$83.43_{4.0\%}$	3.06%
	$F1^\tau$ (SPR) (↑)	$68.92_{99.9\%}$	$65.82_{99.9\%}$	$61.77_{99.9\%}$	$53.33_{100\%}$	22.62%
	$F1^\tau$ (GAM) (↑)	$\mathbf{88.59}_{0\%}$	$\mathbf{88.78}_{0\%}$	$\mathbf{87.89}_{0\%}$	$\mathbf{86.25}_{0\%}$	**2.64**%

often wrongly implicate the network to have low performance. Evaluating the *surfaces* reconstructed using the output points with the ground truth meshes (as priors for GAM) provides a more accurate assessment for the point network's performance.

To demonstrate this we train four point networks with the same backbone architecture but different number of output points (2500, 2000, 1000, 500) for SVR. We train these networks for the same amount of time on two categories (chair and plane) of ShapeNet. After training we not only report the CD on the output points, but also evaluate the surfaces (using F1 score) reconstructed using BPA, SPR and GAM using the ground truth meshes as priors. We also measure the error from networks with 2500 and 500 output points and report it as Δ in Table 4.4.

Results We draw the following conclusions from Table 4.4. (a) Networks trained using low number of output points have higher CD ($\Delta\approx$100%) incorrectly suggesting that those networks are weak and their output points are not reconstructed properly. However, analyzing their F1 scores suggests that all surfaces are similar (small Δ) and all networks perform equally well. This shows that we cannot solely evaluate the output points to as-



(a) Original Mesh (b) Output from [2] (c) BPA (d) GAM

Figure 4.16: Meshing Sparse Point Clouds. GAM can be used to generate surfaces for point networks which output sparse point clouds.

sess the network's performance and should also evaluate the meshes reconstructed using those output points. (b) Meshes reconstructed using BPA and SPR on average fail to include 5.6% and 99% of the output points respectively. Hence, they should not be used to mesh the points from a point network. Both BPA and SPR are highly sensitive to the input point distribution and require dense and uniform points for an accurate reconstruction. (c) For all the four point networks, meshes reconstructed using GAM have a higher F1 score than BPA and SPR. This suggests that the points from all networks capture accurate geometry but the meshing algorithms (BPA and SPR) reconstructed poor surfaces. While BPA cannot guarantee correct topology as it fails to connect all output points, SPR cannot guarantee correct geometry as it interpolates the output points using additional points (Fig. 4.1). GAM is independent of both point density & distribution and always includes only and all the output points of the point network in the reconstructed mesh. It guarantees correct topology & geometry and hence can be used in post-processing to decouple the reconstruction error from the network error. Please see the supplementary for tests on the robustness of GAM.

4.7.2 Reconstruction Surface for Sparse Point Clouds

As GAM is indifferent to both point density & distribution, it can be used with various point networks which output sparse point clouds [84, 54, 2]. As opposed to BPA or SPR, GAM guarantees to reconstruct an accurate surface connecting all the output points using the ground truth meshes as priors. Such meshes can then be used for qualitative and/or quantitative analysis of the network (Fig. 4.16).

4.8 Limitation & Discussion

In this paper, we introduce GAM, a new meshing algorithm to generate a surface for the output points of a point network using a mesh prior. GAM decouples geometry from topology by making the point network solely responsible for geometry and the mesh prior responsible for topology. We show the benefits of such a disentanglement for single-view shape prediction and fair evaluation of point networks. Further, unlike traditional surface reconstruction algorithms, GAM is independent of the density and distribution of the output points and guarantees to reconstruct a surface with correct topology and geometry.

As GAM aims to preserve the geometry from point networks, the resulting meshes are often less smooth than implicit methods. Using GAM to generate adaptive but smooth meshes could be interesting future direction. GAM also requires a mesh prior which is aligned and coarsely resembles the ground truth shape. Although this may seem as a strong assumption, in our experience such mesh priors, if not already present, can be obtained from other reconstruction methods. Therefore, we believe GAM is an attractive meshing algorithm for deep point networks.

Chapter 5

AntHQ : A High Quality 3D Mesh Dataset with Artistic Embellishments.

In this chapter, we first motivate the need for a high quality 3D mesh dataset. We then describe various efforts put in data collection, data processing and designing the web interface. Finally, we conclude by discussing a few promising future applications facilitated by such a high quality 3D dataset.

5.1 Motivation

Recent technologies and algorithms have led to an explosion in the amount of 3D data that we can generate and store. Repositories of 3D CAD models are expanding continuously, predominantly through aggregation of 3D content on the web. 3D range scanners (e.g LiDAR, Kinect), RGB-D sensors and other technology for scanning and reconstruction are providing geometric representation of objects that can eventually become CAD-quality models.

Some of the most prominent 3D datasets include ShapeNet [13], ModelNet [88], ABC [46]. While these datasets have had a major impact on developing learning based algorithms for tasks such as object detection, segmentation, correspondence compution, single-view 3D reconstruction, normal estimation, to name a few, most of the models in these datasets are synthetic (CAD models) and lack details as shown in the inset figure.

In order to fill this gap, we constructed AntHQ (pronounced as antique), a dataset of realistic 3D models containing detials and artistic embellishments.

In constructing AntHQ, we aim to fulfill three primarily goals:

Figure 5.1: AntHQ Dataset. A sample representation of models from our AntHQ dataset.

1. Open up new research directions such as fine grained object classification, high fidelity 3D reconstruction etc.

2. Extend current learning based geometric algorithms to a richer and more realistic class of datasets.

3. Inspire the community to work towards collecting and growing similar high fidelity datasets.

5.2 AntHQ Dataset

AntHQ is a dataset comprising of approximately 130 3D models from each of the five object categories - chairs, tables, mirrors, doors and closets (Figure 5.1). Although in comparison to other shape repositories, "AntHQ" dataset contains only 630 models, we hope such an effort is the first of many in collecting such high fidelity models.

In the following sections, we describe how 3D models are collected for AntHQ, the data processing pipeline, the web-interface for sharing the dataset with the community and finally discuss several promising future applications using this dataset.

5.2.1 Data Collection

To build this dataset, we first complied a list of object categories which frequently contain rich decorations and embellishments. From this list we selected five categories which are also common in other datasets - chairs, tables, mirrors, doors and closets. We then collected for each of these categories nearly 250 models from various publically available online repositories such as Cadnav [1], Yobi3D [2], GradCAD [3], Free3D [4], etc. A subset of these models which contained high quality decorations and styles were manually selected and processed.

5.2.2 Data Processing

In order to carry out any meaningful data processing, we first needed to clean the raw data. This comprised of removing unwanted objects from the 3D models such as floors and walls, removing any unreferenced vertices, duplicate vertices, zeros area faces in the mesh. We also ensured that all the meshes are triangulated, i.e. any quad mesh is converted into a triangular mesh. These preprocessing steps were important to ensure a smooth data processing pipeline.

We started the processing by first aligning all the models to a canonical space using the [5]. Models which failed were manually aligned. A lot of learning based geometric algorithms require the input data to contain the same number of points. Although one can sample equal number of points on the mesh. However, it is difficult to ensure that those sample points represent the details in the 3D models. Hence, next we simplified all the models to contain the same number of vertices. Simplification usually decreases the quality of

[1] https://www.cadnav.com/
[2] https://yobi3d.com/
[3] https://grabcad.com/
[4] https://free3d.com/

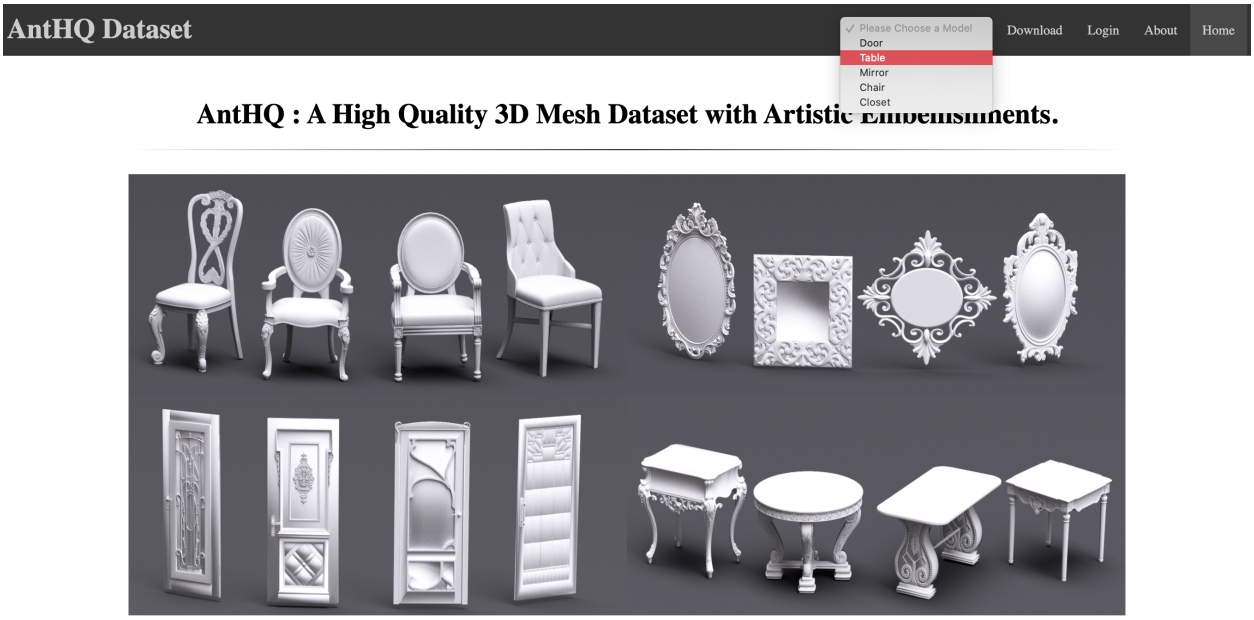


Figure 5.2: Screenshot of our web-interface where a user is selecting an object category.

the 3D models. In order to preserve the details yet reduce the number of vertices, we simplified all the models to 8000 vertices using quadric simplification [25]. All models were subsequently checked for detail preservation. We plan to release both the high-resolution and simplified meshes to the community in ply format for download.

5.2.3 Web Interface

Along with the dataset, we also provide a web interface for convenient access to all of the models within AntHQ. Our interface can be easily navigated by researchers, where they can select the category of interest and browse through all the models in that specific category as shown in Figure 5.2 and Figure 5.3. Further, researchers can view both the high resolution and the simplified instance of the model along with their individual statistics as shown in Figure 5.4. In addition, to make the dataset conveniently accessible to researchers, we provide a single link to download all the models.

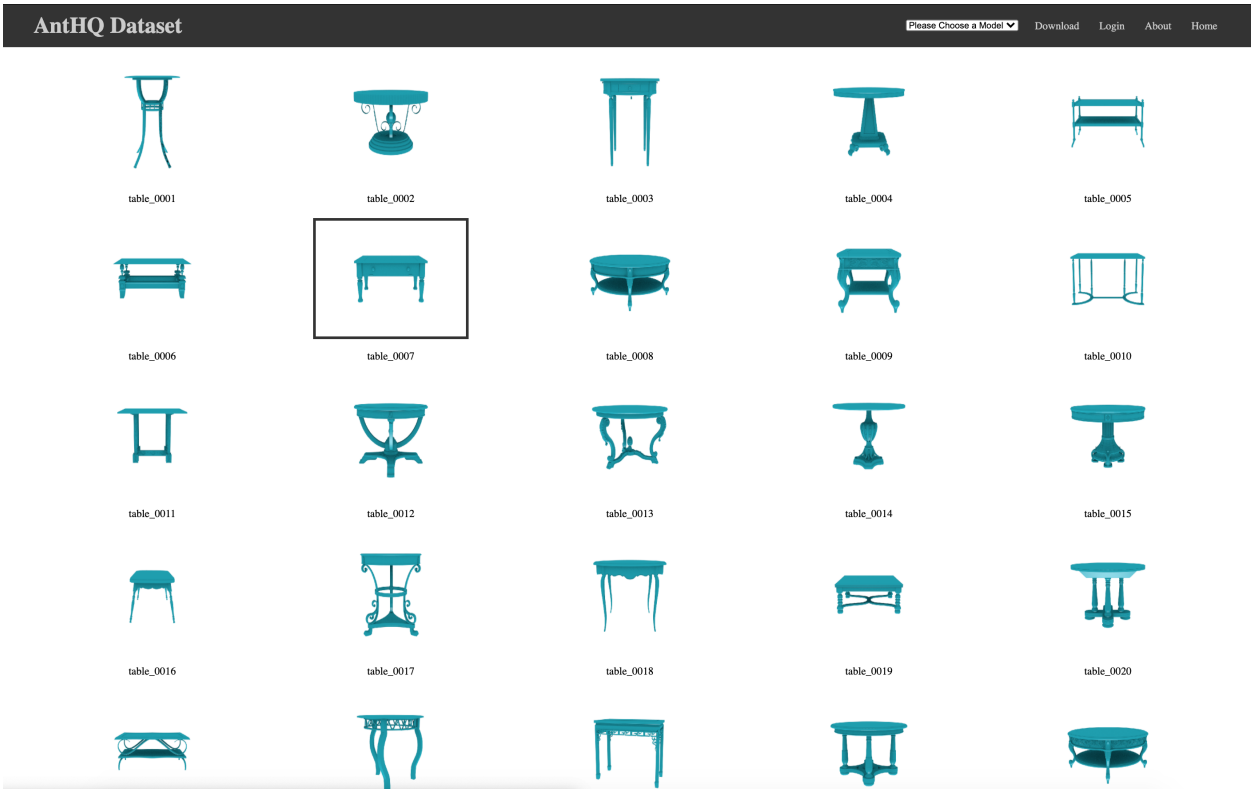


Figure 5.3: Screenshot of our web-interface displaying the models in the table category.

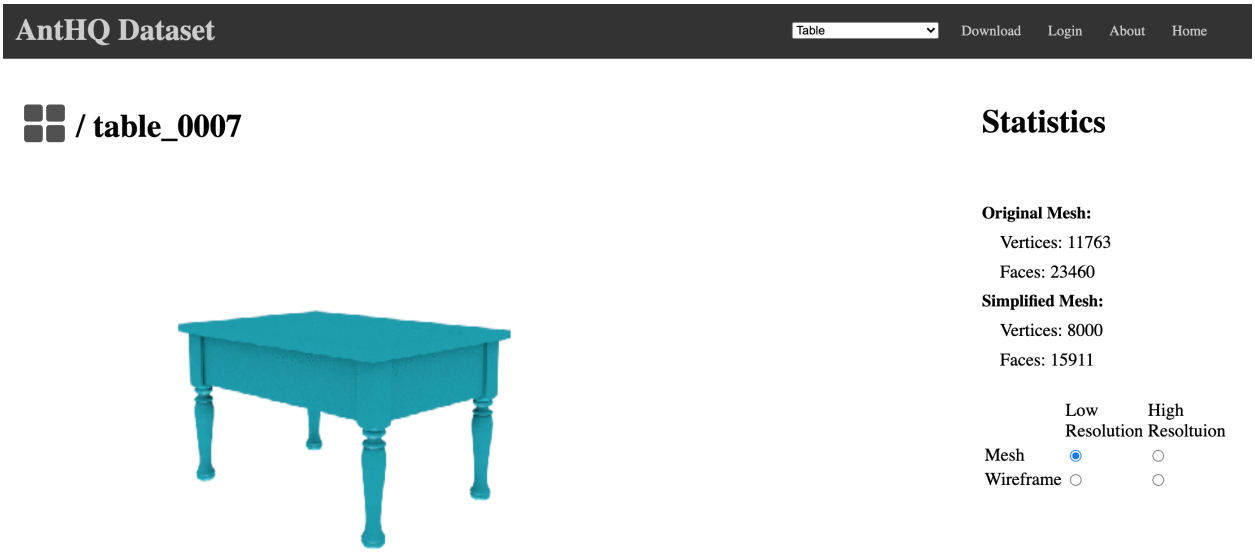


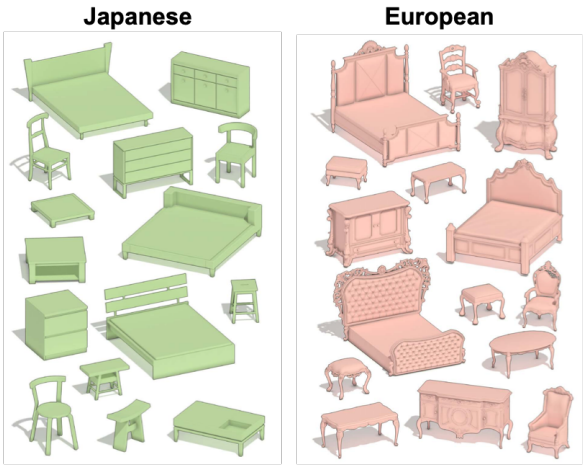
Figure 5.4: Screenshot of our web-interface displaying the statistics of a single table model.

5.3 Future Applications

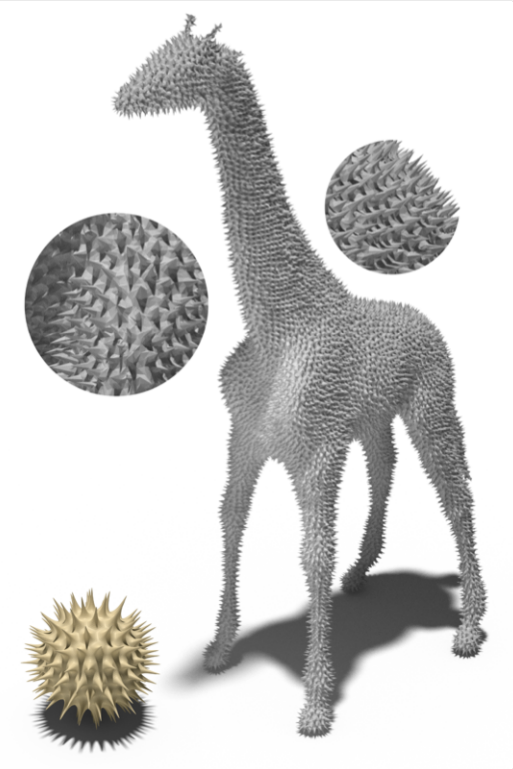
We now discuss few promising applications facilitated by our AntHQ dataset.

Understanding the Style of 3D object. It is well known that fine grained details can provide information about the style of object. For example, in the paper 'what makes Paris look like Paris?' [21], the authors show that small patches in images can not only capture the "look and feel" of the image but also inform us about its geographical location. In the image on the right, neoclassical columned entryway sporting a balcony, a Victorian window, and, of course, the cast iron railing are all very indicative of London. Translating these ideas to 3D, capturing and extracting fine grained details of 3D models can help us better understand the style of 3D models.

Fine Grained Object Classification. Fine grain classification is another topic which is well studied in the computer vision literature. Neural networks can successfully perform fine grain classification in images on specific object categories such as birds, flowers due to the presence of datasets with fine grained labels [86, 60]. However, extending this to 3D is a non-trivial task. One can possibly record



fine grained object labels during 3D model collection. However, such labels are often missing in online repositories. AntHQ dataset will allow researchers to design learning based geometric algorithms which can accurately capture fine grained details in feature embeddings. Such embeddings can then be used to cluster the models in the dataset. On the right, we show a sample image where authors used handcrafted geometric algorithms to cluster 3D models by their style [36].



Detail Transfer. Collecting 3D models with details can also facilitate in detail transfer. For example, using such a high resolution dataset, we can design deep neural networks which can automatically transfer the details of one 3D objects (e.g legs of one chair) to another 3D object of same or different category. In a recent paper [35], similar idea was proposed however the authors focused mostly on isotropic details rather than local details. Nonetheless, such techniques can help create new 3D models and impact e-commerce websites such as Amazon and IKEA. They also have applications in the real estate industry where users can automatically transform their room from one style (e.g. European) to another style (e.g. Chinese).

Chapter 6

Conclusion

In this thesis, we use insights from classical geometry processing algorithms to make two contributions for improving the performance of deep point networks. First, we present a point-surface loss function, named quadric loss, which penalizes the displacement of points in the normal direction thereby preserving sharp features in the reconstructed models. Quadric loss is differentiable and has nice geometric properties making it attractive to most point and mesh based networks. However, as quadric loss is an ellipsoidal loss, it cannot preserve the input point distribution. For points on planar regions, the reconstructed points may lie outside the surface. And hence, quadric loss should be accompanied by a spherical loss such as Chamfer loss, which preserves the input point distribution. Second, we present GAM, a surface reconstruction algorithm to mesh the output points of a point network using a mesh prior. GAM guarantees to generate a surface connecting all the output points but with the topology of the mesh prior. We show the advantages of GAM for single-view shape prediction. Further, we demonstrate that by training point networks with GAM, we can directly optimize the output vertex positions to generate adaptive meshes. Another appealing property of GAM is its invariance to point density and distribution. We demonstrate its advantages for fair evaluation of point networks and generating surfaces for networks which output sparse point clouds.

Both the approaches studied in this thesis leave considerable room for future research. For example, a requirement for computing quadric loss is that there needs to be an input triangulation available to compute the input quadric matrices. In would be interesting to analyze the effects of approximating the input triangulation with k nearest neighbours, which could be directly computed on point clouds. This will increase the generalizability of quadric loss to raw point clouds. Another direction for future research is to see the effects of bidirectional quadric loss, where we also compute the quadric loss between the input points and the quadric matrices computed on the output surface.

For surface reconstruction using output points of a point network, we currently need a mesh prior, which closely resembles the ground truth shape and has correct topology. It would be interesting to explore other alternatives from where we could get the topological information, such as RGB-D images or signed distance functions. Another limitation of GAM is its speed, especially for training point networks. The main bottleneck for the speed is simplification, where we currently process the edges sequentially. A promising future direction will be to explore various optimizations during simplification, such as batch processing of triangles, which do not have any projected points. This would substantially decrease the processing time and make training point networks with GAM much easier.

And lastly, I hope the dataset presented in this thesis "AntHQ" is the first of many high-quality datasets opening the doors to exiting to new applications, especially in augmented and virtual reality, 3D modeling and simulation. I genuinely believe such models will bring us one step closer to better understanding 3D shapes and in turn 3D scenes.

Bibliography

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018.

[2] N. Agarwal, S.-E. Yoon, and M. Gopi. Learning embedding of 3d models with quadric loss. In *BMVC*, 2019.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *TVCG*, 2003.

[4] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 1999.

[5] M. Averkiou, V. G. Kim, and N. J. Mitra. Autocorrelation descriptor for efficient co-alignment of 3d shape collections. In *Computer Graphics Forum*, 2016.

[6] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 1996.

[7] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva. A benchmark for surface reconstruction. *TOG*, 2013.

[8] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 2017.

[9] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *TVCG*, 1999.

[10] M. Botsch and L. Kobbelt. A remeshing approach to multiresolution modeling. In *SGP*, 2004.

[11] F. Calakli and G. Taubin. Ssd: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 2011.

[12] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[13] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[14] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019.

[15] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016.

[16] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136, 2008.

[17] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. In *Computer Graphics Forum*, 1998.

[18] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017.

[19] A. Dai and M. Nießner. Scan2mesh: From unstructured range scans to 3d meshes. *arXiv preprint arXiv:1811.10464*, 2018.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[21] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. Efros. What makes paris look like paris? *TOG*, 2012.

[22] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017.

[23] C. Fellbaum. Wordnet. *The encyclopedia of applied linguistics*, 2012.

[24] L. Gao, J. Yang, T. Wu, Y.-J. Yuan, H. Fu, Y.-K. Lai, and H. Zhang. Sdm-net: Deep generative network for structured deformable mesh. In *SIGGRAPH ASIA*, 2019.

[25] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, 1997.

[26] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

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

[28] G. Gkioxari, J. Malik, and J. Johnson. Mesh r-cnn. In *ICCV*, 2019.

[29] M. Gopi, S. Krishnan, and C. T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 2000.

[30] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier approach to learning 3d surface generation. In *CVPR*, 2018.

[31] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Shape correspondences from learnt template-based parametrization. In *ECCV*, 2018.

[32] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. Pcpnet learning local shape properties from raw point clouds. *Computer Graphics Forum*, 2018.

[33] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. *PAMI*, 2020.

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

[35] A. Hertz, R. Hanocka, R. Giryes, and D. Cohen-Or. Deep geometric texture synthesis. In *SIGGRAPH*, 2020.

[36] R. Hu, W. Li, O. V. Kaick, H. Huang, M. Averkiou, D. Cohen-Or, and H. Zhang. Co-locating style-defining elements on 3d shapes. *TOG*, 2017.

[37] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. Scenenn: A scene meshes dataset with annotations. In *3DV*, 2016.

[38] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NeurIPS*, 2015.

[39] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018.

[40] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. Category-specific object reconstruction from a single image. In *CVPR*, 2015.

[41] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *CVPR*, 2018.

[42] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *SGP*, 2006.

[43] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *TOG*, 2013.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *TOG*, 2017.

[46] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. Abc: A big cad model dataset for geometric deep learning. In *CVPR*, 2019.

[47] C. Kong, C.-H. Lin, and S. Lucey. Using locally corresponding cad models for dense 3d reconstructions from a single image. In *CVPR*, 2017.

[48] A. Kurenkov, J. Ji, A. Garg, V. Mehta, J. Gwak, C. Choy, and S. Savarese. Deformnet: Free-form deformation network for 3d shape reconstruction from a single image. In *WACV*, 2018.

[49] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas. Grass: Generative recursive autoencoders for shape structures. *TOG*, 2017.

[50] Y. Li, H. Su, C. R. Qi, N. Fish, D. Cohen-Or, and L. J. Guibas. Joint embeddings of shapes and images via cnn image purification. *TOG*, 2015.

[51] Y. Liao, S. Donne, and A. Geiger. Deep marching cubes: Learning explicit surface representations. In *CVPR*, 2018.

[52] C.-H. Lin, C. Kong, and S. Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *AAAI*, 2018.

[53] O. Litany, A. Bronstein, M. Bronstein, and A. Makadia. Deformable shape completion with graph convolutional autoencoders. In *CVPR*, 2018.

[54] M. Loizou, M. Averkiou, and E. Kalogerakis. Learning part boundaries from 3d point clouds. In *SGP*, 2020.

[55] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, 1987.

[56] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019.

[57] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *CVPR*, 2019.

[58] C. Nash and C. K. Williams. The shape variational autoencoder: A deep generative model of part-segmented 3d objects. *Computer Graphics Forum*, 2017.

[59] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[60] M.-E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *CVPR*, 2006.

[61] Y. Ohtake, A. Belyaev, M. Alexa, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *TOG*, 2003.

[62] J. Pan, X. Han, W. Chen, J. Tang, and K. Jia. Deep mesh reconstruction from single rgb images via topology modification networks. In *ICCV*, 2019.

[63] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019.

[64] J. K. Pontes, C. Kong, S. Sridharan, S. Lucey, A. Eriksson, and C. Fookes. Image2mesh: A learning framework for single image 3d reconstruction. In *ACCV*, 2018.

[65] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[66] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017.

[67] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 2019.

[68] A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black. Generating 3d faces using convolutional mesh autoencoders. In *ECCV*, 2018.

[69] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *ICCV*, 2017.

[70] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 1996.

[71] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 2002.

[72] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[73] E. Smith, S. Fujimoto, and D. Meger. Multi-view silhouette and depth decomposition for high resolution 3d object representation. In *NeurIPS*, 2018.

[74] E. J. Smith, S. Fujimoto, A. Romero, and D. Meger. Geometrics: Exploiting geometric structure for graph-encoded objects. *arXiv preprint arXiv:1901.11461*, 2019.

[75] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015.

[76] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *CVPR*, 2018.

[77] Q. Tan, L. Gao, Y.-K. Lai, and S. Xia. Variational autoencoders for deforming 3d mesh models. In *CVPR*, 2018.

[78] Q. Tan, L. Gao, Y.-K. Lai, J. Yang, and S. Xia. Mesh-based autoencoders for localized deformation component analysis. In *AAAI*, 2018.

[79] J. Tang, X. Han, J. Pan, K. Jia, and X. Tong. A skeleton-bridged deep learning approach for generating meshes of complex topologies from single rgb images. In *CVPR*, 2019.

[80] N. Verma, E. Boyer, and J. Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *CVPR*, 2018.

[81] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018.

[82] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *TOG*, 2017.

[83] P.-S. Wang, C.-Y. Sun, Y. Liu, and X. Tong. Adaptive o-cnn: a patch-based deep representation of 3d shapes. In *SIGGRAPH Asia*, 2018.

[84] X. Wang, Y. Xu, K. Xu, A. Tagliasacchi, B. Zhou, A. Mahdavi-Amiri, and H. Zhang. Pie-net: Parametric inference of point cloud edges. *arXiv preprint arXiv:2007.04883*, 2020.

[85] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[86] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[87] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*, 2016.

[88] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[89] Z. Wu, X. Wang, D. Lin, D. Lischinski, D. Cohen-Or, and H. Huang. Structure-aware generative network for 3d-shape modeling. *arXiv preprint arXiv:1808.03981*, 2018.

[90] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese. Objectnet3d: A large scale database for 3d object recognition. In *ECCV*, 2016.

[91] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *NeurIPS*, 2019.

[92] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NeurIPS*, 2016.

[93] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018.

[94] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In *CVPR*, 2019.

[95] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Ec-net: an edge-aware point set consolidation network. In *ECCV*, 2018.

[96] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018.