2021

Geometric Unified Method in 3D Object Classification

Mengyi Shan

Geometric Unified Method in 3D Object Classification

Mengyi Shan

Weiqing Gu, Advisor

Nicholas Pippenger, Reader

Department of Mathematics

May, 2020

Abstract

3D object classification is one of the most popular topics in the field of computer vision and computational geometry. Currently, the most popular state-of-the-art algorithm is the so-called Convolutional Neural Network (CNN) models with various representations that capture different features of the given 3D data, including voxels, local features, multi-view 2D features, and so on. With CNN as a holistic approach, researches focus on improving the accuracy and efficiency by designing the neural network architecture. This thesis aims to examine the existing work on 3D object classification and explore the underlying theory by integrating geometric approaches. By using geometric algorithms to pre-process and select data points, we dive into an existing architecture of directly feeding points into a deep CNN, and explore how geometry measures how important different points are in a CNN model. Moreover, we attempt to extract useful geometric features directly from the object data to introduce the feature matrix representation, which can be classified with distance-based approaches. We present all results of experiments and analyzed for future improvement.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest appreciation to Professor Weiqing Gu for all her instructions and help which is crucial for my thesis works, as well as all her encouragement and patience. Without her guidance and persistent help this thesis wouldn't be completed.

I'd like to extend my gratitude to Professor Nicholas Pippenger who agrees to be my second reader, especially for the useful comments on my thesis draft.

In addition, I'd like to thank Professor Jon Jacobsen for his encouragement in the middle of my thesis when I was feeling extremely unsure about my work.

Special thanks to all the professors, staff and my friends from Harvey Mudd College Department of Mathematics whose assistance was a milestone in the completion of this thesis.

Chapter 1

Introduction to 3D Object Classification

The so-called 3D object classification task is easy to interpret right from its name, even without any math or computer science background. To do 3D object classification simply means that given a certain object, represented in 3D form with either points, surfaces or other information specified, we would like to decide what this object is. For example, given a set of points comprising the skeleton of a thin rectangular board with four thin and long legs under it, we would argue that it's possibly a table, but not a television. If there is another rectangular shape put vertically on the top of the board, it's likely a chair. (See Figure 1.1 for example.)

Figure 1.1 Point clouds of a chair and a table, which are likely the object of what 3D object classification will study. Adopted from the ShapeNet Core55 dataset.

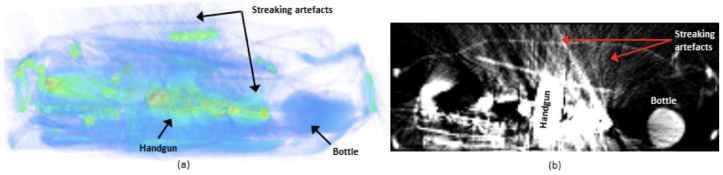


Figure 1.2 Example baggage-CT scan containing handgun and bottle. Left figure shows volumetric rendering. Right figure shows single 2D axial slice with handgun, bottle and artefacts indicated. From Flitton et al. (2015).

Apparently, the task of 3D object classification is not hard for human eyes. In our daily life, every moments our eyes take in information about the surrounding world and decide what the objects are. We could never possibly confuse a normal chair with a table. This is definitely partly because of the plenty of details we can see, including colors, texture, shape, etc. But as in Figure 1.1, even with very basic point clouds, human eyes can decide what the object is without any difficulties. This is because we can detect the shape, or **geometric** information from the objects.

However, there are also lots of cases where we need computers, instead of human eyes, to classify and identify the objects. Especially with the recent availability of inexpensive 2.5D depth sensors (e.g. Microsoft Kinect, Intel RealSense, Google Project Tango, and Apple Prime-Sense). For example, in security check of airports of train stations, a single camera view will provide scan information of a baggage and we need to identify if there's any dangerous objects in the baggage. Flitton et al. (2015) investigates the performance of a Bag of (Visual) Words (BoW) object classification model as an approach for automated threat object detection within 3D Computed Tomography (CT) imagery from a baggage security context (Figure 1.2). Zheng et al. (2009) shows the possibility of applying this technology to medical areas with their projects, which use Marginal Space Learning (MSL) to automatically detect 3D anatomical structures in many medical imaging modalities. Chen et al. (2017) proposes Multi-View 3D networks (MV3D), a sensory-fusion framework that takes both LIDAR point cloud and RGB images as input and predicts oriented 3D bounding boxes used in self-driving vehicles (1.3).

A brief review of existing 3D object classification technologies show that deep learning and especially deep neural network plays a significant role in recent 3D object classification algorithms. Dating back to previous years,

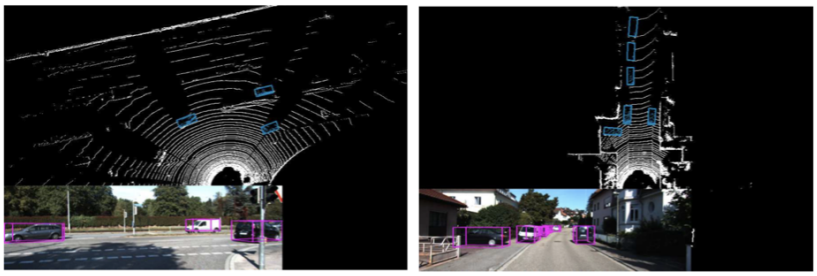


Figure 1.3 3D object detection in real world self-driving case. 3D Boxes are projected to the bird's eye view and the images. From Chen et al. (2017).

deep learning had long shown great potentiality and application in 2D object classification, detection and identification tasks, especially with the help of the new larger datasets include LabelMe by Russell et al. (2008), which consists of hundreds of thousands of fully-segmented images, and ImageNet by Li et al. (2010), which consists of over 15 million labeled high-resolution images in over 22,000 categories. Researchers include Krizhevsky et al. (2017) and Ciresan et al. (2012) explored 2D image classification and achieved very competitive accuracy with deep learning approaches.

While more and more high-quality data sets become available and the computation ability of devices significantly increases, the emphasis in object classification field in computer vision has gradually moved from 2D classification (or image classification) to directly working with 3D point set data that efficiently, directly and accurately models real life problems.

Considering 3D object classification as an interesting, worth studying and developing area, this thesis aims to review, study and compare some of the notable existing algorithms with high performance. Starting from next chapter, the theories of algorithms including the so-called Convolutional Neural Network (CNN) will be introduced. Future chapters after those will introduce how geometric ideas will be compared and combined into existing technologies.

Chapter 2

Introduction to Convolutional Neural Network

Up to now, the most popular method that produces most of the state-of-the-art algorithms with good performance is classification with Artificial Neural Networks (ANN) models, especially Convolutional Neural Notworks (CNNs). These systems are able to learn how to perform tasks by considering examples, generally instead of being explicitly programmed with domain-specific rules. This is not surprising because CNN is already proved to be a fruitful approach in 2-dimensional cases. This chapter will begin with a brief introduction to the mathematical background of artificial neural network, and then focus on discussing CNN.

Definition 2.1. A **feedforward neural network**, aka multi-layer perceptron (MLP), is a series of logistic regression models stacked on top of each other, with the final layer being either another logistic regression or a linear regression model, depending on whether we are solving a classification or regression problem. (Murphy (2012))

Suppose a MLP used to solve regression problem has two layers, then it can be mathematically expressed as

$$\begin{aligned} p(y|\mathbf{x}, \boldsymbol{\theta}) &= \mathcal{N}\left(y|\mathbf{w}^T\mathbf{z}(\mathbf{x}), \sigma^2\right) \\ \mathbf{z}(\mathbf{x}) &= g(\mathbf{V}\mathbf{x}) = \left[g\left(\mathbf{v}_1^T\mathbf{x}\right), \ldots, g\left(\mathbf{v}_H^T\mathbf{x}\right)\right] \end{aligned}$$

where g is a non-linear activation or transfer function (commonly the logistic function), $\mathbf{z}(\mathbf{x}) = \phi(\mathbf{x}, \mathbf{V})$ is called the hidden layer (a deterministic function of the input), H is the number of hidden units, $\mathbf{V}$ is the weight matrix from

the inputs to the hidden nodes, and **w** is the weight vector from the hidden nodes to the output.

Theorem 2.1. *An MLP is a **universal approximator**, meaning it can model any suitably smooth function, given enough hidden units, to any desired level of accuracy.*

Hornik (1991) proved Theorem 2.1 twenty years ago, making a solid stepping stone for other further usage of MLP to various computer-related tasks. The underlying meaning here is that as long as the computing power allows a certain number of hidden units, any smooth function underlying the regression or classification tasks should be able to be approximated by deep neural network.

The purpose of the hidden units is to learn non-linear combinations of the original inputs; this is called **feature extraction** or feature construction. These hidden features are then passed as input to the final generalized linear model.

This approach is particularly useful for problems where the original input features are not individually informative. For example, each pixel in an image is not very informative; it is the combination of pixels that tells us what objects are present. Conversely, for a task such as document classification using a bag of words representation, each feature (word count) is informative on its own, so extracting "higher order" features is less important (Murphy (2012)).

The so-called Convolutional Neural Network (CNN) is a special and widely-used type of neural network. The name "convolutional neural network" indicates that the network employs a mathematical operation called **convolution**. Convolutional layers have analogy in human neuroscience, which is actually the origin and conceptual background of all neural network models. Then convolve the input and pass its result to the next layer, which is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field.

For a more mathematically rigorous definition, convolution is a specialized kind of linear operation. Convolutional networks are by definition neural networks that replace general matrix multiplication with convolution in at least one of the layers.

Definition 2.2. Convolution is the process of taking a small matrix of real numbers (called kernel or filter), passing it over our image and transforming

the image based on the filter values.

$$\mathbf{G}[m,n] = (\mathbf{f} * \mathbf{h})[m,n] = \sum_j \sum_k \mathbf{h}[j,k]\mathbf{f}[m-j, n-k]$$

where f is the input data and h is the kernel.

With the Definition of convolution, (Definition 2.2), we can specify the following definition of Convolutional Neural Network.

Definition 2.3. A **Convolutional Neural Network** is an MLP in which the hidden units have local receptive fields, and in which the weights are tied or shared across the data (usually image), in order to reduce the number of parameters. (Murphy (2012))

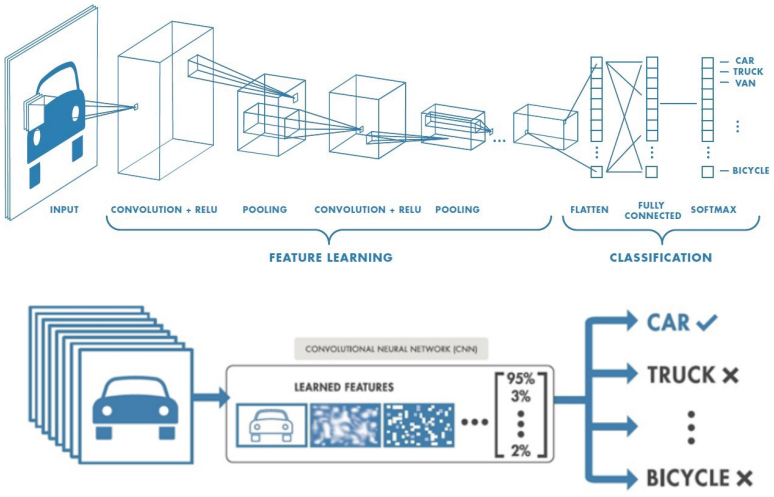


Figure 2.1 An CNN with eight different layers works on the task of identifying a car. It learns features based on local group of pixels and combines the information together through layers to generate a set of probabilities.

A convolutional neural network can assign importance (learnable weights and biases) to various aspects or parts in the input data and be able to differentiate one from the other. Intuitively, the effect of such spatial parameter tying is that any useful features that are "discovered" in some portion of the data can be re-used everywhere else without having to be independently learned. (Murphy (2012)) The resulting network then exhibits

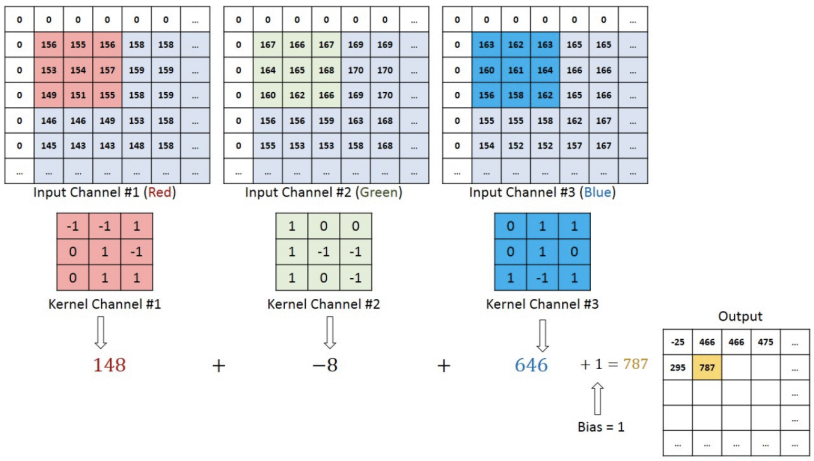


Figure 2.2 Convolutional neural network break down: calculation with a 3-by-3 kernel and three different color channels.

translation invariance, meaning it can classify patterns no matter where they occur inside the input data.

Figure 2.1 shows how a convolutional neural network works on a 2D image case. In order to identify a car, each layer of the neural network picks some details of the original image and uses a specially designed filter to catch important information. Finally the image is summarized into a list of data representing what the filter "think" is important in the image. Not surprisingly, just as in this case, convoluitonal neural networks are mostly applied to visual pattern recognition.

Figure 2.2 provides a mathematical breakdown of what a CNN does. In this case, it has a 3-by-3 filter that goes through the whole matrix, multiply with the sub-matrices, and produces results in a compressed form for each color channel of an image. Different filters could be designed for specific tasks, like edge detection.

Previous studies and those illustrations above shows clearly how convolutional network is promising in the area of 2D image processing. When problem comes to three-dimensional, however, things become much more complex. The data form of 2D images are simple in that all images can be represented with a uniform representation of **pixels**. In Figure 2.2, we have three matrices representing the values of three color channel on each single location of an image. However, this is not the only intuitive representation for

3D data, especially sparse data. Consider a chair object with no background, instead of pixel values, it might be more intuitive to view it as a cloud of points with surfaces covering the points. Some object might be hard to view inside structures, and some objects can only be viewed from several angle as 2D rendering images. All these problems remind us that although it's a great insight to generalize CNN to 3D case, we need to solve the problem of choosing the right **representation** beforehand. The next few chapters will discuss several widely used representations, including their strength and weakness.

Chapter 3

Previous Works

3.1 Volumetric Convolutional Neural Network

3.1.1 A Single Naive Representation: Voxel

As a beginning, we will start from the most intuitive generalization from 2D image to 3D object representation. When dealing with two-dimensional image data, we in fact consider it as a huge matrix of tuples, where each element in a single tuple, is a pixel indicating the color or transparency of that certain location on the graph. It's natural, then, to extend this notion to a 3D grid with unit cube called **voxel**.

Definition 3.1. A voxel represents a value on a regular grid in three-dimensional space.

The so-called voxel representation is the most widely used, traditional and popular approach. Wu et al. (2014) was the first research group introducing this groundbreaking approach which represents a geometric 3D shape as a probabilistic distribution of binary variables on a 3D voxel grid, and they applied it into 3D classification. Each 3D mesh is represented as a binary tensor: 1 indicates the voxel is inside the mesh surface, and 0 indicates the voxel is outside the mesh (i.e., it is empty space).

Wu et al. (2014) used Convolutional Deep Belief Network (CDBN), which is a powerful class of probabilistic models often used to model the joint probabilistic distribution over pixels and labels in 2D images. This is directly adopted from 2D image cases, again showing why this voxel representation is natural and intuitive.

A traditional 3D Volumetric CNN system basically consists of two parts:

1. A volumetric grid representing our estimate of spatial occupancy.
2. A 3D CNN that predicts a class label directly from the occupancy grid.

Theorem 3.1. *The energy, E of a convolutional layer can be computed as*:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{f}\sum_{j}\left(h_j^f\left(W^f * v\right)_j + c^f h_j^f\right) - \sum_{l} b_l v_l$$

where v_l denotes each visible unit, h_j^f denotes each hidden unit in a feature channel f, and W^f denotes the convolutional filter.

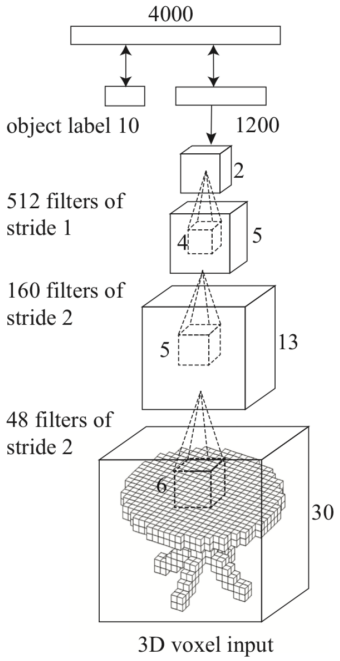


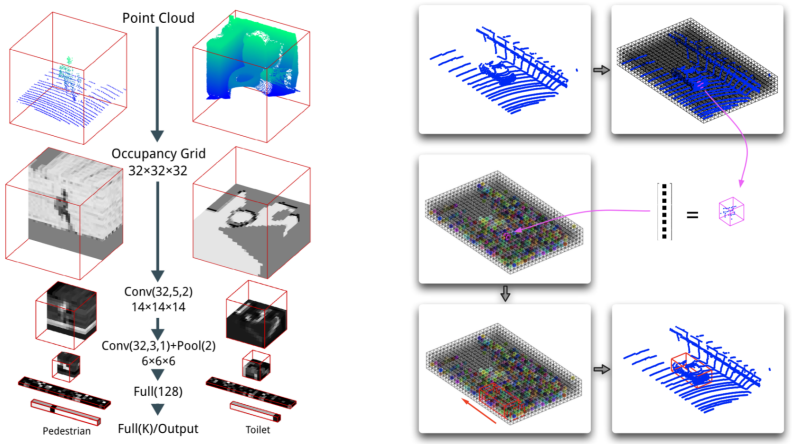
Figure 3.1 CNN representation. It is similar to Figure 2.1 except for generalizing two-dimensional case to three-dimensional case. From Wu et al. (2014).

Theorem 3.1 shows how the energy of a convolutional layer is calculated in their CNN model. It's a traditional model with visible units, hidden units, and a convolutional layer (a more complex version of the 3-by-3 matrix filter shown in Figure 2.2). Although this is not a very specifically designed method, it's extraordinary at its time because it's actually the first

CNN-based 3D object classification algorithm with good performance, and provides motivation for a great number of following researchers.

After training the CNN, the model learns the joint distribution $p(\mathbf{x}, y)$ of voxel data $\mathbf{x}$ and object category label $y \in \{1, \cdots, K\}$.

After them, other researchers improved the volumetric representation by adding details and fine-tuning the neural network for different datasets. Maturana and Scherer (2015) proposed VoxNet that integrated a volumetric occupancy grid representation with a supervised 3D CNN, which significantly improved the performance. Shi et al. (2015) converted each 3D shape into a panoramic view, namely a cylinder projection around its principle axis, and thus developing a robust representation called DeepPano. Zhao et al. (2019) considered the normal vectors after voxelized the input. He et al. (2015) presented a residual learning framework to ease the training of networks that are substantially deeper than those used previously.



a. VoxNet Structure (Maturana and Scherer (2015))

b. Vote3D Structure (Wang and Posner (2015))

Figure 3.2 Example flowcharts show the overall procedure of VoxNet and Vote3D.

In general, the volumetric CNN approach abstracts the information stored in 3D point cloud to free, occupied and unknown voxels, thus turning irregular data into regular representations. This distinction among three types provides an extremely rich information in representation, while can be stored in easy and structured data form. However, a noticeable weakness of this approach is that it requires lots of memory and computation, which

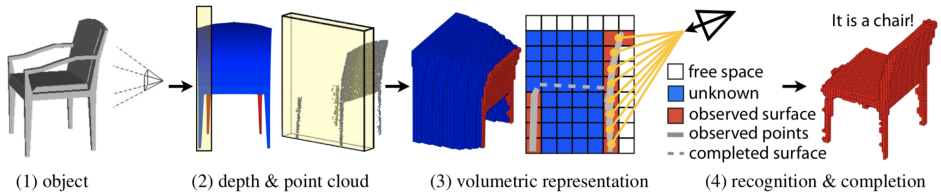


Figure 3.3 View-based 2.5D Object Recognition. (1) A depth map is taken from a physical object. (2) The depth image is captured from the back of the chair. (3) The profile of the slice and different types of voxels. (4) The recognition and shape completion result, conditioned on the observed free space and surface.

greatly restricts the amount of data it could take in as training set.

Additionally, voxelized approach ignores the extent of sparsity in 3D data representation: even though a large majority of the space are not occupied at all, the "empty" information will be marked with equal significance as the "occupied" information. Wang and Posner (2015) illustrates an approach that differs in that its nature comes from the sliding window algorithm widely used in 2D image processing. Wang and Posner generalizes this idea to 3D cases, replacing a sliding square to a sliding window. Compared with the most generic voxel-CNN, this idea specifies in dealing with sparsity in the input 3D point cloud.

3.1.2 Play with 2.5D Data

Note that up to the CNN step, only full completed data set is used to train the model and make predictions. However, as Wu et al. (2014) mentioned, it's also possible to recover full 3D shapes from view-based 2.5D depth maps. Imagine the application with Microsoft Kinetic or other depth detection hardware: in Figure 3.3 when you have a depth map viewed from only one direction, only very limited knowledge is known about the 3D object. But the volumetric ShapeNet is able to construct a probabilistic model on the grids. That is, it will generate a probability for each voxel to be either free, occupied or unknown. This probability distribution then guides us to complete the full model based on assumptions. This shape completion functionality suggests that not the full information in dataset is necessary for generating a successful classification. Then the natural question arises: what points are necessary or important for generating a good classification? This topic will be explored again in further chapters.

3.2 Surface Polygon Mesh

3.2.1 Introduce Mesh

While the locations of points are one good representation and descriptor of 3D shape, another feature marks how 3D object is in nature different from 2D images: it has the so-called surface information that cannot be easily captured by voxelized data or point cloud data. This leads to another popular representation, which is called mesh.

Definition 3.2. The polygonal mesh representation, or mesh, for short, approximates surfaces via a set of 2D polygons in 3D space. (Botsch et al. (2010))

The mesh provides an efficient, non-uniform representation of the shape. Only a small number of polygons are required to capture large, simple, surfaces. On the other hand, representation flexibility supports a higher resolution where needed, allowing a faithful reconstruction, or portrayal, of salient shape features that are often geometrically intricate. (Hanocka et al. (2018))

The most basic and commonly used mesh is the so-called **triangular mesh**. It's widely used in 3D object classification problem by representing the object with surface mesh information and build descriptors on that.

Definition 3.3. A triangle mesh $\mathcal{M}$ consists of a geometric and a topological component, where the latter can be represented by a graph structure (simplicial complex) with a set of vertices

$$\mathcal{V} = \{v_1, \ldots, v_V\}$$

and a set of triangular faces connecting them

$$\mathcal{F} = \left\{f_1, \ldots, f_F\right\}, \quad f_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V}$$

(Botsch et al. (2010))

The geometric embedding of a triangle mesh into $\mathbb{R}^3$ is specified by associating a 3D position $\mathbf{p}_i$ to each vertex $v_i \in \mathcal{V}$:

$$\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_V\}, \quad \mathbf{p}_i := \mathbf{p}(v_i) = \begin{pmatrix} x(v_i) \\ y(v_i) \\ z(v_i) \end{pmatrix} \in \mathbb{R}^3$$

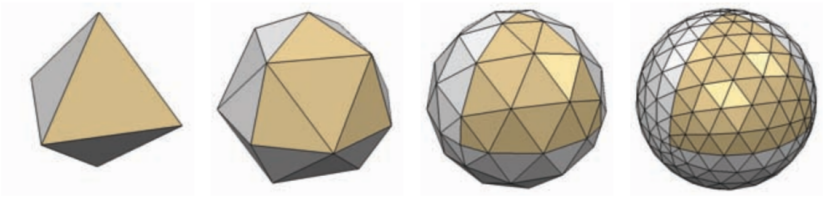


Figure 3.4 Each subdivision step halves the edge lengths, increases the number of faces by a factor of 4, and reduces the approximation error by a factor of about 1/4. From Botsch et al. (2010).

such that each face $f \in \mathcal{F}$ actually corresponds to a triangle in 3-space specified by its three vertex positions. Notice that even if the geometric embedding is defined by assigning 3D positions to the discrete vertices, the resulting polygonal surface is still a continuous surface consisting of triangular pieces with linear parameterization functions. Figure 3.4 shows and example of triangular mesh with different sizes.

Another significant characteristic of the mesh is its native provision of connectivity information. This forms a comprehensive representation of the underlying surface. Compared with the PointNet approach discussed in further chapters, mesh representation perfectly deals with the problem of connectivity and surface representation. The examplar work of Masci et al. (2015) introduced deep learning of local features on meshes and showed how to use these learned features for correspondence and retrieval. Haim et al. (2018) used toric topology to define the convolutions on the shape graph. Fouinat et al. (2018) defined a new convolutional layer that allows propagating geodesic information throughout the layers of the network.

3.2.2 Mesh CNN: Theory and Insight

As one of the pioneer work and currently one of the best performance system, Hanocka et al. (2018)'s system proposed a convolution on the edges of a mesh and also a learned mesh pooling operation that adapts to the task at hand, further expanding the research on meshes and significantly increased the performance. Note that in Figure 3.5, the most traditional non-uniform triangular mesh was used: large flat regions use a small number of large triangles, while detailed regions used a larger number of triangles.

The input edge feature for each edge is a five-dimensional vector including the dihedral angle, two inner angles and two edge-length ratios for each

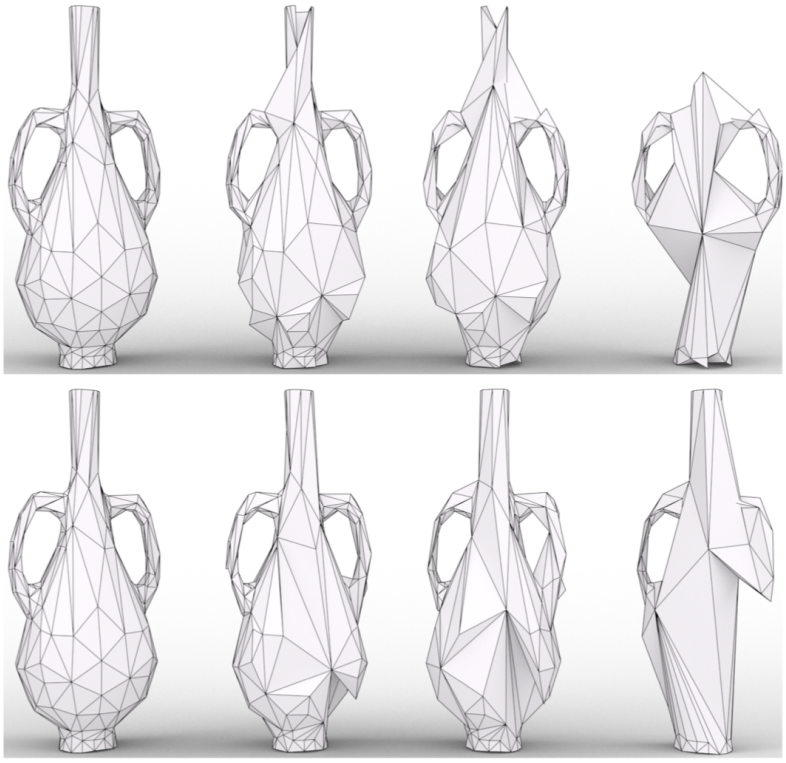


Figure 3.5 Mesh pooling operates on irregular structures and adapts spatially to the task. Unlike geometric simplification, mesh pooling delegates which edges to collapse to the network. Top: MeshCNN trained to classify whether a vase has a handle, bottom: trained on whether there is a neck. From Hanocka et al. (2018).

face. The edge ratio is between the length of the edge and the perpendicular (dotted) line in Figure 3.6 for each adjacent face. The researchers sorted each of the two face-based features (inner angles and edge-length ratios), thereby resolving the ordering ambiguity and guaranteeing invariance. Observe that these features are all relative, making them invariant **to translation, rotation and uniform scale**. (Hanocka et al. (2018).)

However, compared with the geometric feature representation, the mesh representation is not as natural and intuitive if considering how human eyes actually recognize objects: most of the objects have smooth shape, like the vase shown in Figure 3.5 in real life won't be recognized as composing

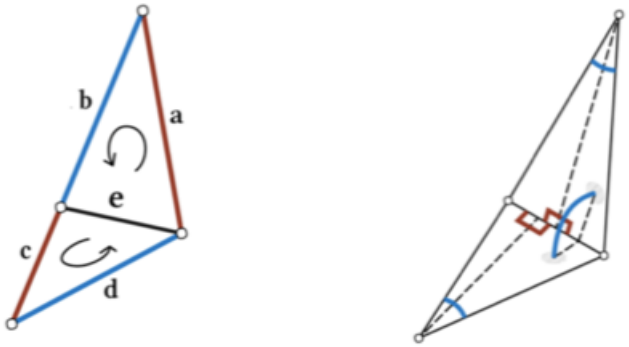


Figure 3.6 Example of representation of a triangular mesh. The 1-ring neighbors of e can be ordered as (a, b, c, d) or (c, d, a, b). This ambiguates the convolutional receptive field, hindering the formation of invariant features.

of small polygons. The success of mesh representation again urges us to explore if there's some more natural way to enhance it by mimicing how human eyes approach the identification task.

3.2.3 Local Polar Coordinate

Before introducing a more advanced technique also based on the idea of mesh, let's define the concept of manifold and Riemannian Manifolds, which will also be used in further chapters.

Definition 3.4. A **manifold** is a topological space that locally resembles Euclidean space near each point. More precisely, each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n.

The rich theory of vector spaces endowed with a Euclidean inner product can, to a great extent, be lifted to various bundles associated with a manifold. The notion of local (and global) frame plays an important technical role.

Definition 3.5. Let M be an n-dimensional smooth manifold. For any open subset, $U \subseteq M$, an n-tuple of vector fields, $(X_1, \ldots, X_n)$, over U is called a frame over U iff $\big(X_1(p), \ldots, X_n(p)\big)$ is a basis of the tangent space, T_pM, for every $p \in U$. If $U = M$, then the X_i are global sections and $(X_1, \ldots, X_n)$ is called a frame (of M)

Now we could define a Riemannian manifold.

Definition 3.6. Given a smooth n-dimensional manifold, M, a Riemannian metric on M is a family, $\left(\langle -,-\rangle_p\right)_{p\in M}$, of inner products on each tangent space, T_pM, such that $\langle -,-\rangle_p$ depends smoothly on p which means that for every chart, $\varphi_\alpha : U_\alpha \to \mathbb{R}^n$, for every frame, $(X_1,\ldots,X_n)$, on U_α, the maps

$$p \mapsto \left\langle X_i(p), X_j(p)\right\rangle_p, \quad p \in U_\alpha, \quad 1 \le i,j \le n$$

are smooth. A smooth manifold, M, with a Riemannian metric is called a Riemannian manifold.

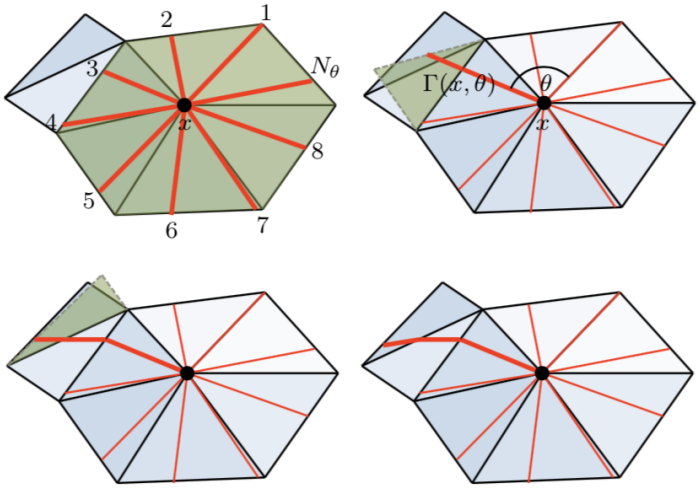


Figure 3.7 Constructed a local geodesic polar coordinates on a triangular mesh. From Masci et al. (2015).

Geodesic Convolutional Neural Networks can be considered as another kind of geometric feature recognition based on Riemannian Manifolds. GCNN is a natural generalization of the convolutional networks (CNN) paradigm to non-Euclidean manifolds. Researchers' construction is based on a local geodesic system of polar coordinates to extract "patches", which are then passed through a cascade of filters and linear and non-linear operators.

Masci et al. (2015) model a 3D shape as a connected smooth compact two-dimensional manifold (surface) X, possibly with a boundary ∂X. Locally around each point x the manifold is homeomorphic to a two-dimensional

Euclidean space referred to as the tangent plane and denoted by T_xX. A Riemannian metric is an inner product $\langle \cdot, \cdot \rangle_{T_xX} : T_xX \times T_xX \rightarrow \mathbb{R}$ on the tangent space depending smoothly on x.

Basically, how GCNN works is by building a local polar coordinate system to represent the information in a 3D object. Broadly speaking, a local feature descriptor assigns to each point on the shape a vector in some multi-dimensional descriptor space representing the local structure of the shape around that point. A global descriptor describes the whole shape, which is harder to establish and not used in this algorithm.

3.3 Multi-View Convolutional Neural Network

While representations like voxel grid or polygon mesh are considered descriptors operating on their native 3D formats, another possible direction is to represent data in a view-based term. In another word, is it possible to recognize 3D shapes from a collection of their rendered views on 2D images? Answering this question, Su et al. (2015) provides a different approach in lots of CNN attempts by tracking the 3D problem back to 2D. They tried to render 3D point cloud or shapes into 2D images and then applied 2D CNN to classify them. Even the first step of classifying only with a single rendered image shows better performance than some state-of-the-art 3D object recognition approaches. After that, they designed a special algorithm also by convolutional neural network to combine several 2D classification result of a single object to get an overall shape descriptor that leads to even more accurate prediction performance.

3.3.1 Build Rendered 3D Models

3D models in online databases are typically stored as polygon meshes, which are collections of points connected with edges forming faces. To generate rendered views of polygon meshes, Su et al. (2015) used the Phong reflection model (Phong (1975)) which basically consists of two parts:

1. Render the mesh polygons under a perspective projection. This type of representation has the advantage that it avoids the problem posed by mathematically curved surface approaches, of solving higher order equations.

2. Determine the pixel color by interpolating the reflected intensity of the polygon vertices. Taking into consideration that the light received

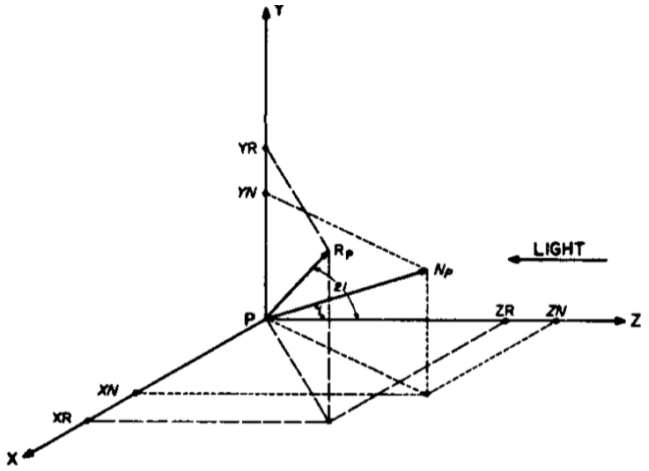


Figure 3.8 Determination of the formula for shading. This model is abstracted into the $x - y - z$ coordinates while light comes from the z axis. From Phong (1975).

by the eye is provided one part by the diffuse reflection and one part by the specular reflection of the incident light, the shading at point P on an object can be computed as:

$$S_p = C_p[\cos(i)(1-d)+d] + W(i)[\cos(s)]^n$$

where C_p is the reflection coefficient of the object at point P for a certain wavelength. i is the incident angle. d is the environmental diffuse reflection coefficient. W_i is a function which gives the ratio of the specular reflected light and the incident light as a function of the incident angle i. s is the angle between the direction of the reflected light and the line of sight. n is a power which models the specular reflected light for each material.

After building the 3D shape, the researchers actually setup viewpoints (so-called virtual cameras) to create a multi-view shape representation. In Figure 3.9, it shows how twelve cameras are set up around the object chair and twelve images will be produced and classified by each camera.

3.3.2 Detailed CNN Design for Aggregation

In classification and retrieval test, the most crucial part is how to aggregate the results from all those different rendered cameras. The following theorem

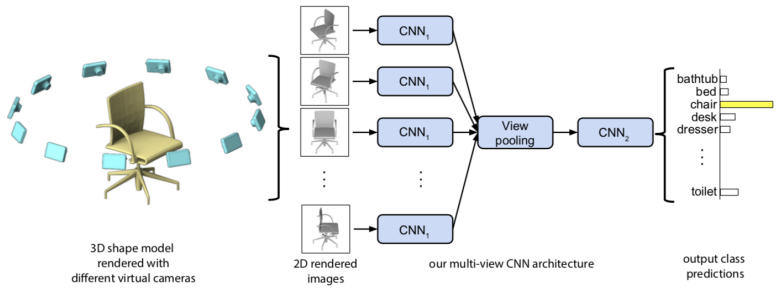


Figure 3.9 Multi-view CNN for 3D shape recognition is illustrated here. At test time a 3D shape is rendered from 12 different views to extract view based features. These are then pooled across views to obtain a compact shape descriptor.

provides a metric to calculate the distance between two 3D objects by computing all $n_x \times n_y$ pairwise distances between images. Simply averaging or concatenating the image descriptors leads to inferior performance.

Theorem 3.2. *Distance between two 3D shapes are defined as*

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_j \min_i \left\|\mathbf{x}_i - \mathbf{y}_j\right\|_2}{2n_y} + \frac{\sum_i \min_j \left\|\mathbf{x}_i - \mathbf{y}_j\right\|_2}{2n_x}$$

Here x, y represent the two 3D objects. x_i, y_j represents the view from each camera, with i, j going from 1 to the number of cameras. In a word, we take the sum of the minimum of distance between views, which help us rectify the possible rotation of the objects, then take the average to decide the estimated distance between objects.

Result shows that 2D information can be highly informative to 3D structures. This method significantly increases spatial resolution (although with less explicit depth information) compared to CNNs taking 3D information as input. It can also take advantage of the much more mature and abundant pre-trained models and data sets of 2D object classification, including LabelMe by Russell et al. (2008) and ImageNet by Li et al. (2010) discussed before.

3.3.3 Advanced Related Works

Yan et al. (2016) discussed Perspective Transformer Nets that formulated the learning process as an interaction between 3D and 2D representations and proposed an encoder-decoder network with a novel projection loss

defined by the perspective transformation. The projection loss enabled the unsupervised learning using 2D observation without explicit 3D supervision. Gao et al. (2018) explored pairwise multiview information and introduced a novel pairwise Multi-View Convolutional Neural Network for 3D Object Recognition (PMV-CNN for short), where automatic feature extraction and object recognition were put into a unify CNN architecture.

Large scale experiments demonstrate that the pairwise architecture is very useful when the number of labeled training samples is very small. Recent researches on multi-view representation basically focused on two directions: multi-view matching and label propagation. An exemplar research in the former area is Wang et al. (2012) which used manifold to manifold distance. The later one is generally related with graph propagation (Gao et al. (2012)).

This approach is intuitive and also widely used at the beginning because the huge number of developed, available dataset and technology for 2D image classification. Evolving from 2D to 3D in this sense is probably the most intuitive approach for experienced researchers. However, the propagation algorithm that builds the bridge from 2D to 3D is not robust enough. For example, Theorem 3.2 used to calculate the distance between 2 objects is still in a primary form. Although it's designed to handle rotation and different angle, it's not elaborated enough to handle all subtle cases.

Moreover, the greatest problem for this approach is still that the camera photo could only take care of outside features. For example, if there's a hole inside the object or some subtle features that is blocked from direct sight but can be reflected by 3D point cloud, the camera couldn't take care of that. Especially if an object is not convex, several camera views will produce significant torsion.

3.4 *PointNet*: Direct Point Cloud Representation

3.4.1 Point Cloud Representation: Opportunity and Challenges

The most basic and naive form of 3D object representation should be simply a cloud of 3D points, embedding in the $x-y-z$ coordinate, each represented as a tuple of (x, y, z). Obviously, it's not possible to take all points from a single object. So, depending on the density and location of these points, the level of difficulty varies. Besides the problem of point density, the 3D point cloud representation has the following few features (Qi et al. (2016)):

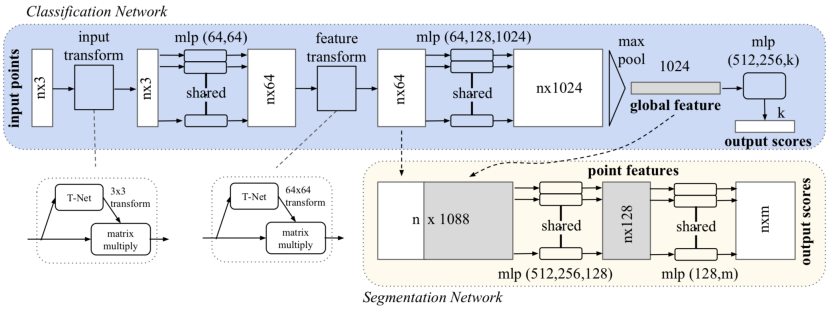


Figure 3.10 PointNet CNN takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. From Qi et al. (2016).

1. Unordered. Unlike volumetric representation, a set of points is not assigned in a special order. A network that consumes N 3D point sets needs to be invariant to $N!$ permutations of the input set in data feeding order.

2. Interaction among points. The points are from a space with a distance metric. It means that points are not isolated, and neighboring points form a meaningful subset. Therefore, the model needs to be able to capture local structures from nearby points, and the combinatorial interactions among local structures.

3. Invariance under transformations. As a geometric object, the learned representation of the point set should be invariant to certain transformations. For example, rotating and translating points all together should not modify the global point cloud category nor the segmentation of the points.

Based on this idea, Qi et al. (2016) presented a modern neural network approach that instead of using CNNs on voxels, takes the raw 3D point cloud data as input. That is, each point is represented by just its three coordinates (x, y, z). Because of the three features mentioned before, it takes special care to design a specific CNN model that satisfies the features of point cloud. Qi et al. (2016)'s algorithm learned optimization function that select interesting or informative points. Then the fully connected layers aggregated these learnt optimal values into the global descriptor and did classification or segmentation. The whole architecture is shown in the block diagram in Figure 3.10.

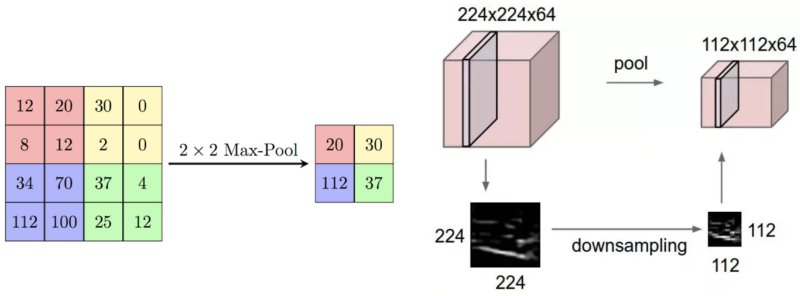


Figure 3.11 Diagrams show examples of max pooling layers. Left image is a pictorial explanation in two-dimensional matrix case. Right image is a real life application on three dimensional vision data.

3.4.2 Max Pooling Layer for Special Features

What's important and unique in this neural network structure are the three specific features designed to accommodate the special properties of point cloud, especially using symmetric function and max pooling layer to make the model invariant to input permutation.

Definition 3.7. Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. (Anand J. Kulkarni and Suresh Chandra Satapathy (2020))

In another word, max pooling layer does nothing more than dividing the data into small sections, and take the maximum point of each section to represent the whole bundle. This is used to reduce the size of data and can easily catch the prominent features. Figure 3.11 shows two examples on calculating max pooling layer by hand. In the left image, for example, we want to take each small square of 2 by 2 dimension and use the single maximum number to represent it, resulting in a matrix that is four times smaller.

We value the max pooling function for the sense of symmetric it shows in nature. In order to make the model invariant to input permutation, it's convenient to use a simple symmetric function to aggregate the information from each point. The high level idea behind *PointNet* is to approximate a

general function defined on a point set by applying a symmetric function on transformed elements in the set, illustrated by Theorem 3.3 below.

Theorem 3.3.

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n))$$

where $f : 2^{\mathbb{R}^N} \to \mathbb{R}, h : \mathbb{R}^N \to \mathbb{R}^K$ *and* $g : \underbrace{\mathbb{R}^K \times \cdots \times \mathbb{R}^K}_{n} \to \mathbb{R}$ *is a symmetric function.*

By Theorem 3.3, the idea of PointNet is to approximate a general function defined on a point set by applying a symmetric function on transformed elements in the set. Empirically, this module is very simple: it approximates h by a multi-layer perceptron network and g by a composition of a single variable function and a max pooling function. Through a collection of h, we can learn a number of f' s to capture different properties of the set.

Theorem 2.1 already shows that MLP can model any suitably smooth function, given enough hidden units, to any desired level of accuracy. In this case where we add the max pooling layers, it's also required to show the universal approximation ability of the neural network to continuous set functions.

3.4.3 Arbitrarily Approximate and Critical Point

For a more mathematical rigorous explanation of the theory behind, let's start with a few background definition.

Definition 3.8. Let X and Y be two non-empty subsets of a metric space (M, d). We define their Hausdorff distance $d_H(X, Y)$ by

$$d_{\mathrm{H}}(X, Y) = \max\left\{ \sup_{x \in X \in Y} d(x, y), \sup_{y \in Y \in X} d(x, y) \right\}$$

where sup represents the supremum and infthe infimum. Equivalently

$$d_H(X, Y) = \inf\{\varepsilon \geq 0; X \subseteq Y_\varepsilon \text{ and } Y \subseteq X_\varepsilon\}$$

where

$$X_\varepsilon := \bigcup_{x \in X} \{z \in M; d(z, x) \leq \varepsilon\}$$

that is, the set of all points within ε of the set X (sometimes called the ε -fattening of X or a generalized ball of radius ε around X).

Formally, let $\mathcal{X} = \{S : S \subseteq [0,1]^m \text{ and } |S| = n\}$, $f : \mathcal{X} \to \mathbb{R}$ is a continuous set function on $\mathcal{X}$ w.r.t to Hausdorff distance $d_H(\cdot,\cdot)$, i.e., $\forall \epsilon > 0, \exists \delta > 0$, for any $S, S' \in \mathcal{X}$ if $d_H(S, S') < \delta$, then $\left|f(S) - f(S')\right| < \epsilon$. Then Theorem 3.4 below says that f can be arbitrarily approximated by our network given enough neurons at the max pooling layer, i.e., K is sufficiently large.

Theorem 3.4. (*Qi et al.* (2016)) *Suppose $f : \mathcal{X} \to \mathbb{R}$ is a continuous set function w.r.t Hausdorff distance $d_H(\cdot,\cdot)$. $\forall \epsilon > 0, \exists$ a continuous function h and a symmetric function $g(x_1, \ldots, x_n) = \gamma \circ MAX$, such that for any $S \in \mathcal{X}$*

$$\left|f(S) - \gamma\left(\mathrm{MAX}_{x_i \in S}\{h(x_i)\}\right)\right| < \epsilon$$

where $x_1, \ldots, x_n$ is the full list of elements in S ordered arbitrarily, γ is a continuous function, and MAX is a vector max operator that takes n vectors as input and returns a new vector of the element-wise maximum.

With this in mind, we can go back to the beginning of the chapter, where we raise the question about density of points for describing a 3D object. In fact, Theorem 3.5 below argues that for any 3D point cloud object, only part of the points are required to successfully identify the object.

Theorem 3.5. (*Qi et al.* (2016)) *Define $\mathbf{u} = \underset{x_i \in S}{\mathrm{MAX}}\{h(x_i)\}$ to be the sub-network of f which maps a point set in $[0,1]^m$ to a K-dimensional vector.*

Suppose $u : \mathcal{X} \to \mathbb{R}^K$ such that $\mathbf{u} = \underset{x_i \in S}{\mathrm{MAX}}\{h(x_i)\}$ and $f = \gamma \circ \mathbf{u}$. Then

1. $\forall S, \exists C_S, \mathcal{N}_S \subseteq \mathcal{X}, f(T) = f(S)$ *if* $C_S \subseteq T \subseteq \mathcal{N}_S$
2. $|C_S| \leq K$

To prove this theorem, notice that obviously, $\forall S \in \mathcal{X}$, $f(S)$ is determined by u (S). So we only need to prove that $\forall S, \exists C_S, \mathcal{N}_S \subseteq \mathcal{X}, f(T) = f(S)$ if $C_S \subseteq T \subseteq \mathcal{N}_S$ For the j th dimension as the output of $\mathbf{u}$, there exists at least one $x_j \in \mathcal{X}$ such that $h_j(x_j) = \mathbf{u}_j$, where h_j is the j th dimension of the output vector from h. Take C_S as the union of all x_j for $j = 1, \ldots, K$. Then, C_S satisfies the above condition.

(1) says that $f(S)$ is unchanged up to the input corruption if all points in C_S are preserved; it is also unchanged with extra noise points up to $\mathcal{N}_S$. (2) says that C_S only contains a bounded number of points, determined by K in (1). In other words, $f(S)$ is in fact totally determined by a finite subset $C_S \subseteq S$ of less or equal to K elements. We therefore call C_S the critical point set of S and K the bottleneck dimension of f. Note that this set of critical points

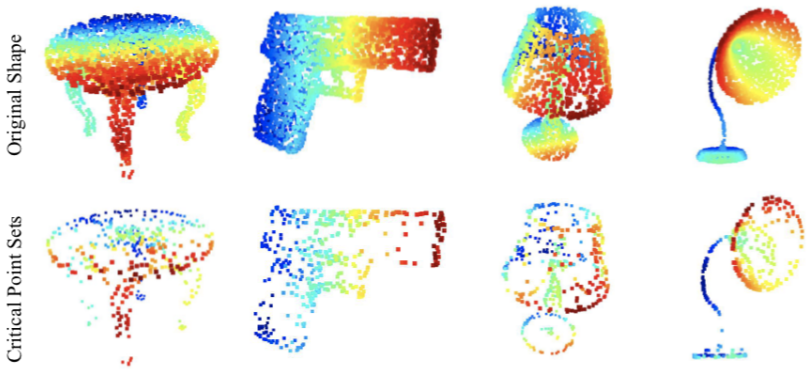


Figure 3.12 While critical points jointly determine the global shape feature for a given shape, any point cloud that falls between the critical points set and the upper bound shape gives exactly the same feature.

are different from the so-called critical points in the definition of differential geometry.

Compared with the voxelized approaches, this representation is much more direct and takes less voluminous input. Also it doesn't require the input to be quantized, and thus saving lots of important information. But the most severe problem of it is that while considering point cloud information, it selectively ignores surface information, local features and connectivity status.

Further work on PointNet-liked CNNs generally aimed to improve its ability to recognize local feature and topological structure. Qi et al. (2017) then further improved the PointNet model's performance by introducing a hierarchical neural network that applies PointNet recursively on a nested partitioning of the input point set. By exploiting metric space distances, this network is able to learn local features with increasing contextual scales. Shi et al. (2018) developed PointRCNN based upon the original PointNet model, which is a bottom-to-up 3D proposal generation method directly generates robust 3D proposals from point clouds, which is both efficient and quantization free. Guerrero et al. (2017), similarly, attempted to improve PointNet's performance by extracting local features directly from point cloud.

3.5 Geometric Feature Extraction

Traditional methods explored and included some non-neural network approaches to this problem, including SVM, Bayes model, propagational graph, and feature extraction.

3.5.1 Shape Descriptor

Fang et al. (2015) firstly converted the 3D data into a vector, by extracting traditional shape features and then use a fully connected net to classify the shape. To explore deeper into their work, we need to first define what is a shape signature and what is a shape descriptor, the representation they used to describe a 3D object.

Definition 3.9. 3D shape signatures and descriptors are succinct and compact representations of 3D object that capture the geometric essence of a 3D object. Shape signature is referred to as a local description for a point on a 3D surface and shape descriptor is referred to as a global description for the entire shape. (Fang et al. (2015))

Hand-crafted shape descriptors are sometimes not robust enough in order to solve the problem of structural variations present in 3D models. Discriminative feature learning from large datasets provides an alternative way to construct deformation-invariant features. This method has been widely used in computer vision and image processing. Specifically, there are three types of traditional shape descriptors with relatively good performance introduced in the sections below. (Fang et al. (2015)).

Heat shape descriptor (HeatSD)

The 3D model is represented as a graph $G = (V, E, W)$, where V is the set of vertices, E is the set of edges, and W represents the weight values for the edges. Given a graph constructed by connecting pairs of vertices on a surface with weighted edges, the heat flow on the surface can be quantitatively approximated by the heat kernel:

$$h_t\left(p_1, p_2\right) = \sum_{i=0}^{\infty} \left(-\lambda_i' t\right) \phi_i\left(p_1\right) \phi_i\left(p_2\right)$$

which is a function of two points p_1 and p_2 on the network at a given time t (Guo et al. (2015)).

Heat kernel signature is then defined by

$$HKS(p) = \left(H_{t_1}(p,p), H_{t_2}(p,p), \ldots, H_{t_n}(p,p)\right)$$

where p denotes a point on the surface, $HKS(p)$ denotes the heat kernel signature at point p, $H_t(p,p)$ denotes the heat kernel value at point p, t_n denotes the diffusion time of the $n-$ th sample point. HKS has attractive geometric properties that includes invariance to isometric transformation, robustness against other geometric changes and local numerical noise, and multi-scale representation with scale parameter of diffusion time (Fang et al. (2015)).

Eigen-shape descriptor (ESD)

$$S = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)\left(x_i - \mu\right)^T$$

where S is the covariance matrix for the set of training shape descriptors x_i, and

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n$$

where v_i is the i-th Eigen-shape.

Fisher-shape descriptor (ESD)

$$S_B = \sum_{i=1}^{c} N_i \left(\mu_i - \mu\right)\left(\mu_i - \mu\right)^T$$

where S_B is the scatter matrix reflecting the margin among different class and μ_i is the mean of class i and μ is the total mean.

$$S_W = \sum_{i=1}^{c}\sum_{x_j \in X_i}\left(x_j - \mu_i\right)\left(x_j - \mu_i\right)^T$$

where S_W is the scatter matrix reflecting closeness within the same classes, μ_i is the mean of class i.

$$S_B v_i = \lambda_i S_w v_i$$

where v_i is the i-th Fisher-shape.

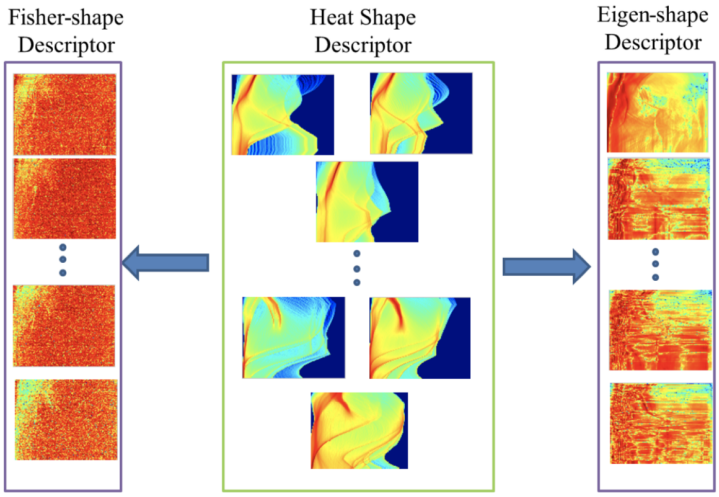


Figure 3.13 Pipeline of generating Eigen-shape descriptor and Fisher-shape descriptor. A collection of Heat shape descriptors are used to train Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). The trained Eigen-shape descriptors are illustrated on the right and Fisher-shape descriptors are shown on the left. From Fang et al. (2015).

3.5.2 Geometric Feature DNN

Guo et al. (2015) combines mesh-labeling methodology with feature extraction and identification to build a feature-based deep CNNs. In their algorithm, CNNs are first trained in a supervised manner by using a large pool of classical geometric features. In the training process, these low-level features are nonlinearly combined and hierarchically compressed to generate a compact and effective representation for each triangle on the mesh. Compared with more traditional feature-extraction classification algorithm, this one excels in that besides a fixed set of basic geometric objects, it aims to combine and compress features together to get more complicated features.

3.5.3 Pair-wise Feature Matching

Another direction of feature-based algorithm is using logic and building a pair-wise feature comparison table. Ma et al. (2018) considered pairwise geometric feature of 3D objects and used a tailored matching algorithm. On the other hand, geometric features are extracted and modified and fed into

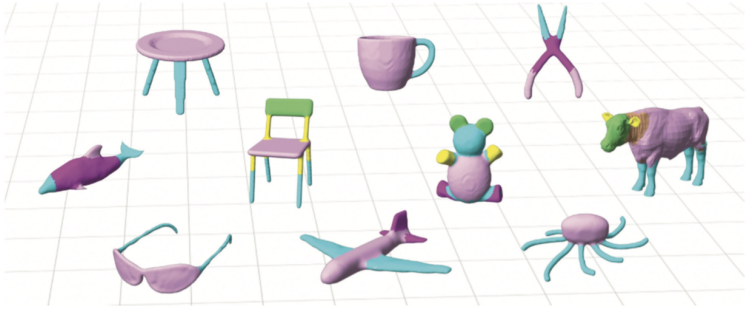


Figure 3.14 The geometric feature DNN can extract features of various parts from a single object. From Guo et al. (2015)

DNNs to do higher-level classification work. With the knowledge that some building objects can be recognized based on the features of their lower level geometry primitives. Figure 3.15 shows the process of feature matching.

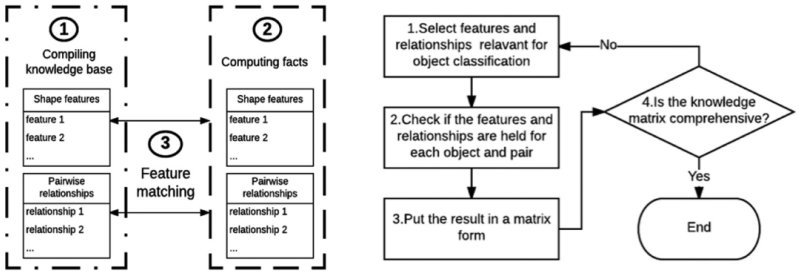


Figure 3.15 Compiles a knowledge matrix requires calculation of geometric features from each object and comparison between each pair of objects. Then a logical judgement process will be used to classify based on information in the knowledge matrix. From Ma et al. (2018).

Theorem 3.6. *The similarity of two vectors could be measured by the angle between them, which can be computed by the inner product of the two vectors. It is computed using the following equation, where* $i, j \in [1,2,3], i \neq j$ *and* $m, n \in [1,2,3,4]$.

$$\mathbf{S}_{(i,j)(m,n)} = \frac{\mathbf{R}^*(i,j) \cdot \mathbf{R}(m,n)}{\left|\mathbf{R}^*(i,j)\right| \left|\mathbf{R}(m,n)\right|}$$

Theorem 7.1 gives a way to calculate the matching coefficient of two different objects with geometric features already identified. This result can be used to judged if two objects match or not based on the available features.

While feature extraction and similarity measure are intuitive, and this approach is based on knowledge about the object shapes themselves and doesn't need lots of data to train, this geometric feature approach is considered natural and pioneering. However, it also suffers from the problem that it restricts to specific types of data. Only when we know what's the possible composition of a bridge can we successfully identify parts of it. If the object is entirely unknown even without possible classes, it's hard to break it down into parts.

Even with these limitations, geometric feature recognition representation is still considered a promising direction.

Chapter 4

Geometry and CNN: Comparison and Combination

Previous chapters give an overall discussion and summary of current state-of-the-art approaches to solve 3D object classification problem. It's not surprised that this problem has so many different possible approaches, some of them entirely unrelated and some rationales even contradicting with each other. This is because 3D object classification, from the origin, is a pretty basic, popular and well-studied problem with an extremely rich space of possible representation, which promotes the usage of convolutional neural network, debatably the single most powerful tool in image processing and computer vision.

CNNs are successful in that they try to use tons of neurons to mimic how human brain works. But then a natural problem rises: how do we human recognize 3D objects? We must rely internally on some CNN-liked structures, but on a higher level than that, if you are asked to describe why this object is a chair but not a table, you would be able to provide some ideas that are directly fed into your eyes, and possibly a simpler processor than the complex layers of neurons. For example, you will be able to identify that a chair has a vertical back supporting structure, while the table only has a board staying on four legs. On the other hand, a bed might likely be a combination of both, but with a larger area. You will be able to identify a leg by detecting some prominent structure that protruded from the main object, and decide that this cannot be a bath tub. All these standards seem to be natural to human, but neural network doesn't really care about this: it takes in all the data, no matter important or no, and no matter what the

original shape appearance is. Then it does lots of arithmetic to identify what are the features among those $n \times 3$ matrices. They do achieve pretty nice results, but from a human's perspective, this is kind of counter-intuitive.

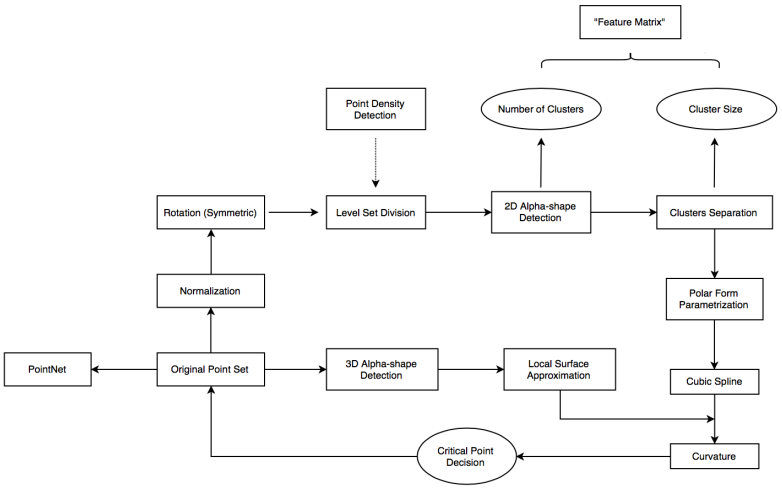


Figure 4.1 Flowchart shows the logic of further explorations.

We human recognizes and classifies objects naturally by **geometry** and **topology** characteristics, like those described in the previous paragraph. CNN, on the other hand, is like a blackbox that takes all of the data and produce a model based on some features: probably the geometric characteristics and probably something else. Not satisfied by this black-box approach, we want to ask the following questions:

1. What feature, or just data points in this case, does a CNN need to produce correct predictions? What points are crucial to the success of a CNN, and what points are unnecessary and probably more like noise? Could we assume that the points that represent more prominent geometry characteristics will be more critical in a 3D point cloud CNN?

2. If we have the idea of what geometric feature is important, on the other hand, is it possible to get rid of CNN from the beginning, but instead define the distance between two objects simply in terms of the geometric features we detected, and use more basic distance-based classifier to recognize the objects? This might not work for

complicated data set, especially those with lots of rotation. But for an easy and intuitive data set, could CNN be replaced by more intuitive, human-liked direct geometric approach?

For 1, the concept of critical point and bottleneck dimension defined in Chapter 3 shows possibility to find those geometrically important points. For 2, in Chapter 3, when we discussed the concepts of shape descriptor and shape signature, we are moving on the scale between pure geometric and pure black-box approach, away from neural network.

Figure 4.1 shows the rough process of possible explorations that could be done in terms of answering these two questions. For finding critical points, we will use the concept of curvature (both for curves and for surfaces) to evaluate how important a single point is. This will be discussed deeper in Chapter 6. Chapter 5 will introduce an algorithm to get the shape formed by points, since we assume our experiments to start with point clouds.

As answers to the second question, Chapter 7 will define our novel idea of building a feature matrix for a geometric 3D object, and how to define distance function on the matrix space for further classification tasks.

Finally, Chapter will specify more experimental details and results, as well as discussion and possible further works.

Chapter 5

Alpha Shape: The Shape Formed By Points

5.1 A Game With Ice Cream Spoon

Consider the normal scenario of 3D object classification, where we are given a set of points but no more information. *PointNet* discussed in Chapter 3.4 described an algorithm that takes all the points as input to a convolutional neural network and generates the classification result.

As analyzed before, not all points in a certain point cloud is crucial to the classification. In fact, by Theorem 3.5 , only those so-called **critical points** are necessary for generating the correct result. In order to figure out what are the crucial points in a geometric way, it's like assuming we are given a set $S \subset \mathbb{R}^d$ of n points in 2D or 3D and we want to have something like "the shape formed by these points." This is quite a vague notion and there are probably many possible interpretations, the α-shape being one of them.

One can intuitively think of an α-shape as the following. Imagine a huge mass of ice-cream making up the space $\mathbb{R}^d$ and containing the points S as "hard" chocolate pieces. Using one of these sphere-formed ice-cream spoons we carve out all parts of the ice-cream block we can reach without bumping into chocolate pieces, thereby even carving out holes in the inside (eg. parts not reachable by simply moving the spoon from the outside). We will eventually end up with a (not necessarily convex) object bounded by caps, arcs and points. If we now straighten all "round" faces to triangles and line segments, we have an intuitive description of what is called the α-shape of S. Figure 5.1 is an example for this process in 2D (where our ice-cream

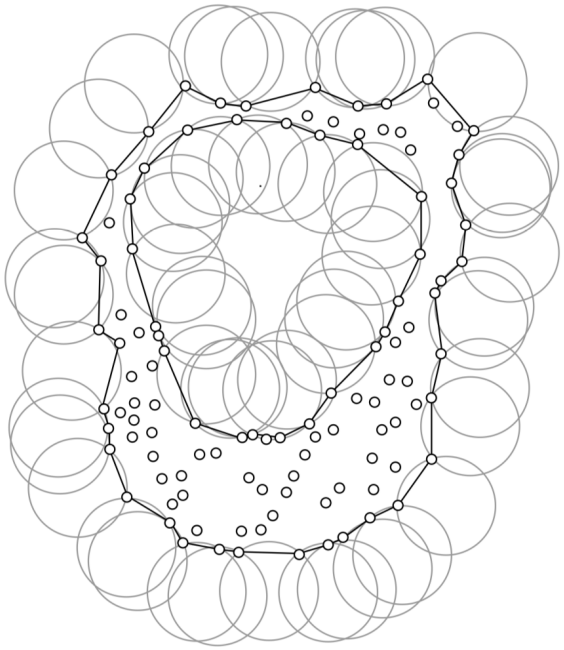


Figure 5.1 An intuitive understanding of alpha-shape in 2D case can be expressed by the circles as the metaphored ice-cream spoons. From Edelsbrunner and Mücke (1994).

spoon is simply a circle).

And what is α in the game? α is the radius of the carving spoon. A very small value will allow us to eat up all of the ice-cream except the chocolate points S themselves. Thus we already see that the α -shape of S degenerates to the point-set S for $\alpha \to 0$. On the other hand, a huge value of α will prevent us even from moving the spoon between two points since it's way too large. So we will never spoon up ice-cream lying in the inside of the convex hull of S, and hence the α-shape for $\alpha \to \infty$ is the convex hull of S. This Theorem will be proved later in the chapter, once we define the mathematically rigorous concept of α-shape.

Theorem 5.1.

$$\lim_{\alpha \to 0} \mathcal{S}_\alpha = S$$

and

$$\lim_{\alpha \to \infty} \mathcal{S}_\alpha = \text{conv } S$$

5.2 Definition

We will start to define the mathematical details of α-shape by first defining a few basic background concepts.

Definition 5.1. For $0 < \lambda < \infty$, let an λ-ball be an open ball with radius λ. Now, a certain λ-ball b (at a given location) is called **empty** if $b \cap S = \emptyset$. With this, a k-simplex Δ_T is said to be α**-exposed** if there exists an empty α -ball with $T = \partial b \cap S$ where ∂b is the surface of the sphere (for $d = 3$) or the circle ($d = 2$) bounding b, respectively.

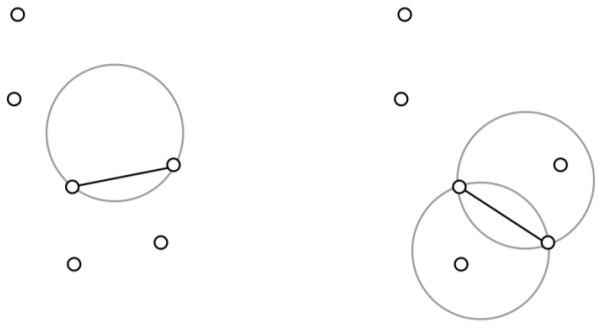


Figure 5.2 An α-exposed ball (left) doesn't contain any other point while a non α-exposed ball does (right).

Coming back to the ice-cream scenario from the introduction we notice that a face is on the boundary of our intuitive α -shape (to be defined) if the ice-cream spoon hits against one or more of the points in S. But this simply means that the simplex spanned by these points is α -exposed. This leads to the following definition of the "boundary" of the α-shape.

Definition 5.2. The boundary $\partial\mathcal{S}_\alpha$ of the α-shape of the point set S consists of all k-simplices of S for $0 \leq k < d$ which are α-exposed,

$$\partial\mathcal{S}_\alpha = \left\{\Delta_T | T \subset S, |T| \leq d \text{ and } \Delta_T \alpha\text{-exposed}\right\}$$

Alpha shape is often compared with another concept, the so-called convex hull.

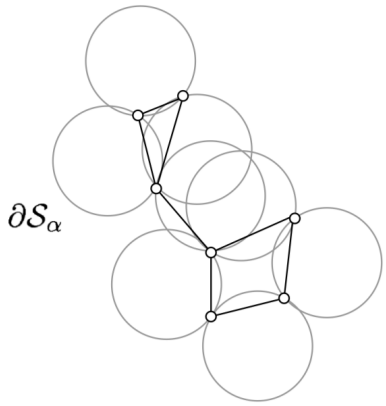


Figure 5.3 Boundary of an alpha-shape

Definition 5.3. The convex hull of a sef points S in n dimensions is the intersection of all convex sets containing S. For N points $p_1, \ldots, p_N$, the convex hull C is then given by the

$$c_{=1}\left\{\sum_{j=1}^{N}\lambda_j p_j : \lambda_j \geq 0 \text{ forall jand } \sum_{j=1}^{N}\lambda_j = 1\right\}$$

Since an infinitely small ball exposes any point in S but none of the higherdimensional simplices, and since an α-ball with α greater than the radius of the smallest enclosing circle of the points S doesn't allow an interior simplex to be α -exposed, we get $\lim_{\alpha\to 0}\partial\mathcal{S}_\alpha = S$ and $\lim_{\alpha\to\infty}\partial\mathcal{S}_\alpha = \partial\,\text{conv}\,S$, respectively. And thus we can follow the intuition and validate previous Theorem 5.1.

5.3 Edelsbrunner's Algorithm

At this stage it's relatively straightforward to formulate a algorithm to obtain the α -shape.

- Compute the Delaunay triangulation of S, knowing that the boundary of our α -shape is contained in it.
- Then we determine C_α by inspecting all simplices Δ_T in DT(S) : If the $\sigma_{T^-\text{ball around }\mu_T}$ is empty and $\sigma_T < \alpha(-$ this is the alpha test-) we accept Δ_T as a member of C_α, together with all its faces.

- All d-simplices of C_α make up the interior of $\mathcal{S}_\alpha$. All simplices on the boundary ∂C_α form $\partial \mathcal{S}_\alpha$

For this algorithm to work we need three things.

- The Delaunay triangulation (which isn't a problem, there are algorithms to do that).
- A test to check whether or not the σ_T- ball is empty. This too can be done, for instance by checking whether p lies in the said ball for every $p \in S \backslash T$.
- A way to see whether a simplex Δ_T in C_α lies on the boundary. For this, let's assume that the Delaunay triangulation algorithm returns (in addition to the triangulation) for every simplex whether or not it is on the boundary $\partial \operatorname{conv}(S)$ of the convex hull. Then:

  Theorem 5.2. *Let Δ_T be a simplex in $C_\alpha(S)$. If $\Delta_T \in \partial \operatorname{conv}(S)$, then it is obviously on the boundary of C_α. Otherwise, it is in the interior of C_α iff all of the simplices in $DT(S)$ properly containing Δ_T lie in C_α, too.*

The algorithm is an efficient implementation of the above procedure, with two additional advantages: First of all the algorithm does not run for a single value α but computes an implicit representation instead which can be used to deduce $\mathcal{S}_\alpha$ for any value of α. More precisely, the algorithm computes for every simplex $\Delta_T \in \mathrm{DT}(S)$ an intervall $I = [a, \infty]$ with the interpretation that $\Delta_T \in \mathcal{S}_\alpha$ iff $\alpha \in I$.[6] (That there is such an interval is a consequence of observation 9, as already mentionned.) Second, the algorithm not only distinguishes among interior and non-interior simplices of C_α (as we have done above), but makes three distinctions instead, one more among the non-interior simplices.

When we increase α continuously from 0 towards ∞ and consider a simplex $\Delta_T \in \mathrm{DT}(S)$, we see (using observation 9) that there are two (possibly empty) intervals (a, b) and $(b, \infty)($ with $0 \leq a \leq b \leq \infty)$ such that

$$\Delta_T \quad \text{is} \quad \begin{cases} \text{not in } C_\alpha & (\text{ for } \alpha < a) \\ \text{in } \partial C_\alpha & (\text{for } \alpha \in (a, b)) \\ \text{interior to } C_\alpha & (\text{ for } \alpha \in (b, \infty)) \end{cases}$$

Altogether we get the following algorithm on the next page:

Algorithm 1 Algorithm for calculation of alpha shape

{ Given a point-set $S \subset \mathbb{R}^d$, computes a list R of simplices Δ_T and }
{two lists B, I of intervals such that $\Delta_T \in \partial\mathcal{S}_\alpha$ if and only if $\alpha \in B_T$}
{ and $\Delta_T \in \text{int}(\mathcal{S}_\alpha)$ if and only if $\alpha \in I_T$.}
R := DT(S)
for each d-simplex $\Delta_T \in R$ **do**
  $B_T := \emptyset; I_T := (\sigma_i, \infty)$
end for
for $k := d - 1$ to 0 by -1 **do**
  for each k -simplex $\Delta_T \in R$ **do**
    if b_T is empty **then**
      $a := \sigma_T$
    else$a := \min\{a_U | B_U = (a_U, b_u), \Delta_U(k+1) -$ Simplex, $T \subset U\}$
    end if
    if $\Delta_T \in \partial\,\text{conv}(S)$ **then**
      $b := \infty$
    else$b := \max\{a_U | B_U = (a_U, b_u), B_U d$ -Simplex mit $T \subset U\}$
    end if
    $B_T := (a, b); I_T := (b, \infty)$
  end for
end for

5.4 Alpha-shape on Real Examples

α-shape can be used to describe the shape of any 3D point cloud. It can be either used at 2D level set, or directly at 3D shapes. In order to simplify the experiment and get an intuitive understanding, we will start with the 2D case of α-shape on level sets. See Figure 5.5 for an example that will be used throughout the thesis.

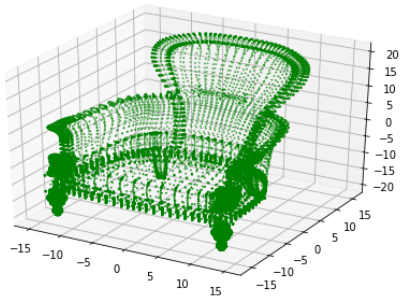


Figure 5.4 Original 3D shape. This example will be used throughout the chapters for other illustration.

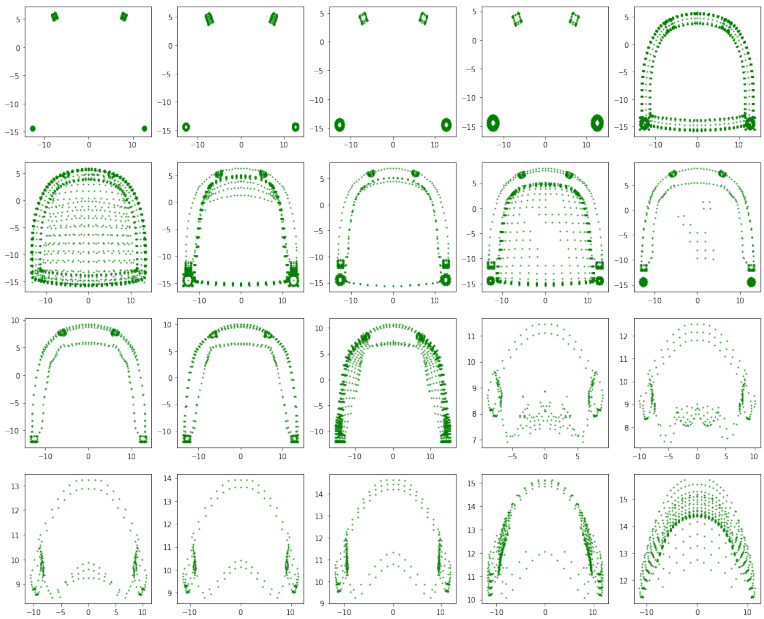


Figure 5.5 All 2D level sets of a 3D object (chair).

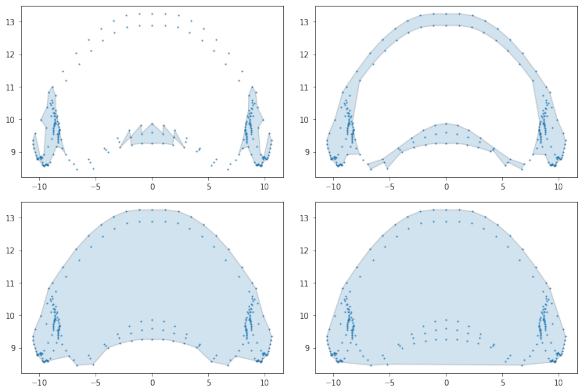


Figure 5.6 α-Shape with 4 different αs: 0.01, 0.5, 1, 2.

Note that different α will lead to entirely different results, and some of them don't make sense in our case of application. As α goes to infinity, the shape goes to the convex hull. While as α goes to 0, the shape approaches the simple set of all points. Figure 5.6 shows the α-shape on the same level set. Note that in this case, the left bottom figure with $\alpha = 1$ shows the best approximation of the shape.

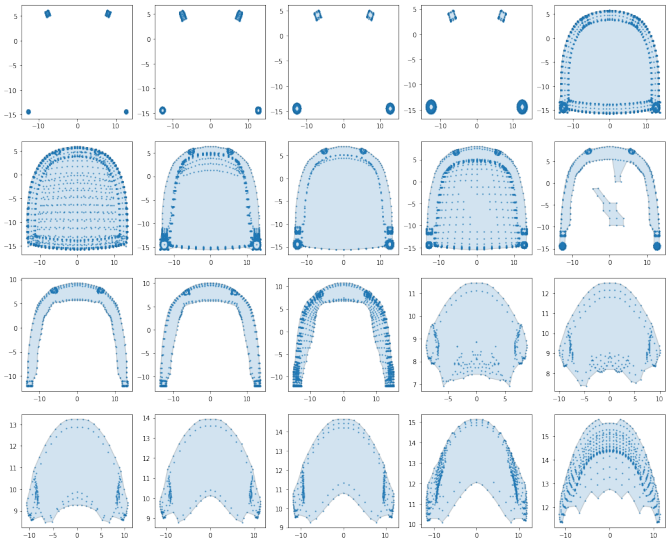


Figure 5.7 Find α-shape of all level sets.

Running our α-shape algorithm on all of the level sets, we can see it successfully approximates the shapes of different parts of the object.

If we generalize the algorithm to 3D case (there is no existing implementation in Python3, but it would be easily implemented from the algorithm provided in the section before), we could approximate the shape of an object at a higher level. Figure 5.8 shows such an example: we can observe that it does capture most of the important details of the object while significantly reducing the number of points used here.

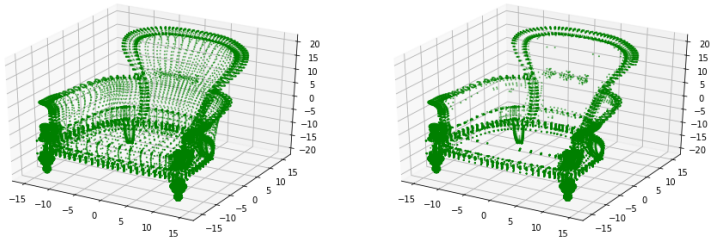


Figure 5.8 3D α-shape result.

As mentioned in Chapter 4, the data processed by the α-shape algorithm could be used in two different ways. It can either be used to decide which points are important as a pre-processing step for *PointNet*, or it can be used directly to extract information describing the shape of an object on a level set level. These two directions will be discussed further in Chapter 6 and Chapter 7. Then experiments and results will be summarized and discussed in Chapter 8.

Chapter 6

Curvature: The Amount of Deviation

6.1 Basic Curve

In mathematics, a curve (also called a curved line in older texts) is an object similar to a line which does not have to be straight.

Let $\alpha : I \to R^3$ be a parametrized differentiable curve. For each $t \in I$ where $\alpha'(t) \neq 0$, there is a well-defined straight line, which contains the point $\alpha(t)$ and the vector $\alpha'(t)$. This line is called the tangent line to α at t.

Definition 6.1. A parametrized differentiable curve $\alpha : I \to R^3$ is said to be regular if α'(t) $\neq 0$ for all t $\in$ I

From now on we shall consider only regular parametrized differentiable curves (and, for convenience, shall usually omit the word differentiable). Given $t \in I$, the arc length of a regular parametrized curve $\alpha : I \to R^3$ from the point t_0, is by definition

$$s(t) = \int_{t_0}^{t} |\alpha'(t)|\, dt$$

where

$$|\alpha'(t)| = \sqrt{(x'(t))^2 + \big(y'(t)\big)^2 + (z'(t))^2}$$

is the length of the vector $\alpha'(t)$. since $\alpha'(t) \neq 0$, the arc.length s is a differentiable function of t and $ds/dt = |\alpha'(t)|$

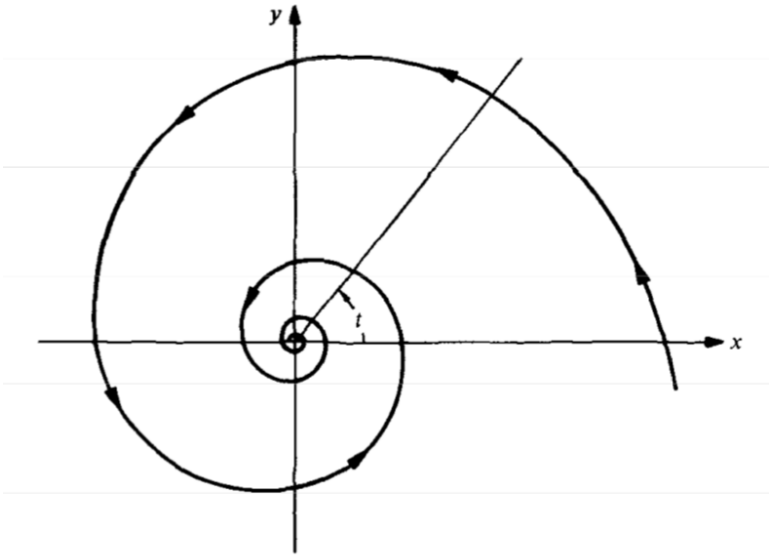


Figure 6.1 Logarithmic curve example

It can happen that the parameter t is already the arc length measured from some point. In this case, $ds/dt = 1 = |\alpha'(t)|$; that is, the velocity vector has constant length equal to 1. Conversely, if $|\alpha'(t)| \equiv 1$, then

$$s = \int_{t_0}^{t} dt = t - t_0$$

i.e., t is the arc length of α some point.

Let $\alpha : I = (a, b) \to R^3$ be a curve parametrized by arc length s. since the tangent vector $\alpha'(s)$ has unit length, the norm $|\alpha''(s)|$ of the second derivative measures the rate of change of the angle which neighboring tangents make with the tangent at s. $|\alpha''(s)|$ gives, therefore, a measure of how rapidly the curve pulls away from the tangent line at s, in a neighborhood of s (see Fig. $1 - 14$). This suggests the following definition.

Definition 6.2. Let $\alpha : \mathbf{I} \to \mathbf{R}^3$ be a curve parametrized by arc length $\mathbf{s} \in$ I. The number $|\alpha''(s)| =$ k(s) is called the curvature of α at s

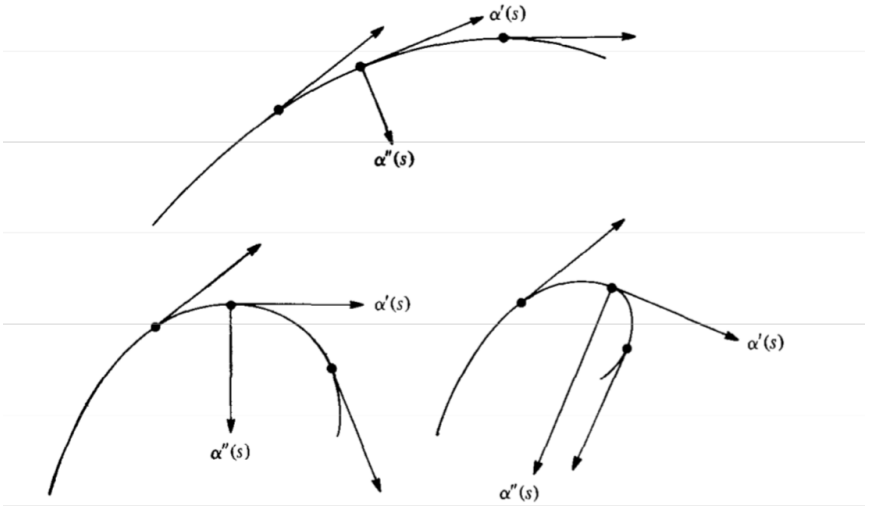


Figure 6.2 Example of curvature for various curves.

6.2 Interpolation and Curvature in Real Data

We can see how curvature can be used as a measure of how much changes are happening at a certain location. This makes it an ideal metric to represent the importance of a point in 3D point cloud model, since we expect that points in an actively changing area might be more crucial to identify the object.

Because the 3D object data doesn't appear in the form of a curve or a surface, it won't be intuitive how to calculate the curvature. In order to simplify this problem, we will start with curvature on curves, which can be approximated after preprocessing the data with α-shape discussed in Chapter 5.

Let's go back to the same figure of all level sets that is presented in Chapter 5. Although it doesn't make sense to consider the set of points as a curve, one possible direction is considering the outer most points (i.e., those on the boundary of α-shape) as making up a curve. We will take all those points and change the coordinates to polar by the following rule:

$$\begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases} \tag{6.1}$$

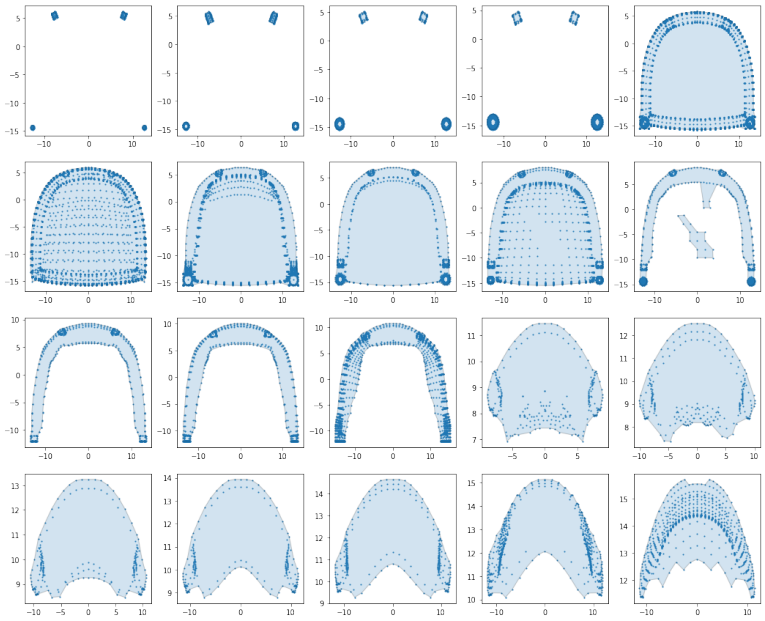


Figure 6.3 Find α-shape of all level sets.

Then we approximate the curve using cubic spline interpolation.

Definition 6.3. Cubic spline is a spline constructed of piece-wise third-order polynomials which pass through a set of m control points. The second derivative of each polynomial is commonly set to zero at the endpoints, since this provides a boundary condition that completes the system of $m - 2$ equations.

After approximating the equations, we quantitized the segments and calculated the curvature at the middle point each small segment. Figure 6.4 uses red to mark the points with large curvature than a learned threshold.

Only keeping those points with curvature larger than the threshold will significantly decrease the amount of points, while keeping the most important information. The amount to decrease can be controlled by selecting the threshold or approach to learn the threshold.

6.3 Curvature for Surfaces

For a curve drawn on a surface (embedded in three-dimensional Euclidean space), several curvatures are defined, which relates the direction of curva-

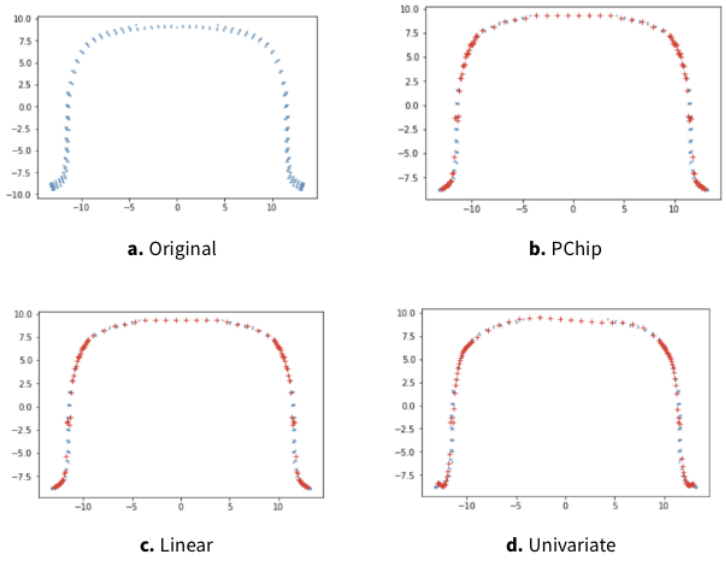


Figure 6.4 Different cubic splines result

ture to the surface's unit normal vector. These are the normal curvature, geodesic curvature and geodesic torsion. The curvature of curves drawn on a surface is the main tool for the defining and studying the curvature of the surface.

Informally Gauss defined the curvature of a surface in terms of the curvatures of certain plane curves connected with the surface. He later found a series of equivalent definitions. One of the first was in terms of the area-expanding properties of the Gauss map, a map from the surface to a 2-dimensional sphere. However, before obtaining a more intrinsic definition in terms of the area and angles of small triangles, Gauss needed to make an in-depth investigation of the properties of geodesics on the surface, i.e. paths of shortest length between two fixed points on the surface.

The Gaussian curvature at a point on an embedded smooth surface given locally by the equation $z = F(x, y)$ in Eucidean space ($\mathbf{E}^3$), is defined to be the product of the principal curvatures at the point; $^{(4)4}$ the mean curvature is defined to be their average. The principal curvatures are the maximum and minimum curvatures of the plane curves obtained by intersecting the surface with planes normal to the tangent plane at the point. If the point is $(0, 0, 0)$ with tangent plane $z = 0$, then, after a rotation about the z -axis

setting the coefficient on xy to zero, F will have the Taylor series expansion

$$F(x, y) = \frac{1}{2}k_1x^2 + \frac{1}{2}k_2y^2 + \cdots$$

The principal curvatures are k_1 and k_2. In this case, the Gaussian curvature is given by $K = k_1 \cdot k_2$ and the mean curvature by

$$K_m = \frac{1}{2}(k_1 + k_2)$$

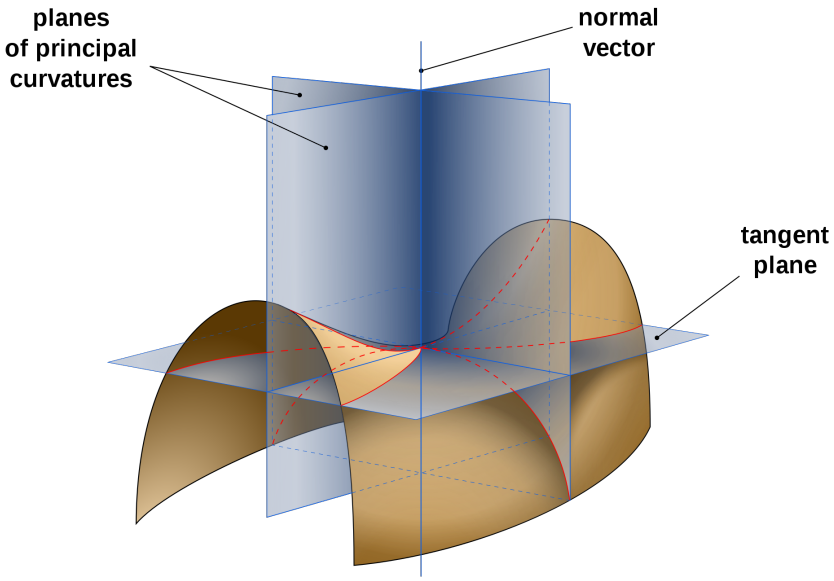


Figure 6.5 The principal curvatures at a point on a surface.

Since K and K_n are invariant under isometries of $\mathbf{E}^3$,in general

$$_K = \frac{RT - S^2}{(1 + P^2 + Q^2)^2}$$

$$K_m = \frac{ET + GR - 2FS}{2(1 + P^2 + Q^2)^2}$$

where the derivatives at the point are given by's)

$$P = F_x, Q = F_y, R = F_{xx}, S = F_{xy}, T = F_w$$

$$E = 1 + F_{r^2}^2 G = 1 + F_y^2, F = F_F F_y$$

For every oriented embedded surface the Gauss map is the map into the unit sphere sending each point to the (outward pointing) unit normal vector to the oriented tangent plane at the P coordinates the map sends (x, y, z) to

$$N(x, y, z) = \frac{1}{\sqrt{1 + P^2 + Q^2}}(P, Q, -1)$$

Direct computation shows that: the Gaussian curvature is the Jacobian of the Gauss map.

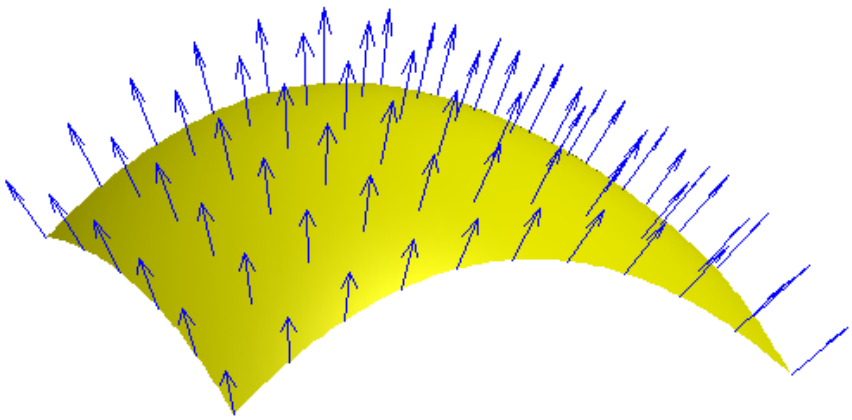


Figure 6.6 The Gauss map sends a point on the surface to the outward pointing unit normal vector, a point on S^2.

6.4 Curvature for Riemannian Manifold

When talking about curvature in 3D cases, we cannot avoid the discussion of manifold, especially Riemannian manifold. For Riemannian manifolds (of dimension at least two) that are not necessarily embedded in a Euclidean space, one can define the curvature intrinsically, that is, without referring to an external space.

Before going into details, we want to first give definition for manifold and Riemannian manifold.

Definition 6.4. An n-dimensional manifold M is a set of points such that each point has n-dimensional extensions in its neighborhood. That is, such a neighborhood is topologically equivalent to an n-dimensional Euclidean space.

Mathematically trained readers may know the rigorous definition of a manifold: A manifold M is a Hausdorff space which is covered by a number of open sets called coordinate neighborhoods, such that there exists an isomorphism between a coordinate neighborhood and a Euclidean space. The isomorphism defines a local coordinate system in the neighborhood. M is called a differentiable manifold when the coordinate transformations are differentiable. See textbooks on modern differential geometry. Our definition is intuitive, not mathematically rigorous, but is sufficient for understanding information geometry and its applications.

Definition 6.5. A Riemannian manifold is defined as a smooth manifold with a smooth section of the positive-definite quadratic forms on the tangent bundle.

A Riemannian metric (tensor) makes it possible to define several geometric notions on a Riemannian manifold, such as angle at an intersection, length of a curve, area of a surface and higher-dimensional analogues (volume, etc.), extrinsic curvature of submanifolds, and intrinsic curvature of the manifold itself.

Note that with the discussions of curvature on surface and on manifold, it's definitely possible to extend our experiment with curvature on the level sets to direct 3D data. Remember that we can also successfully calculate the 3D α-shape with relatively stability. This will be a possible further work.

Chapter 7

Feature Matrix and Shape Partial Derivative

Previous chapters introduce how to find the **critical** points from a cloud of data points comprising a whole 3D object. Namely, we assume that the points on the boundary of an object are crucial, and the points at locations experiencing more changes are crucial. With these two intuitions, we translate them into the mathematical concepts of α-shape and curvature. However, up to now, all work are done based on an existing architecture of Convolutional Neural Network: we are just adding a sense of geometry to preprocess the input that is fed into the built CNN.

As illustrated in Chapter 4, we also want to ask the question of whether it is possible to completely discard the traditional CNN approach, and do the classification work based on other more intuitive and geometric-like features.

7.1 Feature Matrix: Idea and Definition

It is natural to start again with α-shape here, because the α-shape is the most suitable algorithm to approximate the shape formed by a point cloud in this application scenario.

From Figure 7.1, we can easily observe the difference of the overall shape among different level sets. From the bottom four level sets, we can see four small clusters of points indicating four legs. Then among the next five level sets, we can see a large, convex shape approximating the middle part of the chair. The next four level sets show the shape of the back part. The top-most

seven level sets (with an upward curve on the top and a non-convex part at the bottom) shows the top shape, especially arm part of the chair.

There are two important features describing the shape of each level set which would be natural but useful information for classification.

- How many clusters are there? (Indicating **topology** information and structure of the object.)
- What's the shape of each cluster? (Indicating **geometry** information and details of the object.)

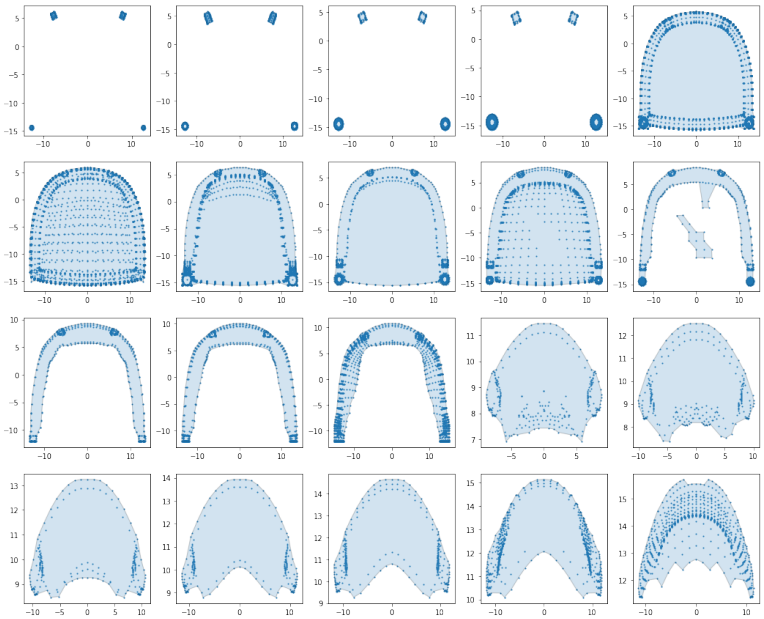


Figure 7.1 Find α-shape of all level sets.

Fortunately, the information of number of clusters and area can be naturally derived from α-shape due to its nature of Polygon object. (Recall the definition of α-shape in Chapter 5 with Delaunay Triangulation.) We might calculate the sum of areas of a triangulation easily from the coordinates with the formula:

$$S = \left| \frac{Ax\left(By - C_y\right) + Bx\left(C_y - A_y\right) + C_x\left(A_y - B_y\right)}{2} \right|$$

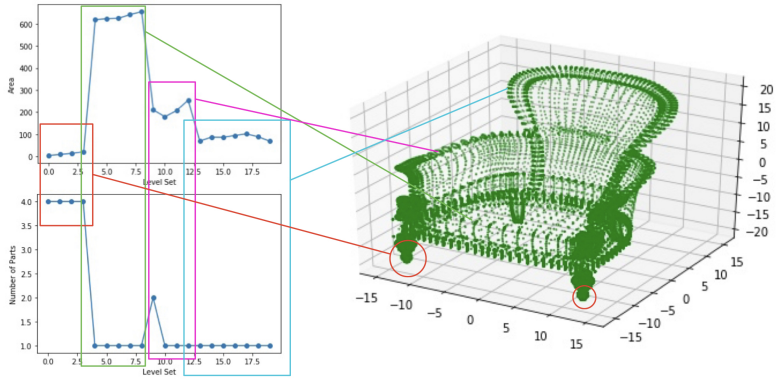


Figure 7.2 Correspondence between the geometric features and the original object.

Figure 7.2 shows the result of those two different measures on all level sets, and how the changing pattern corresponds exactly to the different parts of a chair, which follows our intuition tightly.

Based on the two features discussed above, we could build a **feature matrix** that relates each height to a tuple (N, S). Where N is the number of clusters and S is the size of the shape. This matrix, instead of the full point set, can capture the most important information of this object. Currently its size is n where n is the number of level sets we take. In the future we might want to add more features.

The feature matrix will be a natural measure to describe the shape of a 3D object with limited and extremely structured information. Based on the feature matrices, we could design distance functions so that a basic supervised, distance-based classification algorithm could be applied here, like k-nearest neighbors.

7.2 Distance Function Design

After defining the concept of feature matrix, it's worthwhile to describe the idea that two feature matrices will be close to each other when two objects look similar. Consider the feature matrices of two objects.

$$F_1 = \begin{bmatrix} N_{1,1} & S_{1,1} \\ N_{1,2} & S_{1,2} \\ \vdots & \vdots \\ N_{1,n} & S_{1,n} \end{bmatrix}$$

and

$$F_2 = \begin{bmatrix} N_{2,1} & S_{2,1} \\ N_{2,2} & S_{2,2} \\ \vdots & \vdots \\ N_{2,n} & S_{2,n} \end{bmatrix}$$

Thinking of a natural way to measure the distance between this two matrices: First, we definitely want to treat the first and second column differently by defining two different distance functions and take the weighted sum of those two.

$$\text{Dist}(F_1, F_2) = \omega_1 \text{Dist}(N_1, N_2) + \omega_2 \text{Dist}(S_1, S_2)$$

For the number of cluster column, we need to take care of the fact that some objects have sparse data and a single level set might be entirely empty. This doesn't mean that the object is broken down into two pieces from this point, so we will not penalize a difference between 0 and other number of clusters. We define the following function on each level set.

$$\text{Dist}(N_{1,i}, N_{2,i}) = \begin{cases} |N_{1,i} - N_{2,i}| & \text{if both are not zero} \\ 1 & \text{otherwise} \end{cases}$$

And then simply take the L2-norm on all sets.

$$\text{Dist}(N_1, N_2) = \sqrt{\sum_{i=1}^{n} (\text{Dist}(N_{1,i}, N_{2,i}))^2}$$

For the second column, simply taking the L2-norm suffices.

$$\text{Dist}(S_1, S_2) = \sqrt{\sum_{i=1}^{n} (S_{1,i} - S_{2,i})^2}$$

This distance function is designed without any training on the data, except for two hyper-parameters adjusting the weights. So it probably won't perform really good on the data. One promising possibility is to train the distance function instead of using a hard, hand-crafted function based on the training data.

7.3 Shape Partial Derivative: Another Measure

In the previous sections, we discussed several measures that could be used in the feature matrix. Note that both the number of clusters measure and the average size of clusters measure is a sole description of a single level set. Considering that we are interested in how level sets change, it might make more sense to include a measure describing the trend of changes among adjacent level sets. Here we introduce the idea of **Shape Partial Derivative**.

Looking at a single level set $z = c$ in the (x, y, z) coordinate, we could define the derivative d as the distance between boundary points divided by the distance between two level sets. Namely

$$d = \frac{f(x, y, c + \delta) - f(x, y, c)}{\delta}$$

where δ is a hyperparameter depending on the data.

The limitation of the feature matrix approach is that it only takes level set level information, and wait for the distance function to discriminate the changes as the height of level set changes. The shape partial derivative, on the other hand, by taking differences between different level sets, summarizes the information from each level set to a higher level. Thus it could more vividly describe the shape of the object data.

Chapter 8

Data, Experiment and Results

8.1 Data

The dataset used in the PointNet paper is the Princeton ModelNet40 dataset by Wu et al. (2014). It contains a list of the most common object categories in the world, using the statistics obtained from the SUN database. Once they established a vocabulary for objects, they collected 3D CAD models belonging to each object category using online search engines by querying for each object category term. Then, they hired human workers on Amazon Mechanical Turk to manually decide whether each CAD model belongs to the specified cateogries, using their in-house designed tool with quality control. To obtain a very clean dataset, they choose 10 popular object categories, and manually deleted the models that did not belong to these categories. Furthermore, they manually aligned the orientation of the CAD models for this 10-class subset as well. All the dataset is available for download at their wedsite.

In this project, in order to keep a comparison basis with lots of 3D point cloud experiments project, we also use the ModelNet dataset for training and evaluation. However, as the ModelNet40 dataset is too large and takes up too much time with limited computing resources, we switch to the ModelNet10 dataset for the first step.

Figure 8.1 is an example of 15 objects for chairs in the ModelNet10 dataset. We can observe from the dataset that even in the same category, different objects may be very different in their lookings. Some are very dense with over 180000 points, while some others are very sparse with only several hundreds points.

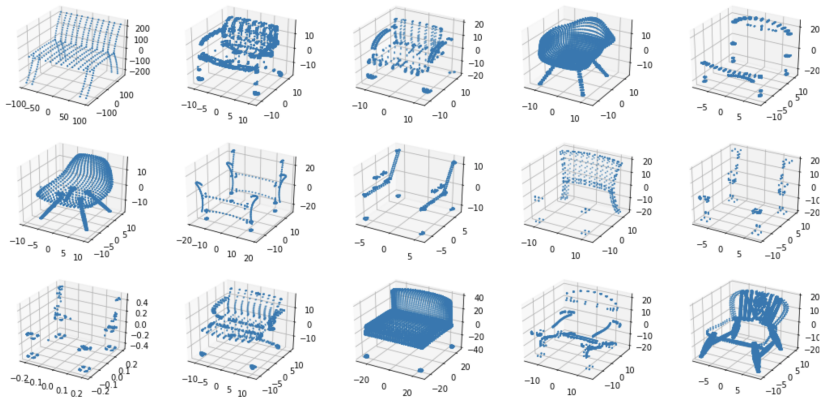


Figure 8.1 ModelNet10 data example

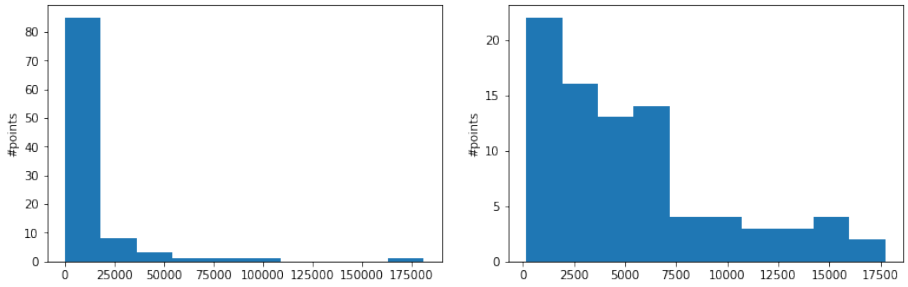


Figure 8.2 ModelNet10 model size

In order to get a straightforward understanding of the size of each 3D point cloud here, the histogram in Figure 2.2 is used to show the number of points in each model. We can see that most of the models have 2500-7500 points, while some have much more or less. Since points with about 700 points are successfully in recognition, we assumes that most of the points in most of the models are redundant, i.e., not "critical points".

This dataset provides us with special meaning for study because of this varied size of object point cloud. We'll be able to decrease the size of each object by identifying what points might be crucial in the process with geometric methods.

Another issue with this dataset is that it fails to represent all the surface information in the 3D point cloud. It does have a part specifying surface mesh information, but we will ignore that part just for this project. As a result, some objects might not look like it if we plot a scatter plot. This

requires more effort to identify critical points. On the other hand, the fact that projects like PointNet can identify objects only with point cloud information again assures us that only a limited subset of the original points are required to build a successful model.

8.2 Experiments

8.2.1 *PointNet* and Curvature

In this experiment, we took the CNN model from *PointNet* paper without reconstructing the architecture, because it's proved to work well on this same dataset and we would have a fair comparison.

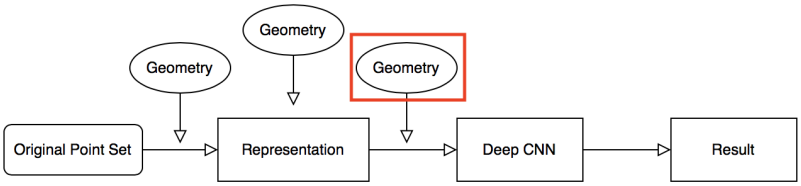


Figure 8.3 Re-visit the system flowchart: adding geometric features to CNN

This experiment runs with the whole data set, with train-test split of 80% vs 20%. It uses Harvey Mudd College Mathematics Department's fast machine to train the neural net and test on unseen data. The details about the neural network architecture is introduced in Chapter 3.4.

8.2.2 Feature Matrix

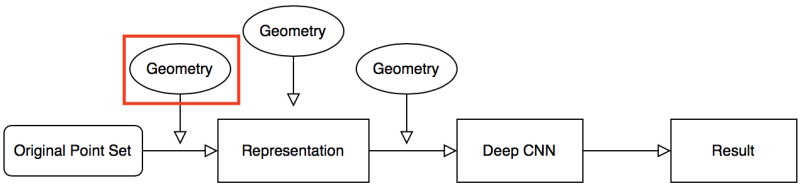


Figure 8.4 Re-visit the system flowchart: using geometric without CNN

In this experiment, we take our own feature matrix representation without building a neural network. We define the distance function as discussed in Chapter 7.

In terms of distance-based classification method, we take advantage of the most traditional approach k-nearest neighbor with $k = 5$.

This experiment runs with the whole data set, with train-test split of 80% vs 20%.

8.3 Results and Analysis

All the experiments results including accuracy and training time are summarized in 8.1.

8.3.1 *PointNet* and Curvature

The results with fewer points are slightly less accurate than the original result, but also take less time. When we applied all new algorithms including α-shape and cubic spline, the evaluation f-score decreases, but because the size of the dataset is much smaller, the time taken is also significantly smaller.

This result suggests that those points selected by the simple combination of α-shape and curvature might not be the **critical points** defined in Theorem 3.5. Analyzing the wrongly classified examples shows that the errors mostly come from cases where the number of points is significantly smaller than average, which makes the α-shape and interpolation algorithm fail since the shape is not even clearly described by the point clouds.

However, what's good about this result is that the training time per epoch significantly decreases after the application of our pre-processing algorithm due to reduction in number of points. This suggests a promising direction for faster training and more geometrically intuitive understanding of the convolutional neural network. This can be improved by designing more domain-specific techniques to identify critical points.

8.3.2 Feature Matrix

Feature matrix, as a much more simpler algorithm designed from scratch, shows much lower accuracy than the CNN-related approaches. This is reasonable since our design is still in the first iteration. Details including selection of features, designing of features and of the distance functions still need much more careful consideration and dedicate work.

However, the result is better than a dummy classifier without lots of pre-train and much better than random. This suggests that our feature matrix design does capture some essential aspects of a 3D object's shape which is useful for discrimination.

Table 8.1 Classification accuracy results

Approach	Evaluation F-score	Min per Epoch
Original *PointNet*	0.91	~ 15
Voxelized	0.81	~ 10
α-shape	0.73	~ 0.6
α-shape + 2D Curvature	0.70	~ 0.2
α-shape + 3D Curvature	0.70	~ 0.3
Feature Matrix	0.60	N/A
Feature Matrix + Shape Derivative	0.68	N/A
Feature Matrix + $x - y - z$ Shape Derivative	0.75	N/A

8.4 Future Works

- The greatest limitation of our proposed algorithms is that the level set algorithm strongly depends on the direction the object faces and the type of the objects. In the dataset we use, the objects are in classes of mostly furniture's and daily objects with significant difference in shape. And the dataset makes sure that we don't need extra rotation to make it stand straight on the "floor". In real world application, obviously we don't have these assumptions. So it's worthwhile to develop a rotation function so that we could find the correct direction, possibly based on symmetry information. Even with successful and stable rotation functions, this algorithm probably won't work if we want to discriminate objects with overall similar shapes but only small differences (like application in medical cases).
- An alternative approach than figuring out a rotation algorithm is entirely discarding usage of 2D level sets, but focusing on 3D object as a whole. This direction is blocked at the beginning of our work, since there is no existing package for calculating α-shape in Python. After it is implemented, however, it would be natural to explore calculation of curvature on surface with α-shape. Various kinds of surface curvature

and manifold curvature would provide different insights on critical point problems.

- Enrich the material of the feature matrix by adding more useful and descriptive features. Currently we only have the number of clusters, average area of cluster, and the shape derivative parameter. It would be beneficial to add more features describing not only how a single level set looks like, but also how does it "interact" with adjacent level sets. Shape derivative is a possible direction representing how the shapes of level sets are changing over time. Note that, however, increasing the number of features will require more delicate hand-crafted distance functions or more time to learn a good distance function.

Bibliography

Anand J. Kulkarni, and Suresh Chandra Satapathy. 2020. *Optimization in Machine Learning and Applications*. Springer Singapore.

Botsch, Mario, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon Mesh Processing*. AK Peters / CRC Press. URL https://hal.inria.fr/inria-00538098.

Chen, Xiaozhi, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. 2017. Multi-view 3d object detection network for autonomous driving. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ciresan, Dan C., Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. *CoRR* abs/1202.2745. URL http://arxiv.org/abs/1202.2745. 1202.2745.

Edelsbrunner, Herbert, and Ernst P. Mücke. 1994. Three-dimensional alpha shapes. *ACM Trans Graph* 13(1):43–72. doi:10.1145/174462.156635. URL https://doi.org/10.1145/174462.156635.

Fang, Y., Jin Xie, Guoxian Dai, Meng Wang, Fan Zhu, Tiantian Xu, and E. Wong. 2015. 3d deep shape descriptor. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2319–2328. doi:10.1109/CVPR.2015.7298845.

Flitton, Greg, Andre Mouton, and Toby P. Breckon. 2015. Object classification in 3d baggage security computed tomography imagery using visual codebooks. *Pattern Recognition* 48(8):2489 – 2499. doi:https://doi.org/10.1016/j.patcog.2015.02.006. URL http://www.sciencedirect.com/science/article/pii/S0031320315000540.

Fouinat, Laurent, Pierre Sabatier, Fernand David, Xavier Montet, Philippe Schoeneich, Eric Chaumillon, Jérôme Poulenard, and Fabien Arnaud. 2018.

Extended age model of lake lauvitel sediments. doi:10.1594/PANGAEA.892908. URL https://pangaea.figshare.com/articles/Extended_age_model_of_Lake_Lauvitel_sediments/11061785/1.

Gao, Y., M. Wang, D. Tao, R. Ji, and Q. Dai. 2012. 3-d object retrieval and recognition with hypergraph analysis. *IEEE Transactions on Image Processing* 21(9):4290–4303. doi:10.1109/TIP.2012.2199502.

Gao, Zan, D.Y. Wang, Y.B. Xue, Guangping Xu, H. Zhang, and Y.L Wang. 2018. 3d object recognition based on pairwise multi-view convolutional neural networks. *Journal of Visual Communication and Image Representation* 56. doi:10.1016/j.jvcir.2018.10.007.

Guerrero, Paul, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. 2017. PCPNET: learning local shape properties from raw point clouds. *CoRR* abs/1710.04954. URL http://arxiv.org/abs/1710.04954. 1710.04954.

Guo, Kan, Dongqing Zou, and Xiaowu Chen. 2015. 3d mesh labeling via deep convolutional neural networks. *ACM Trans Graph* 35(1):3:1–3:12. doi:10.1145/2835487. URL http://doi.acm.org/10.1145/2835487.

Haim, Niv, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. 2018. Surface networks via general covers. *CoRR* abs/1812.10705. URL http://arxiv.org/abs/1812.10705. 1812.10705.

Hanocka, Rana, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2018. Meshcnn: A network with an edge. *CoRR* abs/1809.05910. URL http://arxiv.org/abs/1809.05910. 1809.05910.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. 1512.03385.

Hornik, Kurt. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4(2):251 – 257. doi:https://doi.org/10.1016/0893-6080(91)90009-T. URL http://www.sciencedirect.com/science/article/pii/089360809190009T.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun ACM* 60(6):84–90. doi:10.1145/3065386. URL http://doi.acm.org/10.1145/3065386.

Li, Fei-Fei, J. Deng, and K. Li. 2010. Imagenet: Constructing a large-scale image database. *Journal of Vision - J VISION* 9:1037–1037. doi:10.1167/9.8.1037.

Ma, Ling, Rafael Sacks, Uri Kattel, and Tanya Bloch. 2018. 3d object classification using geometric features and pairwise relationships. *Computer-Aided Civil and Infrastructure Engineering* 33(2):152–164. doi:10.1111/mice.12336. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12336. https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12336.

Masci, Jonathan, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. 2015. Shapenet: Convolutional neural networks on non-euclidean manifolds. *CoRR* abs/1501.06297. URL http://arxiv.org/abs/1501.06297. 1501.06297.

Maturana, D., and S. Scherer. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 922–928. doi:10.1109/IROS.2015.7353481.

Murphy, Kevin P. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Phong, Bui Tuong. 1975. Illumination for computer generated pictures. *Commun ACM* 18(6):311–317. doi:10.1145/360825.360839. URL https://doi.org/10.1145/360825.360839.

Qi, Charles Ruizhongtai, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2016. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR* abs/1612.00593. URL http://arxiv.org/abs/1612.00593. 1612.00593.

Qi, Charles Ruizhongtai, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR* abs/1706.02413. URL http://arxiv.org/abs/1706.02413. 1706.02413.

Russell, Bryan C., Antonio Torralba, Kevin P. Murphy, and William T. Freeman. 2008. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision* 77(1):157–173. doi:10.1007/s11263-007-0090-8. URL https://doi.org/10.1007/s11263-007-0090-8.

Shi, B., S. Bai, Z. Zhou, and X. Bai. 2015. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters* 22(12):2339–2343. doi:10.1109/LSP.2015.2480802.

Shi, Shaoshuai, Xiaogang Wang, and Hongsheng Li. 2018. Pointrcnn: 3d object proposal generation and detection from point cloud. *CoRR* abs/1812.04244. URL http://arxiv.org/abs/1812.04244. 1812.04244.

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

Wang, Dominic Zeng, and Ingmar Posner. 2015. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*. Rome, Italy.

Wang, R., S. Shan, X. Chen, Q. Dai, and W. Gao. 2012. Manifold–manifold distance and its application to face recognition with image sets. *IEEE Transactions on Image Processing* 21(10):4466–4479. doi:10.1109/TIP.2012.2206039.

Wu, Zhirong, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2014. 3d shapenets: A deep representation for volumetric shapes. 1406.5670.

Yan, Xinchen, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. 2016. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. 1612.00814.

Zhao, Wenbo, Xianming Liu, Yongsen Zhao, Xiaopeng Fan, and Debin Zhao. 2019. Normalnet: Learning based guided normal filtering for mesh denoising. 1903.04015.

Zheng, Yefeng, Bogdan Georgescu, and Dorin Comaniciu. 2009. Marginal space learning for efficient detection of 2d/3d anatomical structures in medical images. In *Information Processing in Medical Imaging*, eds. Jerry L. Prince, Dzung L. Pham, and Kyle J. Myers, 411–422. Berlin, Heidelberg: Springer Berlin Heidelberg.