Exploring Generative 3D Shapes Using Autoencoder Networks*

Yanwen Xu
yxu83@ucsc.edu[1]

[1]University of California, Santa Cruz

1 Abstract

Three-dimensional (3D) mesh representation is one of the most fundamental topics in the field of computer graphics and computer vision. We re-implemented a recent study that introduced a novel, simple approach for effectively normalizing similarly shaped 3D meshes for machine learning applications. The parameterization algorithm can construct a triangle mesh efficiently and robustly with a consistent topology that is compactly parameterized as a height map, regardless of the mesh's irregular format or non-uniform underlying structure. We leveraged the power of an autoencoder to explore the latent space behind the 3D shapes in the same category. Furthermore, to overcome the limitation of previous works, we improved the parameterization algorithm with a novel PolyCube maps-based method that is capable of covering a broader range of shapes, including concave shapes.

2 Introduction

3D machine learning is an interdisciplinary field that fuses computer vision, computer graphics, and machine learning. In recent years, a tremendous amount of progress has been made in the field of 3D machine learning. Various applications, such as classification, style transfer, and procedural content generation, have been introduced. We are interested in seeing how procedural content generation can benefit from the advances in deep learning.

Learning 3D shapes is primarily about finding a low-dimensional manifold in the high-dimensional space of the set of shapes. To synthesize new shapes, we need to construct a forward and backward mapping from the low-dimensional manifold to the high-dimensional space. The manifold encodes the features of the shape in the same category and can, therefore, be decoded and interpolated to produce new shapes absent from the sample space.

Triangle meshes are the most commonly used surface representation in computer graphics. Mesh data of 3D shapes is a collection of vertices, edges, and faces. Mesh data has the properties of complexity and irregularity. It is difficult to apply machine learning to unstructured meshes as their underlying structure can be very complex and non-uniform. A recent SIGGRAPH paper [Umetani, 2017] proposed a novel approach to parameterize unstructured shapes. The fundamental idea is to store 3D shapes as height fields with respect to predefined normals via a shrink-wrap algorithm. However, this approach has a few limitations: it only handles one class of shape and nearly convex shapes, for instance, a torus.

*This is originally a 2017 paper from SIGGRAPH Asia by Nobuyuki Umetani. My term project is primarily based on this paper, but we have re-implemented the experiment. Unless explicitly stated, the figures, algorithm, data, experiments, and results are part of my work.

	fix	compact	linear	generic
our parameterization	✓	✓	✓	✓
Umetani's parameterization	✓	✓	✓	✗
triangle mesh	✗	✓	✓	✓
multi-view projection	✓	✗	✗	✓
voxel	✓	✗	✗	✓
point cloud	✓	✗	✗	✓

Table 1: Comparison of approaches in machine learning of 3D shapes

3 Related Work

There are several popular types of data for 3D shape representation, including polygonal mesh, multi-view projection, volumetric grid (voxel) [Li et al., 2016], and point cloud [Qi et al., 2017]. Although polygon meshes are the most popular, due to their irregular format, most researchers transform such data to regular 3D voxel grids. Table 1 compares some common parameterization approaches of 3D shapes with our approach. When evaluating parameterization problems for machine learning, we consider the following factors:

- **fit**: Whether the input and output vectors have a fixed dimension
- **compact**: Whether the input and output vectors are as small as possible (no redundancy)
- **linear**: Whether the input and output vectors are linear or complicated
- **generic**[1]: Whether the approach is able to handle as many types of shapes as possible

The voxel representation represents values on a regular axis-aligned grid in 3D space. In practice, voxel representation is implemented in fixed-dimension vectors (e.g., three-dimensional array). However, the voxel approach has a limited resolution since it requires one more dimension of information than a 2D bitmap. Besides, the voxel approach for 3D models still has the data redundancy issue similar to the pixel approach for bitmap images.

The point cloud is a set of data points in space but does not require an axis-aligned grid. The point cloud approach is advanced and tends to be very computationally intensive.

We study the approach by [Umetani, 2017], which focuses on surface data representation. However, the approach still has limitations, as described in Section 2.

4 Data

We used 1243 normalized car models from Umetani that were published with the paper [Umetani, 2017]. The normalized models have been manually removed sharp non-convex details such as tires, side-mirrors, spoilers, and antennas from the models. The original data came from the ShapeNet[2] [Chang et al., 2015] Data set with $3.5k$ shapes under the "car" category. These cars include various styles such as sedans, wagons, pickup trucks, sports cars, and classic cars. However, the original data was not normalized, and was human labour intensive (takes 5 min to process each model). Thus we decided to use the data came with Umetani. Our final data set has 1230 models, as some of the models were removed due to data corruption after processing. We use an 80/20% train/test split, for 984 training models and 246 testing/evaluation models.

[1]This factor was not from the original umetani's summary.

[2]ShepeNet includes 3 Million+ models and 4K+ categories. A Data set that is large in scale well organized and richly annotated.

5 Parameterization of 3D Shapes

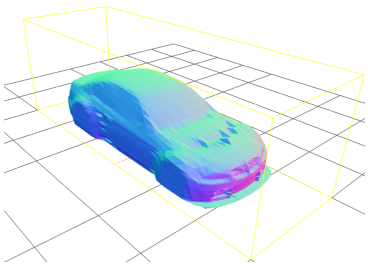

Figure 1: **The $2 \times 2 \times 6$ m bounding box that enclose all normalized training data.**

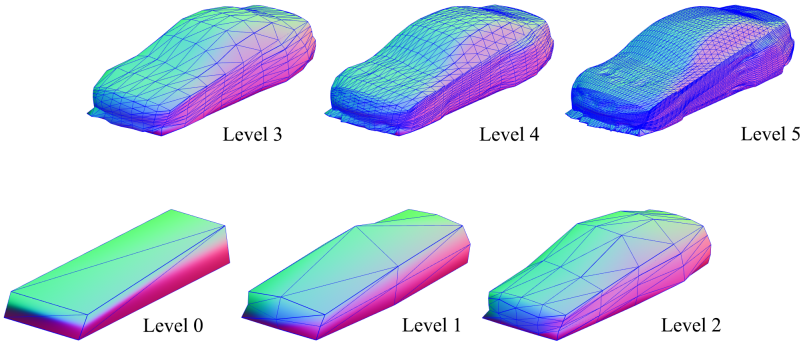

Figure 2: **Shrink wrapping paramerization. Example of parameterization in different subdivision levels.**

In this section, we present an effective and robust algorithm to consistently parameterize the 3D shapes for machine learning applications, i.e., produce meshes that all have a uniform topology. The algorithm is primarily based on a shrink-wrapping algorithm, in which we recursively subdivide a cube N times and hierarchically project the source mesh's vertices onto the target mesh.

5.1 Bounding Box

The car meshes are normalized to fit at-scale within a $2 \times 2 \times 6$ m bounding box (Figure 1). We move the box's eight corner points onto a best-fit location on the car's surface, which in our case was provided as part of the training data.

5.2 Spherical Normal

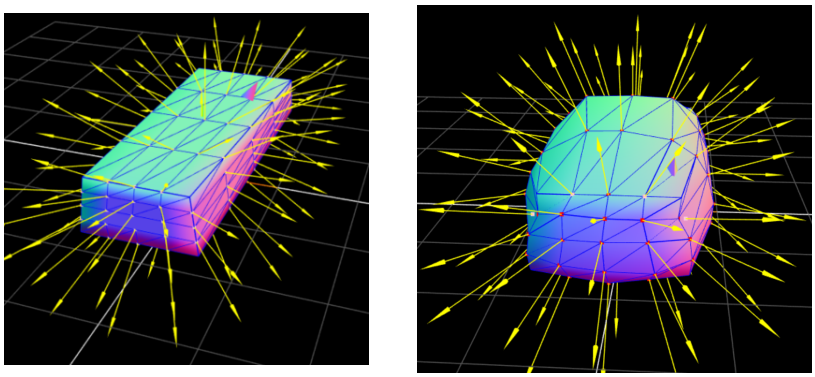

Figure 3: **A snapshot of projecting the vertices along the spherical normal $\vec{d}$ on a level-2 (See Figure 2) subdivided mesh. (left) Unprojected. (right) Projected.**

We have predefined a spherical normal $\vec{d}$ that is omnidirectional (Figure 3) at the origin in

the space. This normal values is required in the vertex projection stage as described in Section 5.3.

5.3 Subdivision and Projection

We repeatedly subdivide the cube mesh by adding vertices at the center of each edge and triangle face. During each subdivision step, new vertices are projected along the predefined spherical normal to hit on the mesh's surface (Figure 2). The normal of the newly added vertex is determined by up to two ray-casts outwards or inwards to the model's surface. Each vertex thus only has one degree of freedom and can be represented uniquely as a single scalar value.

The following is the pseudocode[3] that we used to subdivide and project mesh and is different from [Umetani, 2017]. Let E denote the set of all Edges on the initial cube; let $\vec{d_w}$ denote the predefined normal on the vertex w,

Algorithm 1 Our Shrink-wrap Algorithm

for $1 \dots 5$ **do**
  $E\prime \leftarrow \emptyset$
  for all $\{u, v\} \in E$ **do**
    $w \leftarrow (u + v)/2$
    $\vec{d_w} \leftarrow (\vec{d_u} + \vec{d_v})/2$
    Shoot a *Ray* along $\vec{d_w}$ from w, record height h.
    if *Ray* missed **then**
      Shoot a *Ray* along $-\vec{d_w}$ from w, record height h.
    $E\prime \leftarrow E\prime \cup \{\{u, w\}, \{w, v\}\}$
    $E \leftarrow E \setminus \{\{u, v\}\}$
  $E \leftarrow E \cup E\prime$

For time complexity, we must shoot a ray-cast from each newly added vertex. The naive ray-cast algorithm checks the ray intersection against every face of the mesh, which takes $O(n^2)$; and we perform a ray-cast on n vertices, yielding a total of $O(n^3)$ time complexity. However, using an *Octree* can significantly reduce the complexity of the ray-cast down to $O(n \log n)$. An *Octree* is a tree data structure used to partition a three-dimensional space by recursively subdividing it into eight octants.

5.4 Data Storage and Reconstruction

The models are shrink-wrapped with 5 subdivision levels, yielding 6138 projected vertices and 8 corner points. Thus, the input parameter has a total of $6138 + 3 \times 8 = 6162$ dimensions. The eight corner vertices and vertex offset scalars are stored as a flat array.

The set of projection heights H_s for all the surface points indicates how much the point are moved toward the normal direction, encoding the shape at the coarsest level together with the positions of the corner points $\hat{P}_c$. Thus, the array is $\{\hat{P}_c, H_s, H^1_{ss}, H^2_{ss}, \dots\}$.

This array is what we use as inputs/outputs to the neural network. The meshes are reconstructed back by re-applying the shrink-wrap algorithm but with the passed in corners and offsets instead of ray-casts to a source mesh.

[3] This is our naive $O(n^3)$ approach. The original algorithm from the paper [Umetani, 2017] was much clever and only had an complexity of $O(n)$.

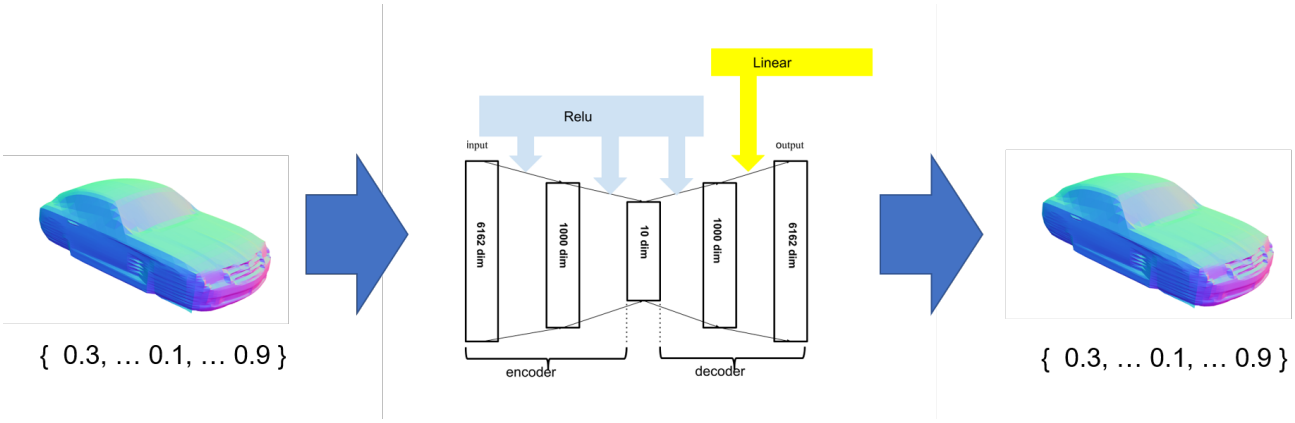

Figure 4: **(center) Configuration of our autoencoder network. (left/right) Sample input/output from this encoder-decoder.**

6 Machine Learning

So far, we have explained our parameterization framework, which encodes shape and field in fixed dimensional vectors. In this section, we explain how we have leveraged the power of deep learning, in order to approximate the output parameter vector from the input parameter vector. In machine learning problems, there are often too many factors describing the input. These factors are basically variables, called features, and they are often redundant. For example, in 2D image classification, a $256 * 256$ pixel image with an RGBA color space has a total of $256 * 256 * 4 = 262144$ features to learn, while a lot of the pixels share semantic information with their neighbors. The greater the number of features, the harder it is for the neural network to work. However, *dimension reduction* is the process of reducing the number of random variables, under certain circumstances.

6.1 Autoencoder

A prior study by Zhu et al. [Zhu et al., 2016] has shown the capability of projecting 3D shapes into 2D space using an autoencoder for feature learning on the 2D images. An autoencoder is a type of neural network used to learn efficient data codings through dimensionality reduction in an unsupervised manner. The neural network is trained to ignore signal "noise." This characteristic of an autoencoder is practical in our case since we desire to learn the low-dimensional encoding of 3D shapes.

The mathematical definition for a autoencoder framework autoencoder from [Baldi, 2012] is defined as the following:

- Let $\mathbb{F}, \mathbb{G}$ be the input set and the encoded set, respectively.
- Let $n, p \in \mathbb{N}$, and $0 < p < n$. Where n, p are the input and encoded dimension, respectively.
- Let $\mathcal{A}$ be the class of functions from $\mathbb{G}^p$ to $\mathbb{F}^n$.
- Let $\mathcal{B}$ be the class of functions from $\mathbb{F}^n$ to $\mathbb{G}^p$.
- Let $\{x_1, \ldots, x_m\}$ be the set of m traning vectors in $\mathbb{F}^n$.
- Let Δ be the distortion function defined over $\mathbb{F}^n$.

For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder transorms an input vector $x \in \mathbb{F}^n$ into an output vector $A \circ B(x) \in \mathbb{F}^n$. The corresponding autoencoder problem is then to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$ that minimize the overall distortion function:

$$\min E(A, B) = \min_{A,B} \sum_{t=1}^{m} \Delta(A \circ B(x_t), x_t)) \tag{1}$$

And in our experiments, we used Mean-Squared-Error (MSE) as our distortion function, thus $\Delta = \text{MSE}$. Let Y be an vector with n dimension, where:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2}$$

Combining Equaition 1 and 2 and then cleanup,

$$\min E(A, B) = \arg\min_{A,B} \|A \circ B(x_t) - x_t\|^2 \tag{3}$$

Obviously, if the training was done properly with sufficient amount of data, the distortion function will be minimized as small as possible. Thus, the outcome implies that the input features will be as close as possible to the original input.

The transition from $\mathbb{G}^p$ to $\mathbb{F}^n$ serves as the *encoder* portion of the autoencoder. The transition from $\mathbb{F}^n$ to $\mathbb{G}^p$ serves as the *decoder* portion of the autoencoder. The encoded data $\mathbb{G}$, often stored in the middle layer in the autoencoder networks, is also called the *latent space*. This encoded data is hidden in the latent space, and can only be obtained through fitting the training data into the neural network.

6.2 Training

We used an underlying feed-forward autoencoder network (Figure 4) as a simple generative model to extract the manifold of shapes in the same category. Our model has five layers, where the first and the second layer is the *encoder*, and the fourth and fifth layers is the *decoder*. The input dimension is 6162 as stated in Section 5.4. We experimentally set the second layer with 1000 dimension, and the middle layer (the latent space) with 10 variables. The model's layout is mostly symmetric and was built in Keras/TensorFlow.

Since the vertex offsets can be negative and are not strictly normalized, we used a linear activation function for the last layer and a Leaky ReLU[4] with $\alpha = 0.1$. The Leaky ReLu allows a small gradient when the unit is not active, where the α value represents that small gradient or the "learning" rate. We set the dropoff rate to 0.2 for all the other layers to avoid overfitting problem during the training process.

The Leaky ReLU activation function is defined as:

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \text{ where } \alpha = 0.1$$

And the activation function Identity is defined as:

$$f(x) = x$$

[4]In the original paper, the author uses Sigmoid function instead of ReLU. However, Sigmoid function did not preform well in our experiments. And with the help from Molly Zhang, we gave a try to Leaky ReLU.

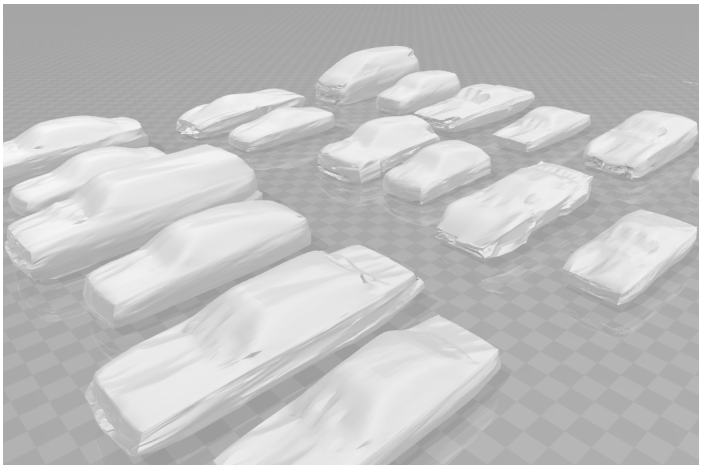

Figure 5: Some sample input and outputs from our autoencoder. The Input models are indicated as the larger models, and the output models are the smaller models next to them. The output become smaller in scale because it was normalized by the *decoder*. It is not a problem because we can always scale the model back.

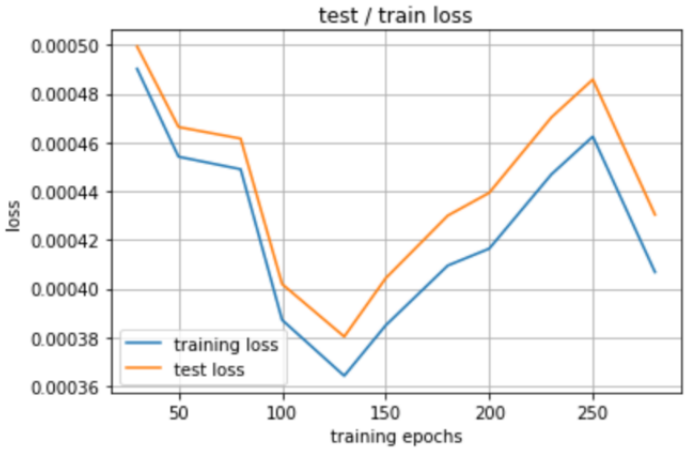

Figure 6: Training result from TensorFlow. 125 epoch produces the best outcome.

The model is trained using MSE (Equation 2), an Adam optimizer, and a batch size of 50. An Adam optimizer is a effective replacement for the classical stochastic gradient descent optimizer. Note that it does not matter in our experiment which optimizer to use. We pick Adam because it is easy to configure, where the default configuration parameters do well on most problems. Furthermore, we used the trained model to explore and synthesize a variety of shapes.

6.3 Evaluation

We used various metrics to evaluate the effectiveness of our autoencoder model:

- Train/test loss as in Figure 6. The smaller the better.
- Visual inspection of train/test samples for the autoencoder as in Figure 5.
- Visual inspection of random models sampled from our latent space as in Figure 8.

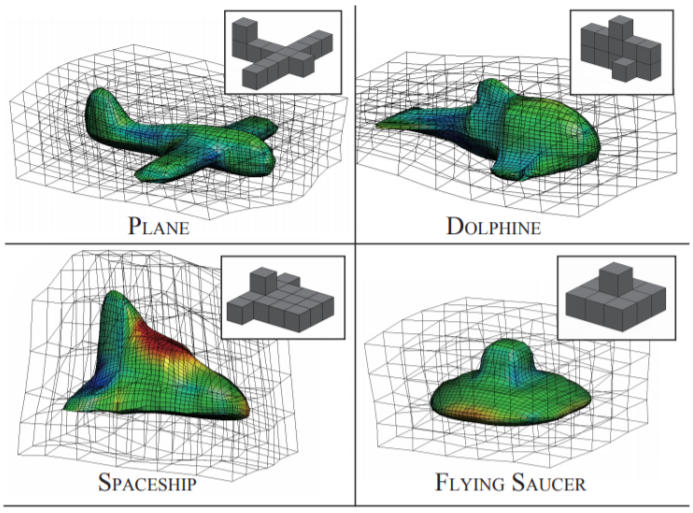

Figure 7: Examples of Applying PolyCube 3D shape and field paramterization for a various shapes. This firgures is from [Umetani and Bickel, 2018].

- Examining how well features can be added or subtracted to produce new meshes.

As a result, our trained model is capable of learning the latent space behind the training and testing data effectively.

7 PolyCube

In the previous work by [Umetani, 2017], the cube-based shrink-wrap technique is limited to car-like, almost concave, shapes. We can parameterize a wider variety of shapes, using a PolyCube that approximates the shape. A PolyCube is a set of axis-aligned unit cubes that are connected face-to-face. Based on [Umetani and Bickel, 2018], polycube mapping is suitable to parameterizing a wide range of 3D shapes.

As shown in Figure 7, during the parameterization process, we are able to start from a polycube instead of the naive bounding box. The approach is compatible with our shrink-wrap algorithm (See Algorithm 1). There is no overhead cost to replace the starting cube with a polycube.

8 Conclusions

We have shown that vertex data can be used effectively with a feed-forward neural network when the inputs have a uniform topology and orientation and similar shapes. We learned that shapes need to be represented by fixed dimensional vectors/tensors. Triangle meshes are not suitable for machine learning because 3D mesh data have complex structures for their topology, and the number of vertices is inconsistent.

To address these problems, we re-implemented a novel and robust algorithm that normalized unstructured triangles mesh into a consistent topology. The algorithm parametrized the surface data as a height map, which is represented by a single dimensional tensor.

We found a mapping from 3D shapes into a lower-dimension manifold via the autoencoder network and used it for feature learning on the lower-dimension manifold. The latent space

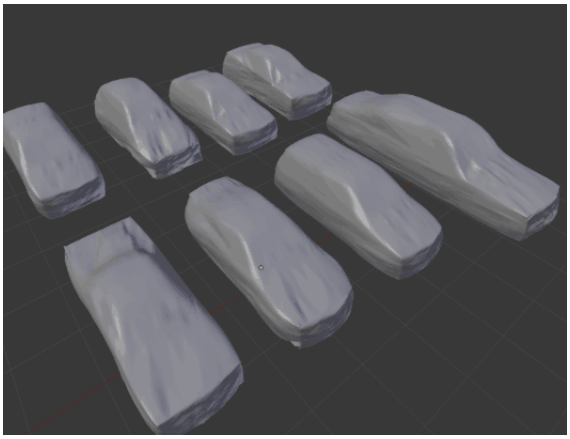

Figure 8: Several latent models in perspective. The network was trained with 125 epoch. The latent space was learned from the 984 training models.

appears to learn the shapes of various 3D models and linearly recombine them; thus, one could synthesize new shapes by randomly sampling through the latent space.

For future works, there still is a fair amount of work that could be done with this shrink-wrap algorithm. Besides, applying more advanced machine learning models, such as Generative Adversarial Network (GAN) or Variational Autoencoder (VAE) would be an obvious next step. Again, with proper parametrization algorithms, GAN and VAE could be applied as input data with uniform size and consistent structure.

8.1 Open Problems

Though the progress of the field of 3D machine learning is significant within the past few years, many important questions are still open:

- The testbed for evaluating the current 3D machine learning algorithms are still not sufficient. For most of the time, We still have to manually visual inspect the result (as in Section 6.3).
- We still do not have sufficient ability to discover structures from 3D shape collections. As human can perceive shapes easily with certain abstractions of its structure, there does not yet exist a way for computer to perceive sophisticated object.

9 Acknowledgement

This research was initially a collective work with Seiji Emory and Zhewei Wang for an AI course project at UCSC. We thank our TA Molly Zhang for the assistance with TensorFlow setup. We thank Professor Phokion G. Kolaitis and TA Kevin Bowden for reviewing this paper and gave insightful feedbacks.

References

[Baldi, 2012] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49.

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

[Li et al., 2016] Li, Y., Pirk, S., Su, H., Qi, C. R., and Guibas, L. J. (2016). Fpnn: Field probing neural networks for 3d data. In *Advances in Neural Information Processing Systems*, pages 307–315.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660.

[Umetani, 2017] Umetani, N. (2017). Exploring generative 3D shapes using autoencoder networks. In *SIGGRAPH Asia 2017 Technical Briefs*, page 24. ACM.

[Umetani and Bickel, 2018] Umetani, N. and Bickel, B. (2018). Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics (TOG)*, 37(4):89.

[Zhu et al., 2016] Zhu, Z., Wang, X., Bai, S., Yao, C., and Bai, X. (2016). Deep learning representation using autoencoder for 3d shape retrieval. *Neurocomputing*, 204:41–50.