Latent Shape Space Analysis of Point Cloud Data [Final Report]

Samaksh Goyal
Stanford University
sagoyal@stanford.edu

Sidhika Balachandar
Stanford University
sidhikab@stanford.edu

Finsam Samson
Stanford University
finsam@stanford.edu

Abstract

Recently, 3D point clouds have become the preferred method for several geometric modeling and computer vision problems. They provide an accurate yet sparse sampling of the physical world and provide easy integration to traditional image processing pipelines. They have also been rigorously explored in the deep learning community. Yet, a fundamental question that remains is to understand the latent space the model is learning for a given point cloud. In this paper we investigate this question by evaluating the latent space of an autoencoder that was trained to reconstruct geometric categories with varying degrees of structural and topological similarity. As found by others in the field, we see that permutation invariance is a critical part of modeling point cloud data.

1. Introduction

3D representations of real-life objects are important for problems involving computer vision, robotics, medicine, augmented reality, and virtual reality. 3D data can be represented as point clouds, meshes, or implicits. In our project we choose to use a 3D point cloud dataset. Point clouds are a scattering of 3D points on the surface of an object and are created with depth sensor tools such as LIDAR or RGB-D cameras (see Figure 1). Point clouds are a popular form of representing 3D data particularly for deep learning tasks because they are the simplest representation of 3D objects. Point clouds only contain information about points in 3D space, they do not contain any information about connectivity between points. An alternative representation such as voxel based methods have been attempted in the past but the resolution of voxels can change depending on the sampling pattern of a scanner so this is not used in practice.

In our project we aim to investigate the latent space of an autoencoder model for 3D point cloud data. Within the deep learning field, there has been a push towards evaluating the interpretability of models and making models more interpretable. Within the sub field of deep learning for 3D data, interpretability is especially important for problems such as shape generation. For a shape generation problem you are provided a reference object and the goal is to produce a new different object. Suppose we want to generate a couch based off of a reference chair object. We can use an autoencoder model to create a latent space for 3D point cloud representations of chairs. Suppose the latent space is structured such that the first dimension represents the height of the chair legs and the second dimension represents the width of the chair. Then to create a couch, we would want to apply transformations to our encoded input chair such that we decrease the first dimension (decrease the height of the chair legs) and increase the second dimension (increase the width of the chair). Thus, having a well structured and interpretable latent space is crucial for the shape generation problem. Interpretability is also beneficial for other 3D data based deep learning problems as it allows researchers to identify when autoencoding models fail and how to improve these models.

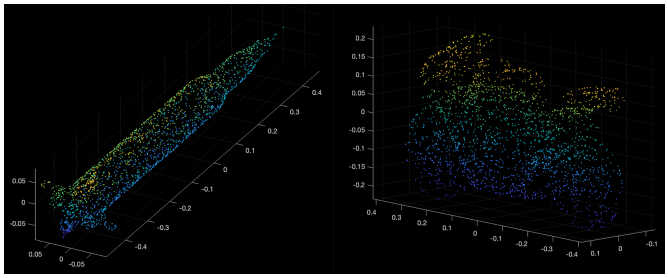

Figure 1. Point cloud representation of a rocket (left) and a motorcycle (right).

2. Related Work

Point Cloud Representations: Fan et al. includes foundational work introducing representation of 3D data as point clouds, proposing an architecture to predict plausible 3D point cloud reconstructions from single image data [5]. This model outperformed state-of-the art methods at that time, most of which did not represent 3D data as point clouds.

Qi et al. includes also seminal work on learning from 3D point cloud data, where they randomly sample points to learn features of the 3D point clouds, aggregating these for improved performance on benchmark tasks [9]. We also

plan to use this method as well as other data augmentation methods introduced in this paper for our project.

Generative Models: Achlioptas et al. utilized an autoencoder model to learn a generalizable latent space, then explored several generative models using novel metrics to identify those best suited for learning representations of 3D point clouds, then generating them [1]. We are particularly interested in the autoencoder model used to learn representations, and we use their 3D point cloud data processing pipeline in our baseline experiments.

Aumentado-Armstrong et al. introduces an additional geometric disentanglement step to earlier generative models, creating latent subgroups from a learned latent space to further represent rotation, intrinsics, and extrinsics [2].

Continuous Representations: Chen et al. used a implicit field decoder which is a binary classifier determining whether a point is within the 3D shape or not given an implicit field, eventually generating a continuous surface [3]. Notably, they also generate 3D shape interpolations from learned latent spaces of their models.

Mescheder et al. also generates a continuous representation through the use of occupancy networks, which also train a binary classifier identify points within a 3D space as either within the 3D object or outside of it [8]. They were also able to generate new plausible 3D shapes by alternative sampling of their learned latent space.

Wang and Xu et al. approach continuous representations similarly, but calculate Continuous Distance Functions rather than a binary classification for each candidate point [10]. From this calculated distance, a continuous representation is generated.

Chibane et al. generated continuous implicit function representations specifically from 3D point cloud and 3D voxel data [4]. They did so in the domain of human posture and poses.

Feature Learning from Point Clouds: Li et al. introduces a novel method to spatially convolve over points in a 3D point cloud to better learn features, analogous to convolutional layers for images and videos [7]. Previously, attempts to convolve over point cloud data without the transformation used in this paper were not successful, making this work a notable advancement in feature learning for 3D point cloud data.

Li et al. builds a Self-Organizing Map with which a model could be trained to encode any 3D point cloud as a single feature vector [6]. The latent representation learned with model was then benchmarked on classification and segmentation tasks.

3. Methods

3.1. Autoencoder

We use an autoencoder to create the latent space of 3D point cloud data. An autoencoder is a machine learning model used to learn efficient data encodings. It is made up of two parts: an encoder, ϕ, and a decoder, ψ, such that

$$\phi : \mathcal{X} \to \mathcal{F} \tag{1}$$

$$\psi : \mathcal{F} \to \mathcal{X} \tag{2}$$

$\mathcal{X}$ represents the set of all 3D point clouds and $\mathcal{F}$ represents the set of all latent representations of elements in $\mathcal{X}$. Then the encoder can be thought of as a function mapping an input 3D point cloud to a latent representation of the input. The decoder can be thought of as a function mapping a latent representation to it's 3D point cloud. Since an autoencoder feeds the output of the encoder to the decoder, ideally the output of an autoencoder model is equivalent to the 3D point cloud fed in as input.

For the baseline autoencoder, both the encoder and decoder are represented by a multilayer perceptron with a leaky ReLU nonlinearity and batch norm applied after each layer. In order to obtain a dimensionality reduction between the encoder input and the encoder output, we create the MLP with linear layers that decrease the dimension size. Similarly, for the decoder, the MLP is created with linear layers that increase the dimension size.

In order to investigate the latent space of the autoencoder, we only need to run the encoder model on an input point cloud. However, the decoder model is important during the training process. During training, we run the encoder and decoder model on each point cloud in the training dataset. Since ideally the output of the decoder exactly matches the input to the encoder, the loss represents the mismatch between the input and output. For the baseline model, we calculate the MSE loss between the input point cloud and the output point cloud.

3.2. Chamfer Loss

One of the quirks of point cloud data is that it has no ordering. In other words, point cloud data is permutation invariant, the points can be reordered or permuted and they will still represent the same object. This impacts both the loss and the autoencoder model. We will begin by talking about its impact on the loss. The MSE loss calculates the mean squared error between each point in the input point cloud and the output point cloud. In particular, the ith point in the input point cloud is compared to the ith point in the output point cloud. Since point cloud data is permutation invariant, it is undesirable to enforce this pairing between input and output points. Thus for our second model, instead of using MSE loss, we use a loss function based on the Chamfer Distance.

The Chamfer distance (CD), like MSE loss, calculates the error between two sets of points. CD measures the squared distance between each point in one set to its nearest neighbor in the other set. The equation for the Chamfer-loss is provided below

$$d_{CH}(S_1, S_2) = \frac{1}{N} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \frac{1}{N} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2 \quad (3)$$

In simpler terms, CD takes a point x in the input point cloud. It then tries to find the point y in the output point cloud that is most likely to match the input point. Note that MSE loss chooses the matching output point to be the point at the same index as the input point. CD instead searches over all points in the output point cloud and chooses the point closest in distance to the input point. After pairing the input point with an output point, the error is then calculated using the L2 norm, similar to the MSE loss. Also similar to the MSE loss, in the end in order to find the loss with respect to the entire input point cloud we calculate the mean error across all points in the input point cloud. This represents the first summation term in equation 3.

The second summation term repeats this process but instead takes each point in the output point cloud and finds the corresponding closest point in the input point cloud. For MSE loss, since the input and output points are paired by index, the mapping is equivalent whether evaluated with respect to the input or the output points. However, for CD, there is no guarantee that we will have the same pairings. For example, suppose we have two input points and two output points

$$ip_0 = (0, 0, 0)$$
$$ip_1 = (0, 0, 0)$$
$$op_0 = (0, 0, 0)$$
$$op_1 = (100, 100, 100)$$

There is no restriction to prevent the two input points from mapping to the same output point op_0. However when calculating the loss with respect to the output points, op_0 can only match with one point ip_0 or ip_1. Moreover due to op_1, the loss with respect to the output is much larger than the loss with respect to the input. Thus we must calculate the loss with respect to the input points and the loss with respect to the output points. Our final loss is the sum of these two terms.

3.3. Pointnet Autoencoder Model

As stated before, the point cloud's permutation invariant property also affects the autoencoder model. Consider some input point cloud ip. Let ip' represent a permutation of the points in ip. Since ip and ip' represent the same image, we would want the autoencoder to output the same point cloud for both inputs ip and ip'. However, our baseline model, which uses an MLP, does not provide this permutation invariant property. As an example, consider a simple linear model that applies the weight $(0, 1, 2, 3, 4, 5)$ to a flattened input point cloud to produce an output point cloud. Suppose we have two input points

$$ip_0 = (0, 0, 0)$$
$$ip_1 = (0, 0, 1)$$

Then our corresponding output points are

$$op_0 = (0, 0, 0)$$
$$op_1 = (0, 0, 5)$$

However, if we permute our input to create input points

$$ip'_0 = (0, 0, 1)$$
$$ip'_1 = (0, 0, 0)$$

Then the corresponding output points are

$$op'_0 = (0, 0, 2)$$
$$op'_1 = (0, 0, 0)$$

This generalizes from a simple linear model to an MLP. For our next model we structure the encoder into two steps. We first transform the input point cloud into a permutation invariant representation of the point cloud as described by [1] and [9]. Next we run an MLP on the permutation invariant representation. The decoder model is still a simple MLP.

We will now discuss the first step of the encoder model which creates a permutation invariant representation of the input point cloud. In order to make a model invariant to input permutation, we can use any of the following three strategies: 1) sort the input into a canonical order; 2) augment the training data by all kinds of permutations; 3) use a simple symmetric function to aggregate the information from each point. Strategy 1 requires us to preprocess our data such that each point cloud matches the canonical representation. Strategy 2 increases the training data set size and thus increases the time it takes to train the model. Strategy 3 does not require preprocessing and it does not increase training time, thus it is the option we will use.

Strategy 3 applies a symmetric function on the input point cloud in order to create a permutation invariant representation of the point cloud. A symmetric function is a function that is permutation invariant. This means that it is invariant to the order of the inputs to the function. For example, + and $\times$ are invariant functions since they are commutative operators. We will approximate a symmetric function f using functions g and h in the following way:

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n)) \quad (4)$$

In equation 4, $\{x_1, \ldots, x_n\}$ represents the input set of n points and each $x_i \in \mathbb{R}^3$. h will be approximated as a Conv1d function and $h : \mathbb{R}^3 \to \mathbb{R}^d$. We will call $\mathbb{R}^d$ the channel dimension. Then g will be approximated as a max function over the channel dimension and $g : \mathbb{R}^{n \times d} \to \mathbb{R}^d$.

We will start by explaining how the max function represents a symmetric function. Suppose h represents the identity function. Let our input point cloud be

$$ip_0 = (1, 10, 2)$$

$$ip_1 = (2, 0, 3)$$

After applying g, we get

$$g(ip_0) = (1, 10, 2)$$

$$g(ip_1) = (2, 0, 3)$$

After applying h, we get

$$h(g(ip_0), g(ip_1)) = (1, 0, 2)$$

If we permute our input such that

$$ip'_0 = (2, 0, 3)$$

$$ip'_1 = (1, 10, 2)$$

After applying g, we get

$$g(ip'_0) = (2, 0, 3)$$

$$g(ip'_1) = (1, 10, 2)$$

After applying h, we get

$$h(g(ip'_0), g(ip'_1)) = (1, 0, 2)$$

When g is the identity function, h outputs a single permutation invariant point that represents the min x, min y, and min z coordinates. Thus we can see that the composition of g and h is permutation invariant.

We will now discuss why we model g with a Conv1d function. Suppose we start with a point cloud with 1000 data points. If g is the identity function, the max function reduces the dimensionality from 1000×3 to 3 and thus throws out a lot of valuable data. Thus we must first apply a dimensionality increasing function to each of the points before taking the max. We use the Conv1d function to increase the dimensionality. We choose a Conv1d funtion over a simple linear layer because the Conv1d uses fewer parameters.

4. Dataset

Our dataset consists of uniformally sampled point clouds from the the ShapeNetCore database. ShapeNetCore is a database of objects represented as point clouds. It consists of objects from 57 different categories including: car, airplane, table, skateboard, etc. Each point cloud is made up of 2048 points. We use the preprocessed data available on the GitHub repository of [1]. Some examples of the point cloud data are provided in Figure 1 below.

In addition to being permutation invariant, point clouds are also rotationally and translationally invariant (SE (3) invariant). This means that a rotated and translated point cloud still represents the same object as the initial point cloud. Similar to permutation invariance, there are three strategies to handle SE (3) invariance: 1) preprocess the input into a canonical orientation and position; 2) augment the training data by all kinds of rotations and translations; 3) use a autoencoder model that enforces SE (3) invariance. Strategy 1 requires us to preprocess our data such that each point cloud matches the canonical representation. Strategy 2 increases the training data set size and thus increases the time it takes to train the model. Strategy 3 does not require preprocessing and it does not increase training time, thus ideally we would use this approach. Existing SE (3) invariant models include TFN networks or 3D Steerable CNNs. However for simplicity in this work we use strategy 1. Each point cloud in the data set is rotated such that it is axis aligned and within a unit sphere.

We train our models on point clouds from two different categories. We use a training/validation/test split of 0.90/0.09/0.01. Note that the testing dataset size is small because our focus is to investigate the latent space not on getting high accuracy. In fact, most of our latent space analysis is done using examples from the training data set.

5. Experiments

5.1. Autoencoder Architecture

The autoencoder model we used is based of the model in the paper by [1]. Their model is structured as follows: The encoder contains 1D convolutions with batch normalization and ReLUs, with kernel size of 1 and stride of 1. This allows us to treat each 3D point independently. The decoder contains fully connected layers with FC-ReLUs. An Adam optimizer is used with an initial learning rate of 0.0005, β_1 of 0.9 and a batch size of 50 to train all AEs.

We created three different models:

1) The baseline autoencoder architecture has a bottleneck layer of 128 (which was found to be the most generalizable in [1]). Our layers for the encoder take as input flattened 2048×3 points in the point cloud, followed by layers of size 4096, 2048, 1024, 128, and the layers for the decoder are the same in reverse order. We also add batch

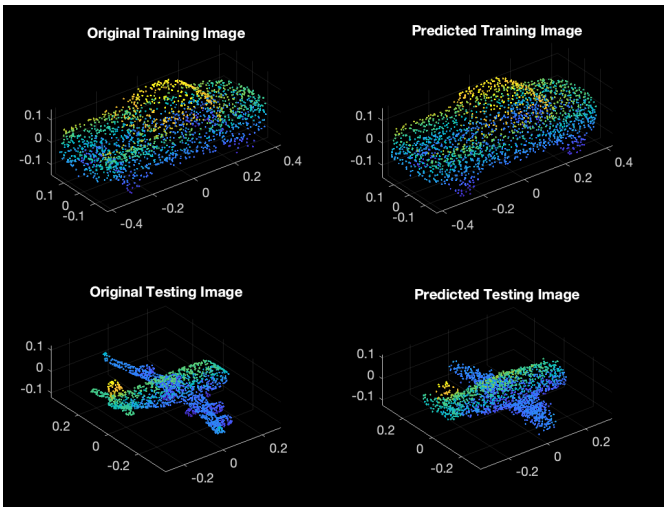

Figure 2. Comparison of decoded training vs testing point cloud using pointnet model.

normalization and leaky ReLU to ensure back propagation of gradients. The baseline model was also trained using a mean square error (MSE) loss.

2) In our next model we used the same architecture but substitute the MSE loss with the Chamfer loss.

3) For our final model, the architecture we used was inspired by [11] to incorporate permutation invariance. For our encoder we used three sets of 1 dimensional convolutions, batchnorm, and leaky relu to increase the dimension of each point from 3 (x,y, z) to 512. Then we took a max over all the points and used 2 fully connected layer to produce the same hidden dimension size as the baseline, 128. For the decoder we use the same architecture as our baseline. We also use Chamfer loss.

6. Results and Discussion

6.1. AE Validation

Recall in our milestone, we **only** create a model for shapes corresponding to different types of cars. While evaluating the model we found that the model had only partially learned to represent the overall shape of the car. For example for sports cars it learned to create a shallow elongated frame, and for cars like suv's it learned to vertically clump the set of points. This held for both testing and training car point clouds. Furthermore in our baseline model during testing the blob shape loses almost all of the key characteristics that allows a human to visually classify it as a car.

In contrast, our pointnet based architecture that uses Chamfer loss and permutation invariance provides significant visual improvement. To best reflect the advancements we also train our model on two different categories: cars and planes. This makes the learning objective for the model more difficult as it must learn to distinguish data points representing different categories.

Model	Train Loss	Val Loss
MLP + MSE	0.0059	0.0330
MLP + Chamfer	0.6407	1.8889
Pointnet + Chamfer	0.7740	1.0683

Table 1. Losses for three different models

We see that in figure 2 when we visualized the results for a car point cloud that was present in the train set, we are able to almost fully recover the original shape. Most interestingly we conserve small domain specific details like the curvature of the wheels of the car, however we lose the the rigid structure of the truck bed which now has a softer profile. As a test we try to visualize an airplane and the model is able to get a reasonable approximation that matches the general frame and span of the wings. However during testing we lose many details like the nose angle and the 4 engines beneath the wings. We can see that the autoencoder is struggling with this since it produces a heavier set of point clouds where the engines should be, however it does not recover their cylindrical shape.

6.2. Quantitative Evaluation

The final loss values for our three models after training for 500 epochs on the car and airplane dataset are shown in Table 1. We saw that most of our models overfit on the training dataset. This was not a concern to our project because we ran most of our interpolation experiments on examples from the training dataset.

From the table we see that the MSE loss is much lower than the Chamfer loss. This indicates that the MSE loss does a poor job at evaluating the true loss for the model. We can also see that the Pointnet val loss is lower than the MLP val loss while the Pointnet train loss is larger than the MLP train loss. This is likely because the Pointnet permutation invariant architecture does a better job at preventing overfitting of the model.

We noticed that the losses decreased for all three models. We do not provide loss curves since we only train three models where only two of the models use a comparable Chamfer loss. We also noticed that the loss across epochs was quite unstable. We believe this is because this is because as the model fit better to one category the loss for the other category spiked. Thus there were corresponding spikes in the loss.

6.3. Latent Space Analysis

To better compare the latent space learned by our three models, figure 3 provides the interpolated representations between two interpolants: a car and an airplane.

In figure 3, the baseline model consisting of fully connected layers and MSE loss is on the bottom row. Because

the encodings of the two interpolants are extremely inaccurate, so are their intermediate interpolations.

In figure 3, our second model in which retain our fully connected layers but add in Chamfer loss, is on the middle row. The encoding is of greater visual quality for both interpolants. We see that using a permutation invariant loss function has considerable advantages. Since Chamfer distance is attempting to minimize the misalignment of each point from the other point, there is a serious penalty if even one of the x,y,z coordinates aren't close to the target.

An interesting note in the interpolation is that we have a sharp transition between time 4 and time 5 of the middle row. Recall that since we are interpolating in the latent space and then decoding back into the point cloud space, what we might possibly be seeing here is a 128 dimensional hyperplane separating the spaces of the cars and planes. Once enough of the individual dimensions are close to a specific typical "car" or "airplane" vector then the decoder recognizes this transition and transforms it into that dimension.

Our last model, for which we replace the fully connected layers with a permutation invariant layer, is shown in the top row. Time steps 2 and 4 represents a more hybrid shape of the mixture of a car and a plane. This seems to suggest that the latent space learned by adding permutation invariance is more continuous. Furthermore step 2 is quite intriguing because it shows the formation of wing like structures on the car while also preserving the the hood and trunk of a pickup truck. Thus showing a smoother transfer of information than compared to the previous model.

Lastly, recall the goal of interpolating in the latent space: to measure the degree to which our compressed latent dimension is able to capture the geometric topology of the space represented by the car-plane shapes. Based on figure 3 it seems that compressability was largely improved by using Chamfer distance, while the smoothness of the geometric topology was improved by permutation invariance.

6.4. Interpolation Analysis

In the above interpolations we specifically found two categories: a car and an airplane, that are structurally and topologically different. To contrast this, in this section we interpolate between a chair and a sofa which share closer similarities in both of those aspects.

Visually we can see that the seat of the chair is expanding to form the couch cushions and the legs of the chair descend towards the floor to form the base. This interpolation confirms our earlier statement that our permutation invariant architecture is learning a smooth geometric topology. These images match those found in [1] figure 11.

Relating back to our example presented in the introduction, we have shown that the latent space dimensions have some correlation to chair leg length and width of the chair. Thus the latent space corresponding to the Pointnet Autoencodeer model is fairly interpretable.

7. Conclusion

Understanding the latent representation of shapes is a complex optimization problem that situates itself at the intersection of computer graphics and machine learning. We choose to do this by implementing an autoenoder architecture for point clouds and interpolating in it's latent space to understand the topological geometry of shapes from several different categories. The core of the auto encoder follow those that are in the image classification pipeline, however there some additional constraints when dealing with 3D point cloud data as opposed to 2D image data, which are previously outlined in several papers [9] [1]. To overcome some of the 3D data specific challenges we employed two specific tools. 1. We used Chamfer loss function that enforces permutation invariance. 2. We use a permutation invariant layer that is able to construct the same embedding for any ordering of the input points. This is relevant in 3D point clouds since a scanning software can emit points in any spatial ordering while in images there is a distinct pixel based coordinate system.

With these two tools we were able to model the geometric topology of different shapes through their similarity with other shapes in the latent space learned by our autoencoder. Concretely we found that when interpolating between geometry that had starkly different structural and topological geometry the boundary between them was more rigid (see figure 3) while those with similar features (see figure 4) had smooth boundaries.

8. Future Work

In this work we presented a modification of [1]'s original work where we interpolate between latent representation of at most 2 categories (ie. car and plane, sofa and chair) while the orignal paper only interpolates within a single category. However, recently this idea has been expanded to multi-category models [11] where the goal is to model the underlying manifold that represents all shapes. With such a model, one could even try to perform algebraic operation on the latent space (the original inspiration of our proposal) to qualitatively measure the effectiveness of the learned latent space.

Another direction of future work could be to better understand the latent space we have currently produced. Suppose that the points scattered on the surface were given a canonical ordering, then it would be interesting to only interpolate one of 128 dimensions and see if we can mutate parts of one category into another. For example one could imagine learning converting only the door of the car into the wings of an airplane while leaving the hood, and trunk

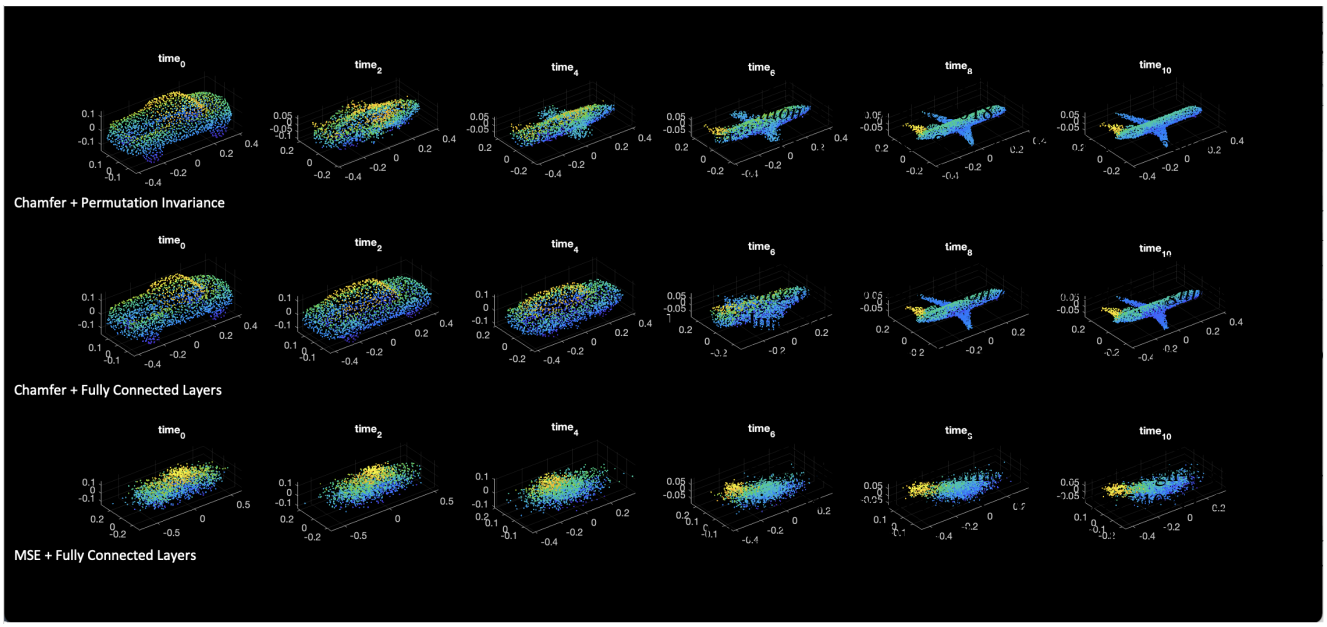

Figure 3.

of the car intact.

9. Contributions Acknowledgements

Samaksh, Sidhika, and Finsam gathered the data, discussed trade off of different permutation invariance loss functions.

Additionaly Samaksh and Sidhika constructed the model and evaluated the interpolation results.

Finsam also did the literature review.

Github repo: https://github.com/sidhikabalachandar/CS231nFinalProject

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

[2] T. Aumentado-Armstrong, S. Tsogkas, A. Jepson, and S. Dickinson. Geometric disentanglement for generative latent shape models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8181–8190, 2019.

[3] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[4] J. Chibane, T. Alldieck, and G. Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[5] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image, 2016.