# Shape Deformation Using 2D Guidance

Shahkar Javid[1,2,3]

[1]Università degli Studi di Roma "La Sapienza",
[2]Matricola: 2047305
[3]Email=Javid.2047305@studenti.uniroma1.it

**Abstract**
Shape deformation refers to the changes or alterations in the shape or geometry of three-dimensional (3D) objects or surfaces by applying a transformation to its components. 3D data is more complex than 2D images you may face several challenges including 3D data representation, batching, and speed etc. there is much more work done on 3D to 2d shape deformation, but in this paper we will be working on shape deformation of 3D object using only 2D guidance, means we have to use images or similar input to deform our 3D object as per our requirement. We will also try to integrate adversarial loss during our training to get efficient result or may just to see the difference between the two methods.

## 1. Highlights

- Generating a 2D dataset from a target mesh
- Implementing 3D deformation using 2D guidance
- Introducing adversarial loss while mesh deformation with 2D images
- Introducing attention layer into discriminator model to get more accurate result. Self-attention mechanisms capture dependencies between pixels or regions, enabling the model to focus on relevant parts of the input.
- Visualizing and analyzing the result of all the approaches while also evaluating their losses.

## 2. Introduction

There are many different ways to represent a 3D shape like Voxel Grid, Point Cloud, Mesh, Multiview Images etc. The choice of the representation depends on the data availability and influences also the architecture of the deep learning model. In this paper we are working on deforming a 3D object, more specifically a 3D mesh according to our desired shape using only 2D guidance. In Shape Analysis a shape (or 2D Manifold ) is represented as a 2D surface embedded in 3D, i.e. the shell of a 3D volume. By 2D guidance, we mean that we will only feed 2D images to our algorithm to transform our 3D source mesh. This implies that we first need to download or generate our own 2D dataset of our desired target shape. In order to do that we need to use a technique called rendering. Rendering is the process of computing an image from the scene parameters: Geometry, Camera, Light, Materials. Rendering algorithms are not designed to be invertible so instead we use Differentiable renderer in deep learing. Differentiable renderers are rendering algorithm that are fully differentiable, i.e. we can backpropagate the gradient of a loss function defined on the rendered image down through the scene parameters

To create a dataset for our task we also use a similar rending algorithm by simply loading an .obj file using pytorch3d and converting them into vertices and meshes.

During training the model, we will work on different losses altogether like silhouette loss, Discriminator loss, edge loss, normal loss, and Laplacian loss. Although our goal is to get the desired 3D shape which can be achieved by optimizing all these losses but we will not be as much concerned with the numerical value of these losses rather than the visual effect it has to our deformed mesh.

We have experimented with three different methods to achieve our results. First, we tried to use a simple 2D-3D approach, where we are feeding our 2D data which are silhouette images of our target mesh from different angle and faces. After feeding that into our training model we try to deform our source mesh into our desired target mesh by learning the deform vertices of our source mesh. We had to train for almost 2000 iterations to get a desired deformed target mesh.

The second method we use where we try to introduce adversarial loss into our training loop to get a better result. For this purpose we need to design a separate discriminator type model and feed it the predicted 2D images of our source mesh as well as the 2D images of our target mesh. The final result from this approach was a bit fuzzy then the simple method we tried before but it was still acceptable given half the training time.

The other approach we tried is to introduce a self-attention layer into our discriminator model. The self-attention mechanism is a technique commonly used in deep learning models, particularly in natural language processing (NLP) and computer vision tasks. It allows the model to weigh the importance of different input elements when making predictions. In the context of images, self-attention mechanisms capture dependencies between pixels or regions, enabling the model to focus on relevant parts of the input. In the context of our discriminator, we can integrate self-attention to enable the model to focus on important

features of the input image. Here we observe that the final result that is deformed source mesh is quite smooth and matching with the target mesh visually.

Throughout our experiments, we monitored the losses for each approach, but our primary focus remained on the visual results. Due to the inherent instability of losses in the presence of a discriminator network, we prioritized the visual accuracy of the deformed mesh over the stability of the loss values. As long as the source mesh was effectively deformed into the target mesh, the exact values of the losses were considered secondary. For visualization, We arbitrarily choose one particular view that will be used to visualize the result.

# 3. Related Work

The roots of shape deformation research can be traced back to the early days of computer graphics and geometric modeling in the 1960s and 1970s. Early works focused on representing and manipulating geometric shapes using mathematical formulations.

One of the pioneering works is "Computer Aided Geometric Design" by Thomas Sederberg and Jianmin Zheng (1989), which provides an overview of geometric modeling techniques, including shape representation and deformation. In the 1980s and 1990s, researchers began exploring the use of physical simulation and finite element methods for shape deformation. These techniques allowed for more realistic and physically plausible deformations of shapes.With the advent of geometry processing techniques in the early 2000s, shape deformation research saw significant advancements. Techniques such as Laplacian smoothing, mesh regularization, and energy minimization became central to shape deformation algorithms.

Works such as "Mesh Optimization" by Alliez et al. (2003) and "Surface Deformation for Shape Manipulation" by Sorkine et al. (2004) introduced novel approaches for mesh deformation based on geometric optimization and energy minimization

Recent advances in deep learning have significantly impacted the field of shape deformation by offering new techniques for learning deformation models directly from data.[1][2] These approaches leverage the power of neural networks to automatically learn complex mappings between input and output shapes, enabling more flexible and data-driven deformation methods. Techniques such as deep neural networks[3], generative adversarial networks (GANs), and variational autoencoders (VAEs) have been used for learning shape deformation models from data.

Works such as "Learning 3D Deformation of Animals from 2D Images" by Gao et al. (2019)[4] and "GraNet: Graph-Attention Network for 3D Object Classification" by Li et al. (2020) [5][6] explore the use of deep learning for shape deformation tasks. Learning Implicit Fields for Generative Shape Modeling" by Chen et al. (2019) demonstrate how deep learning can be used to learn shape representations suitable for deformation and interpolation.

Most recently a new framework “PyTorch3D” has been introduced by Meta formerly known as Facebook. It provides efficient and flexible tools for working with 3D data, including meshes, point clouds, and textures, using the PyTorch deep learning framework. While PyTorch3D is not a traditional research paper, its development represents a significant contribution to the field of computer graphics and geometry processing, particularly in the context of shape deformation. By using pytorch3D we can easily implement flexible mesh representation and processing, it also has differential rendering module allowing us to train neural networks that directly manipulate 3D shapes while taking into account their appearance in rendered images. Differentiable rendering is a relatively new and exciting research area in computer vision, bridging the gap between 2D and 3D by allowing 2D image pixels to be related back to 3D properties of a scene. For example, by rendering an image from a 3D shape predicted by a neural network, it is possible to compute a 2D loss with a reference image. Just recently a very famous paper was publish in 2021 titled "Text2Mesh: Text-Driven Neural Stylization for Meshes" by Oscar Michel and other[7] which focusses on stylizing a 3D mesh by predicting color and local geometric details which conform to a target text prompt.Text2Mesh requires neither a pre-trained generative model nor a specialized 3D mesh dataset.

There were many similar shape deformation work done in the past just like aforementioned, in some paper the experimented with 3D to 3D deformation there was no 2D guidance involved, simply chamfer loss was calculated between source point clouds and target point clouds in the optimization step, which pytorch3D can do very easily and efficiently. Since pytorch3D seamlessly integrates with the broader pytorch ecosystem that make it perfect for our task.

# 4. Description

As mentioned before, we are going to use the approach to deform our source mesh into the desired target mesh. But before getting into that we need to create a dataset to trained our model and learned a bit about the losses we are going to us in our experiment.

## 4.1. Dataset Creation

To create a dataset we use pytorch3D, first, we load the target object which is an .obj file, which is then converted into vertices and meshes. We then scale normalize and center the target mesh to fit in a sphere of radius 1 centered at (0,0,0). It should be noted that normalizing the target mesh, speeds up the optimization but is not necessary. since our target mesh is normalized it make sense that our source mesh which we want to deform must also be normalized. In this project, we experimented with several shapes to deformed, but it should be noted that the more complex the shape is the more time it will take to deform during training.Initially, we only used

a sphere of radius 1 for simplicity but the source mesh can be of any shape, we also experimented with other shape and it took way more time to deoform aith many artifacts. after that, we use the pytorch3d resterizing module to define the settings for rasterization and shading.Rasterization settings for differentiable rendering, where the blur radius initialization is based on [2]. Refer to rasterize_meshes.py on the pytorch3d website mentioned in online resources in the end for explanations of these parameters. The differential silhouette renderer where we can sample different camera positions that give us multiple viewpoint of target mesh and then We render a synthetic dataset of silhouette images which can be seen in figure 1 and 2 , where our target mesh is dolphin and we want render multiple 2D images from different angle.

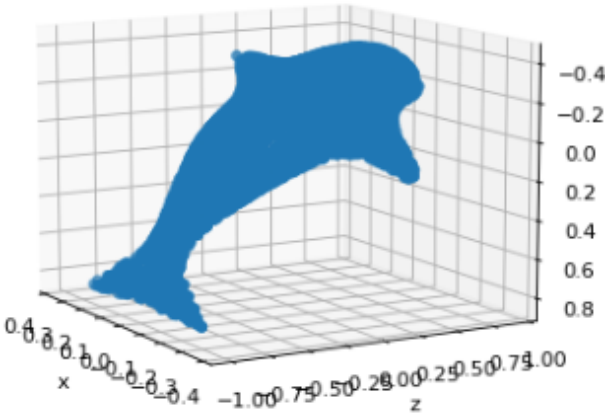


**Figure 1:** Target Mesh

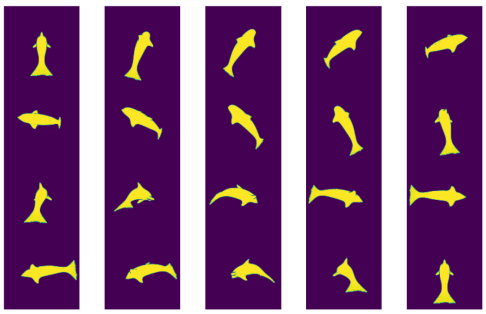

**Figure 2:** Dataset

## 4.2. Losses

In the project we are going to work on many different losses altogether to get our desired shape as said before we are going to monitor the losses but in the end, we are only interested in the final visual result rather than the numerical value of losses, there are several losses that came to mind when we work on task related to shape deformation also known as shape regularizers like chamferloss, laplasian loss etc. the losses we are going to use are briefly explain below;

### 4.2.1. Silhouette Loss

Silhouette loss that we will use in our optimization is nothing but just a mean squared loss between the predicted silhouette images of our deformed/source mesh and target mesh, Its kind of like MSE loss.

### 4.2.2. Mesh Edge Length

The mesh edge length regularize aims to minimize the length of edges in the predicted mesh. This regularizer penalizes deviations from a desired edge length distribution, promoting meshes where edges are neither too long nor too short. It helps prevent the creation of overly stretched or compressed regions in the mesh, contributing to the overall smoothness of the deformed shape.

### 4.2.3. Mesh Normal Consistency

This regularizer enforces consistency across the normals of neighboring faces in the mesh. Normal consistency is crucial for ensuring that the surface of the mesh appears smooth and continuous. It penalizes abrupt changes or discontinuities in surface orientation between adjacent faces, which can lead to visual artifacts such as sharp creases or discontinuous shading.

### 4.2.4. Mesh Laplacian Smoothing

The Laplacian smoothing regularizer is based on the Laplacian operator, which calculates the difference between the average of a vertex's neighbors and the vertex itself. By minimizing this difference for all vertices in the mesh, Laplacian smoothing encourages a globally smooth deformation of the shape. It helps to distribute deformations smoothly across the surface of the mesh, reducing localized distortions and preserving overall shape characteristics.

### 4.2.5. Discriminator Losss

When we will start introducing adversarial loss in our optimization loop we will have to work on discriminator loss as well. In this loss the discriminator network has to identify the predicted 2D images rendered from the deformed mesh as fake and the 2D rendered image from the target mesh as real using MSE loss as criterion.

In the end we have to average all losses together and take an optimization step but before that We also assigned some weight to each loss contributing in the optimization step,see table 1. the weight varies for each approach because it is a kind of hyperparameter and one can find out the specific

**Table 1**
Weigthage Assigned For Loss functions During Each Approach

| Approach | Silhouette Loss | Edge Loss | Normal loss | Laplacian Loss | Discriminator Loss |
|---|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 0.01 | 1.00 | - |
| 2 | 1.5 | 0.7 | 1.0 | 0.01 | 1.00 |
| 3 | 0.30 | 1.5 | 0.5 | 0.01 | 0.5 |

weight given to the losses by grid search and experimenting with different values and analyzing the results.

## 4.3. First Approach

We started with the simple approach first, after creating our own dataset using pytorch3D renderer as mentioned earlier and shown in figure 1.2 we first load a random source mesh that need to be deformed to our target mesh, in our case we are loading a spherical source mesh, but it can be of any shape. The optimizer we choose in this case is SGD optimizer with momentum = 0.9 and learning rate =1. After that during our optimization loop we offset vertices of our source mesh and save it in a new mesh, here the vertices that have been offset or deformed are learnable parameters. Means model is trying to learn in each iteration which vertices of our source mesh to deform. After this step we calculate four different losses mentioned in section 3.2. different weightage was given to each loss which was finalized through grid search after many experiments. The 2D images of new source mesh was rendered to calculate the silhouette loss by comparing it with 2D images of target mesh. The optimization loop was run for 2000 iteration. We try to monitor the losses and visualize the result during the training as well as in the end of the training. For visualization in 3D the mesh was simply converted into point clouds and then plotted, we also visualize it in 2D from a single view for both target mesh and source mesh to get a better picture of the results. The pipeline of this approach can be understand easily from figure 3.

In figure 5 you can observe how a source mesh is being deformed into a target mesh over a course of iteration which in our case is a dolphin.

## 4.4. Second Approach

In the second approach we introduce adversarial loss into our optimization loop. It means we must design a separate discriminator network. The network we design consists of for convolutional layer with instance normalization and the activation function that was chosen is leaky-Relu in between the conv2D layer and sigmoid in the end to get the classification of fake and real image. We integrate this discriminator network with the optimization loop of our algorithm during training we feed the real 2d images of target mesh into discriminator as well as the 2d images of deformed mesh to our discriminator network. It must classify both real and fake correctly. We calculate the discriminator loss

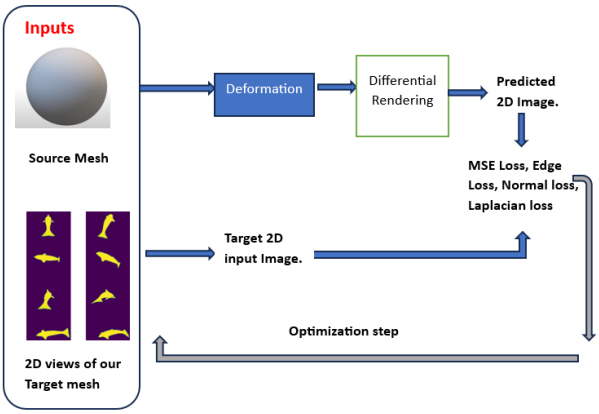


**Figure 3:** the whole pipeline of method 01 can be observed in a simple and easy way

using MSE loss as criterion. In the end some weightage is also given to discriminator loss as well and combined with other losses to take the optimization step. The optimizer we choose here is not SGD but Adam optimizer because it is best known that Adam optimizer work very well when it comes to adversarial training. The rest of the algorithm is the same as first approach, the pipeline of this approach can be understood easily from figure 4.

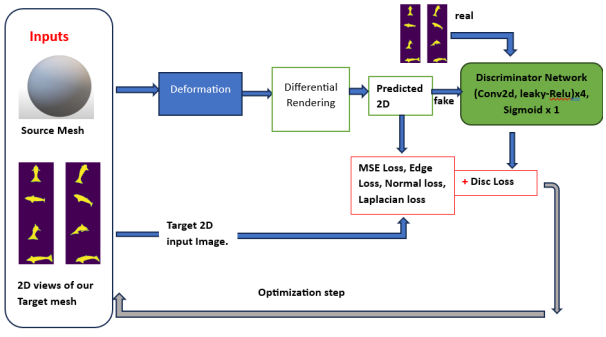


**Figure 4:** The whole pipeline of method 02 can be observed where discriminator network is also integrated

## 4.5. Third Approach

In the third and the last approach we just tried to change the discriminator network try to make it more complex,

**Table 2**
Hyperparameters

| Approach | learning rate | Optimizer | Epochs |
|---|---|---|---|
| 1 | 1.0 | SGD | 2000 |
| 2 | 1e-2 | Adam optimizer | 1000 |
| 3 | 1e-2 | Adam optimizer | 1000 |

because the result that we got from the second approach was a bit fuzzy and rough. So, what we did first was to introduce a couple of more convolutional layers into it and the we try to integrate a self-attention mechanism into our discriminator network. Convolutional layers have limited receptive field, which can make it difficult for the discriminator to capture larger structures in the image. This is where the self-attention mechanism comes in. The self-attention mechanism allows the discriminator to focus on specific regions of the image that are relevant for a particular task. This is like how attention works in language translation and image captioning, where the model focuses on different parts of the input to generate the output. The rest of the algorithm work just as the previous one, we compute the discriminator loss; combine it with other four losses, each loss has its own weightage for contribution in optimization step. After that we backpropagate to learn the deform vertices.

In table 2 you can see all the hyperparameter for aforementioned approaches.

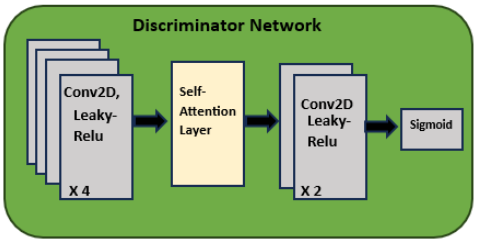


**Figure 5:** The inside of Discriminator network can be visualized in detail in above diagram

# 5. Results

To analyze our approaches we were mainly focused on the visual result rather than the losses, but still we were keeping track of the all the losses to be as low as possible specially in first approach. Below you can see the result of each of our method

## 5.1. First Approach

In the first approach, which was without adversarial loss, we train our model for up to 2000 iteration. Which was more than enough to get our desired shape in smooth and efficient way. In later figure 6 you can see the source mesh being gradually deform into target mesh over the course of 2000 iteration. We also visualize the mesh in the for of point clouds to get better visual understanding.

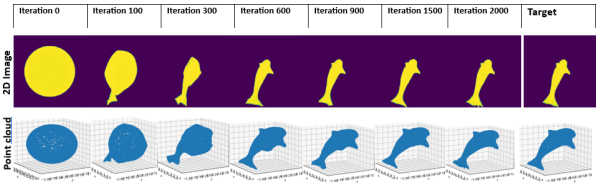


**Figure 6:** 1st approach: Gradual mesh deformation over the course of 2000 iteration, on the far right you can see the target 2D image and target 3D mesh

We also keep track of all the four losses and recorded them for each iteration for later evaluation and understanding which can be seen in figure 7 above.

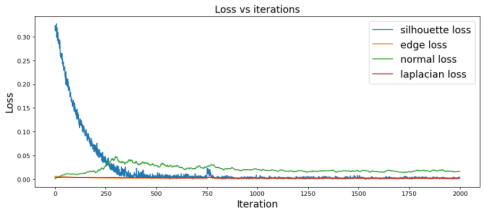


**Figure 7:** Losses optimization plotting over each iteration

## 5.2. Second Approach

In this approach we introduce discriminator loss, to make our model more efficient. The training take longer time in this method then the previous one, so we run our training loop for only 1000 iteration. Which is half as much we did in earlier approach. The visual result were a little fuzzy and rough but given that we train it for only 1000 iteration the result was quite acceptable. In mentioned figure 8 the deformation during training loop can be visualize both in 2D as well as in 3D views.

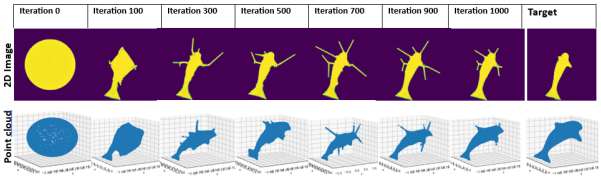


**Figure 8:** 2nd approach: Gradual mesh deformation over the course of 1000 iteration, on the far right you can see the target 2D image and target 3D mesh

For the losses , we also maintain the record all the losses over each iteration including the newly introduce discriminator loss. In figure 9 you can see all the losses, which is very different than the losses we saw in earlier approach. This is only because of our discriminator network which is playing a min-max game between the training loop. Due to this we are only concern with the visual effects of deformed mesh rather than the numerical values of losses.

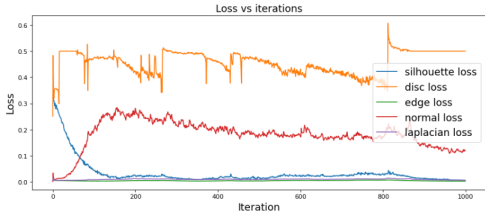


**Figure 9:** 2nd approach: Losses optimization plotting over each iteration

### 5.3. Third Approach

In the last approach as mention earlier we only made changes to discriminator network , tried to make it more complex so it can better understand the features within the image and we can avoid the roughness and irregularities we got in previous discriminator network, by also introducing attention layers within it. We train this network also for 1000 iteration and the visual result were quite satisfactory you can observe the final result in below figures and comparison between the simple discriminator network and the complex one. You can clearly see the difference in result in mention figures 10 for the simple discriminator network and the one with self-attention mechanism. There are much less artifact as compared to earlier.

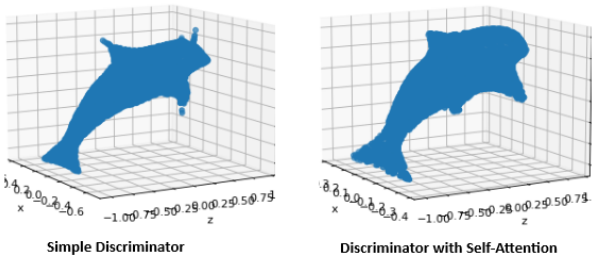


**Figure 10:** On the right side result of simple discriminator can be observed while on the left side the result of discriminator with Self-attention mechanism

## 6. Conclusion

In this paper, we explored the task of shape deformation using 2D guidance, where a 3D object is deformed to match a target shape using only 2D images as input. We proposed three approaches to tackle this problem, each building upon the previous one. Our first approach used a simple 2D-3D approach, where we fed 2D silhouette images of the target mesh into our training model to deform the source mesh. Our second approach introduced adversarial loss into the optimization loop, which improved the results but still produced a slightly fuzzy output. Finally, our third approach incorporated a self-attention mechanism into the discriminator network, resulting in a smooth and visually appealing deformed mesh.

Through our experiments, we demonstrated the effectiveness of our approaches in deforming 3D objects using 2D guidance. We showed that by incorporating adversarial loss and self-attention mechanisms, we can improve the quality of the deformed mesh and reduce the training time, but it should also be noted that this can also lead to mode collapse which I faced many times while experimenting with the weights of the losses. Additionally, we found that deformation is heavily dependent on the target shape as well. The more complex the shape, the longer it will take to deform. This work has potential applications in various fields, including computer-aided design, computer graphics, and robotics.

Future work includes exploring other techniques to improve the deformation results, such as using more advanced neural network architectures or incorporating additional constraints into the optimization process. Additionally, we plan to extend our approach to handle more complex shapes and scenes, and to investigate its applicability to real-world problems. Overall, we believe that our work represents an important step towards developing more sophisticated and versatile shape deformation algorithms that can be used in a wide range of applications.

## References


[1] Z. L. Y. F. W. L. Y.-G. J. Nanyang Wang1, Yinda Zhang2, Pixel2mesh: Generating 3d mesh models from single rgb images, ECCV (2018).

[2] S. Liu, T. Li, d Hao Li, Y. Chen, Soft rasterizer: A differentiable renderer for image-based 3d reasoning (2019).

[3] T. H. Kato, Yoshitaka Ushiku, Neural 3d mesh renderer (2018).

[4] R. B. Angjoo Kanazawa1, Shahar Kovalsky2, D. Jacobs1, Learning 3d deformation of animals from 2d images (2019).

[5] Z. L. Shengwei Qin, L. Liu, Robust 3d shape classification via non-local graph attention network (2019).

[6] Y. X. Rong Huang, U. Stilla, Granet: Global relation-aware attentional network for semantic segmentation of als point clouds (2019).

[7] R. L. S. B. R. H. Oscar Michel, Roi Bar-On, Text2mesh: Text-driven neural stylization for meshes (2021).

# 7. Appendices

## A. Online Resources

- PyTorch3D
- .obj file for target mesh

## B. Additional Results

We also experimented with may different shapes to deform and analyze their result, some of the result can be sen in this section.

### B.1. Shape 2: Airplane

In the figure 11, you can see the transforming result of source mesh into target mesh before during and after training.also note that the target shape which in this case is an airplane has too many complex and tiny details which cannot be achieve easily, it may required additional training time or different approach.

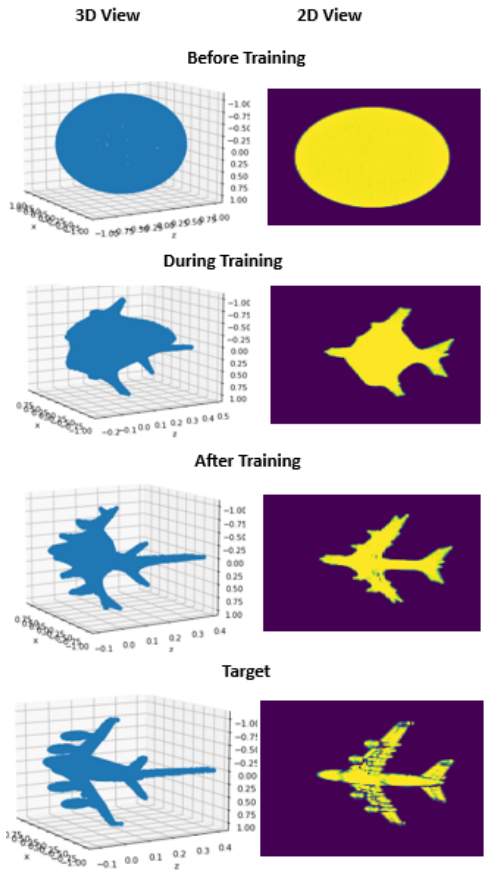


**Figure 11:** shape deformation from sphere into Airplane

### B.2. Shape 3: Human

We can also observe the result of the deformation of the sphere into a human in figure 12, which has a lot of artifacts given that the target shape is too complex, but we train it for 2000 epochs which is the same as we train it for our dolphin mesh which was way simpler than human-shaped mesh.

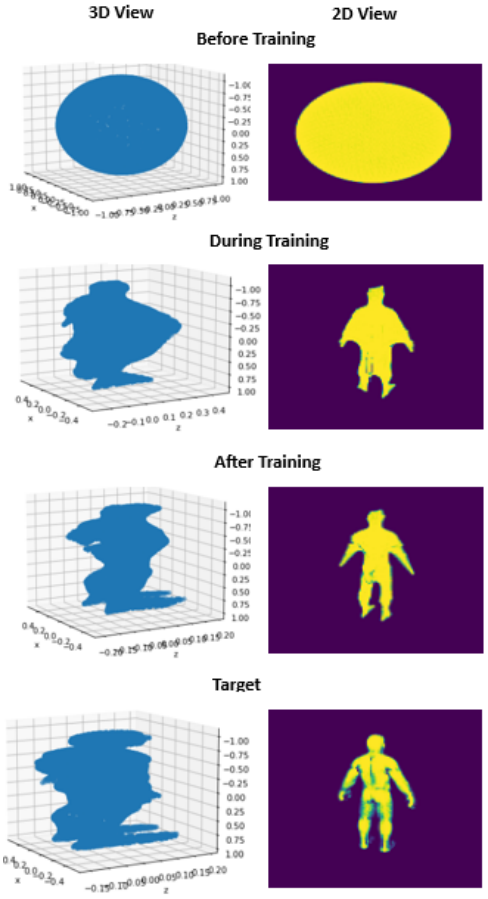


**Figure 12:** shape deformation from sphere into Human