Shape Generation using 3D-GAN

Ryan Huang

December 28, 2023

1 Introduction

This project aims to 1) create a procedure to generate spheres and 2) train a model to mimic its behavior. To generate the spheres, I randomly generated points from the parametric equation of a sphere. Then, to train the model, I used a 3D Generative Adversarial Network (3D-GAN) model. In this report, I show the progress, results, and evaluations below.

2 Sphere Generation Procedure

2.1 Sphere Generation

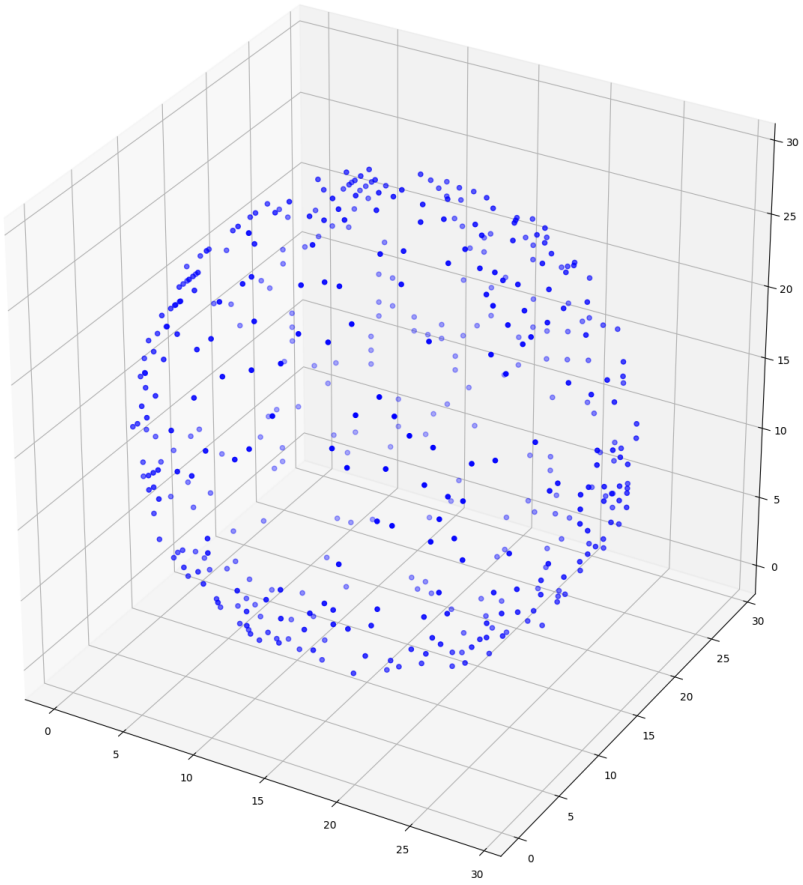
To generate spheres randomly, I first uniformly generated random values for the spherical coordinates (θ, ϕ). Then, they were converted to cartesian coordinates by the conversion:

$$(x, y, z) = (\rho cos\theta sin\phi, \rho sin\theta sin\phi, \rho cos\phi)$$

This would help me create point clouds of spheres. However, one issue I noticed was that lots of the points were clustered around the top and bottom of the sphere (the poles). A solution to this I found online was to sample ϕ randomly from arccos(x), where $-1 < x < 1$.

2.2 Sphere Dataset

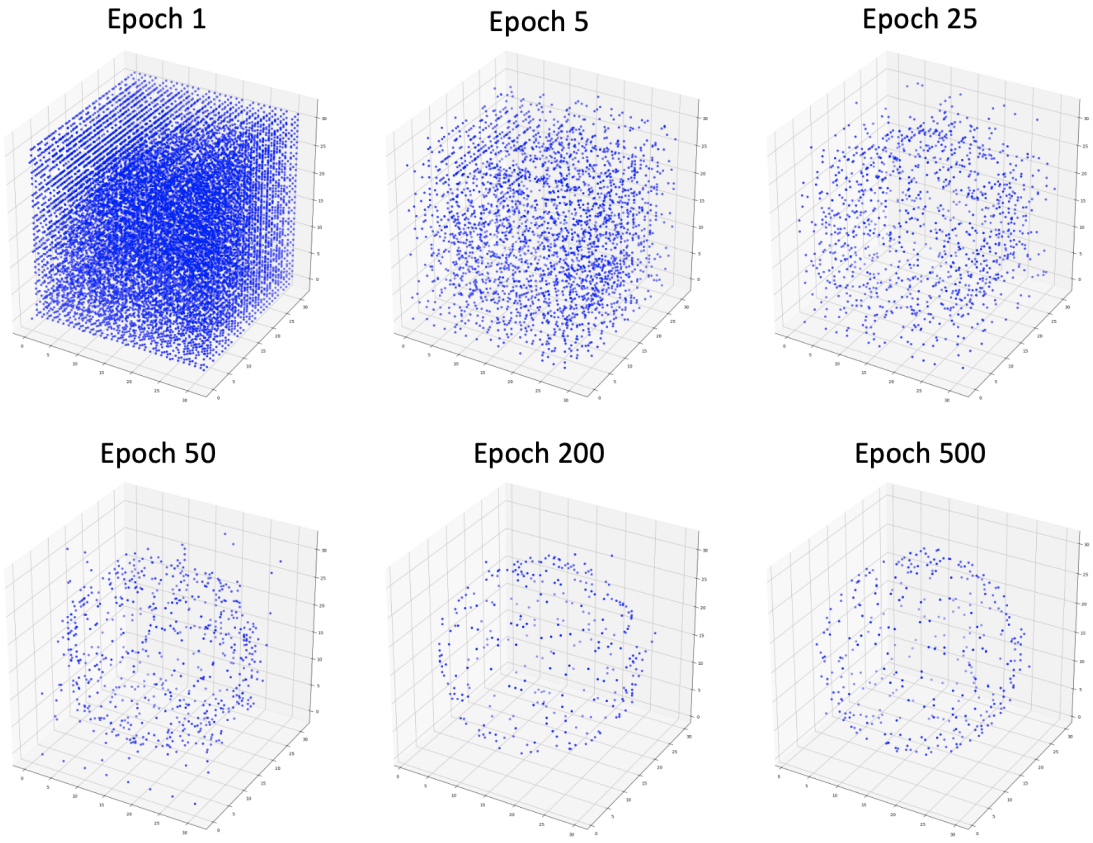
Using this generation procedure, I generated 1000 spheres, all point clouds with 500 points (note that there is also a dataset with 1000 points). I picked 15 as the radius because this would best fit in to the 32x32x32 cube generated by the model. These point clouds were saved as 3D matrices in .npz files. This is an example generated sphere:



3 3D-GAN Model

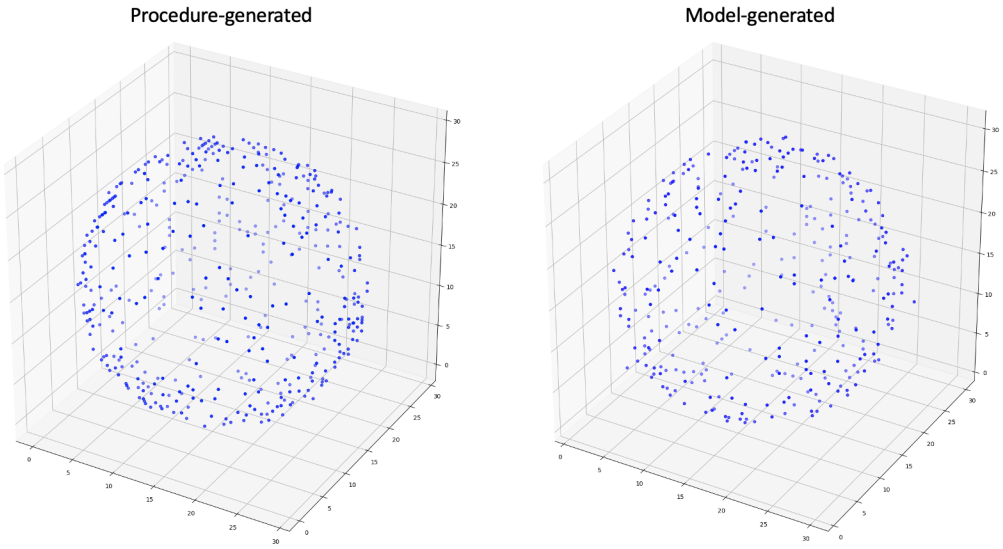
The GAN model consists of two components: a generator and discriminator. In the implementation outlined by Wu et al., the generator maps a 200-dimensional vector randomly sampled from a latent space to a 64x64x64 cube (representing the object in 3D) [1]. However, in my implementation, I map the latent vector into a 32x32x32 cube, since the 64x64x64 cube required more computing power than my computer could handle. Per the paper, the 200-dimensional vector z is a randomly sampled noise vector uniformly sampled from [0,1]. The discriminator determines how well we can distinguish the generated object to the objects in the dataset.

The generator has 5 convolutional layers with ReLU layers between each and a sigmoid layer at the end; the discriminator is similar with LeakyReLU layers instead of ReLU. In my implementation I trained my model for 500 epochs with batch sizes of 32. Shown below is the progression of training:



4 Generation Performance

Visually, we can see that the generated shape is definitely a sphere. We can easily see this by comparing the sphere generated by the procedure and the sphere generated by the trained model:



To show that my trained model correctly generated a sphere mathematically, we can compute the root mean square error (RMSE) of the model-generated spheres to the actual sphere. Since we know the radius of the spheres is 15, the expected average radius in each generated should be 15.

As shown in Table 1 (next page), the RMSE of the generated spheres is 0.017, which indicates that radius of the spheres, on average, differ from the radius of the procedurally-generated spheres by 0.017. Overall, the generated spheres are fairly close to a perfect sphere.

5 References

[1]$https://doi.org/10.48550/arXiv.1610.07584$
[2]$https://pytorch.org/tutorials/beginner/dcgan\_faces\_tutorial.html$
[3]$https://github.com/rimchang/3DGAN-Pytorch/tree/master$

Annulus	**Average Radius**
1	15.020
2	15.014
3	15.021
4	15.009
5	15.016
6	15.025
7	15.024
8	15.009
9	15.010
10	15.015
RMSE	0.017

Table 1: Table with the characteristics of the 10 generated spheres and the root mean square error (RMSE) from the original spheres. After generating the spheres, it was visually apparent that the generated shapes were similar to the original spheres. However, to numerically evaluate their similarities, the average radius were computed for each shape. Then, the root mean square error between these values and the values of the original spheres were calculated.