Masterarbeit

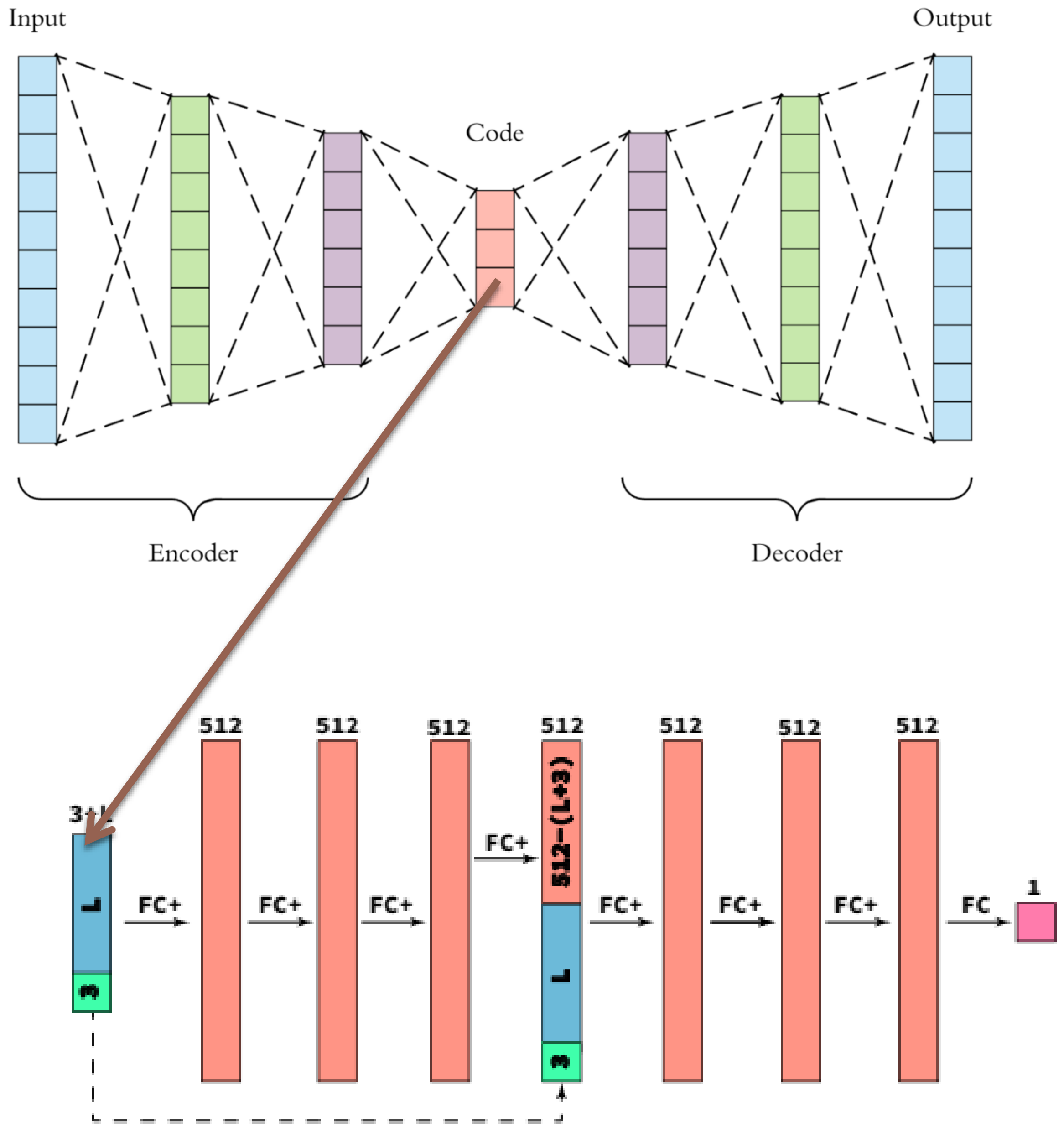
Yannick Kees

01.06.2022

Alternatives to Fourier Features

- SIREN (sin as activation)
- SAPE (progressive encoding, Spatial adaptivity, Sparse Grid sampling)
- Learned Initializations for Optimizing Coordinate-Based Neural Representations

Learning the solution operator



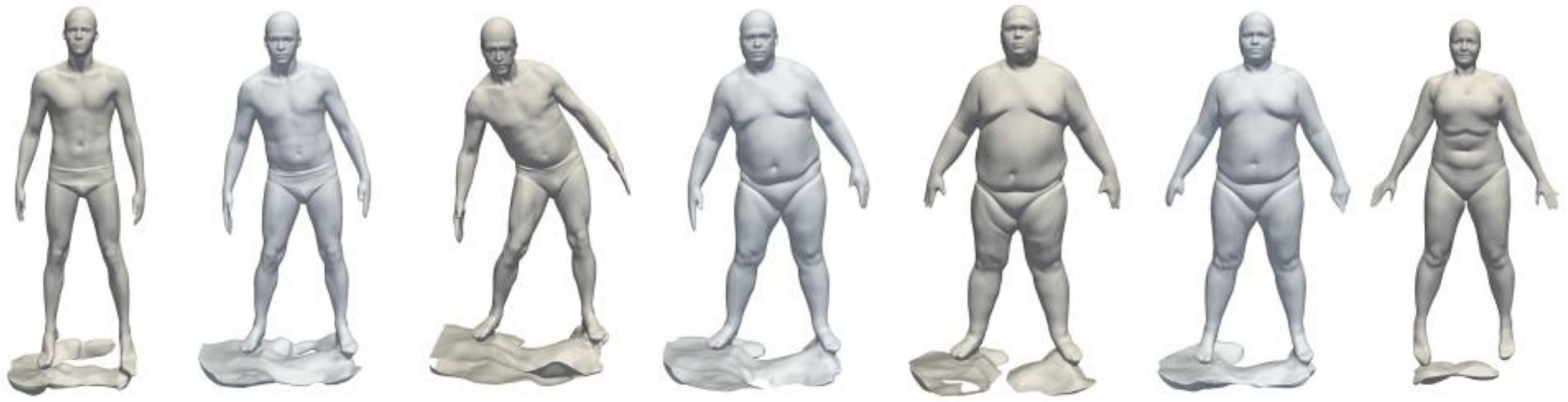
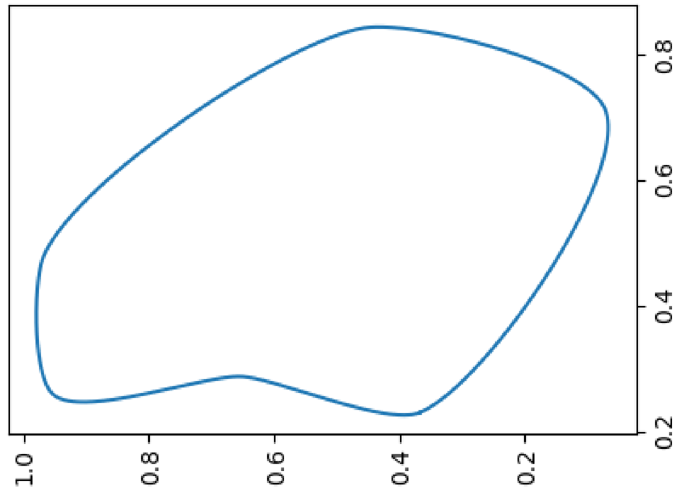


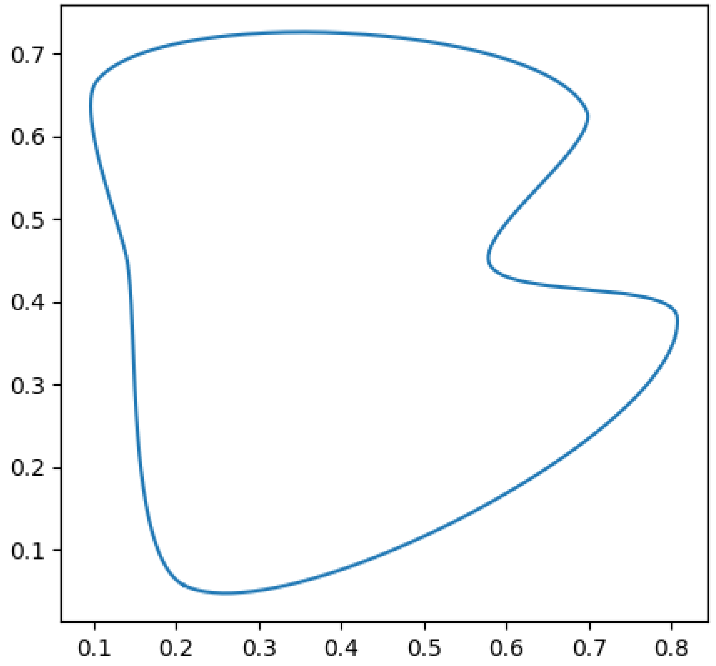
Figure 12. Averaged shapes: Zero level sets (in blue) using averages of latent vectors of train examples (in gray).

Shape space exploration. For qualitative evaluation of the learned shape space, we provide reconstructions obtained by interpolating latent vectors. Figure 12 shows humans shapes (in blue) corresponding to average interpolation of latent vectors z_j of training examples (in gray). Notice that the averaged shapes nicely combine and blend body shape and pose.

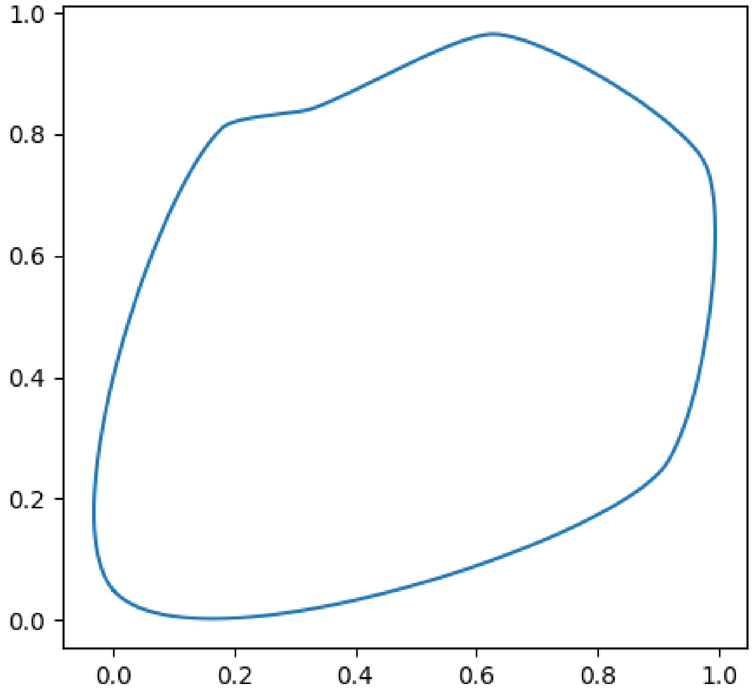
2D/3D Point Clouds

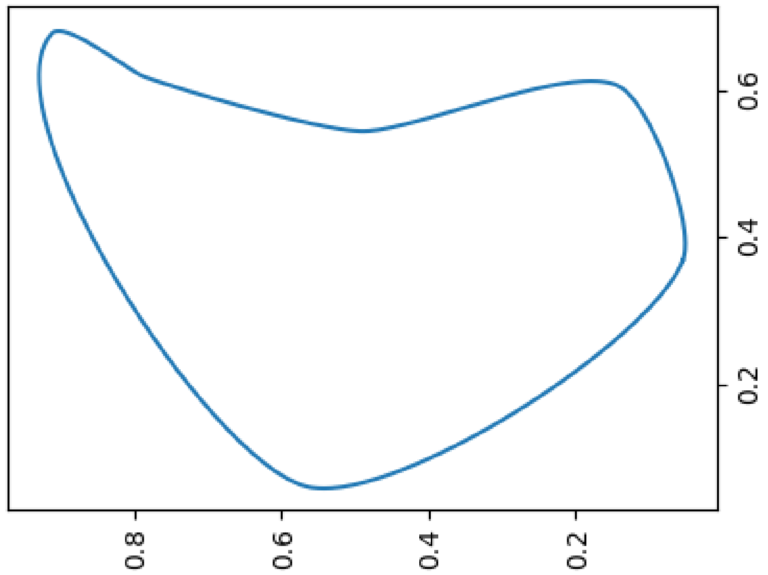
Randomly created



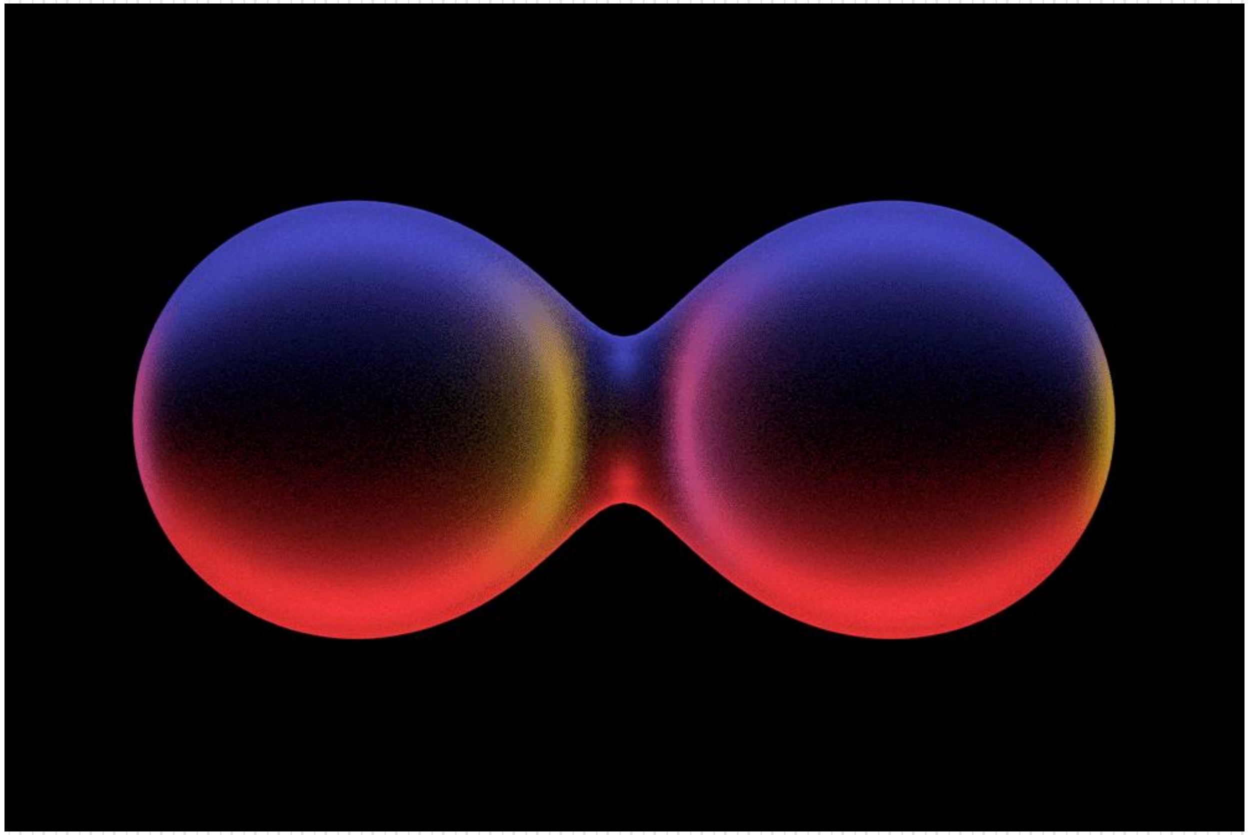


Bezier Curves





Metaballs



2. Metaball Mathematics

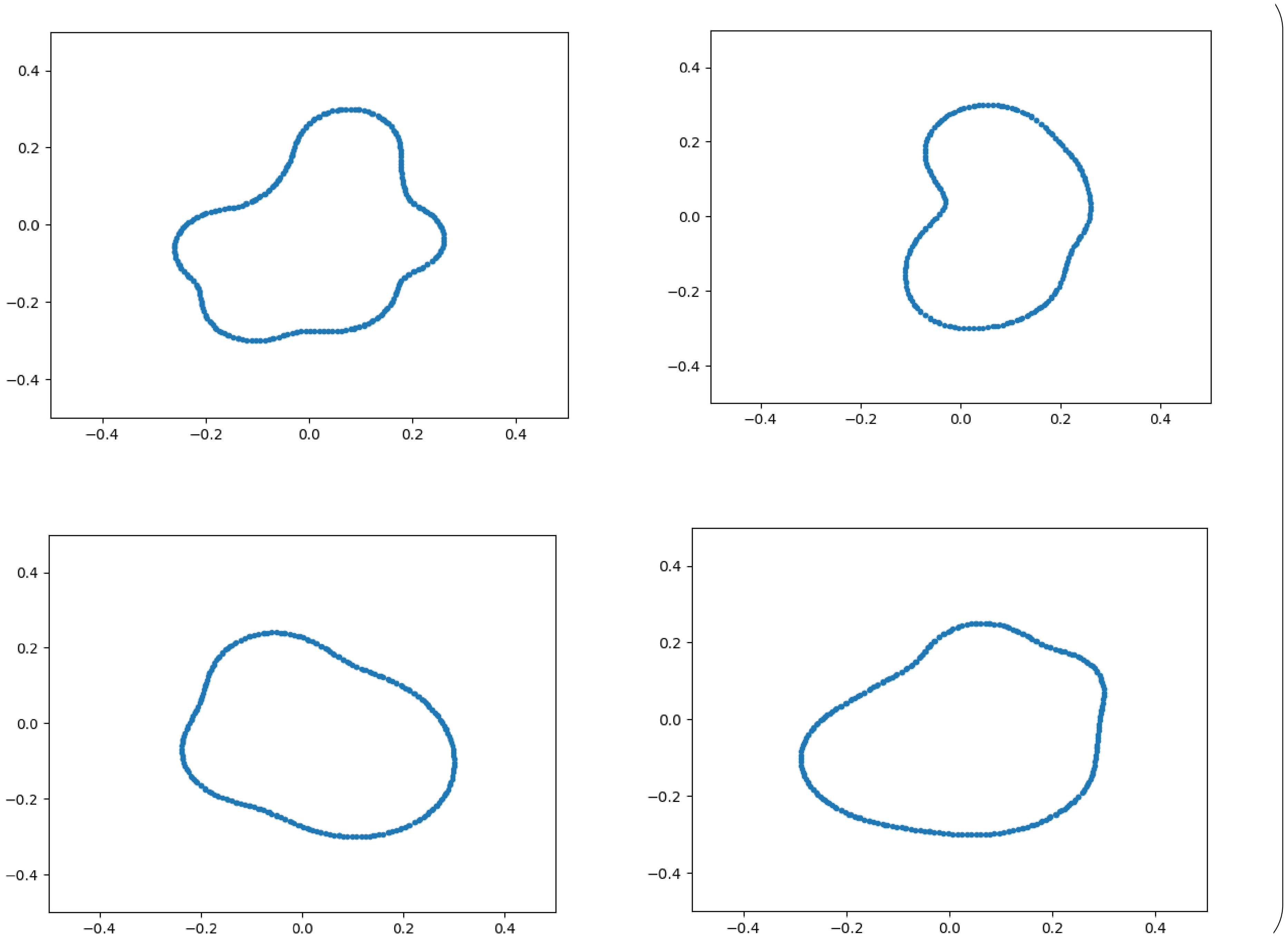
Scalar Potential Field Metaballs are described using the implicit Equation 1 below:

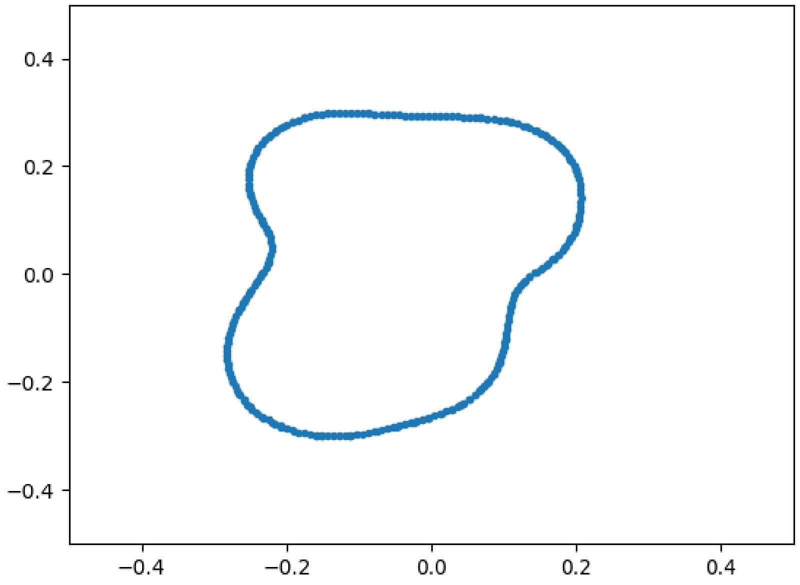
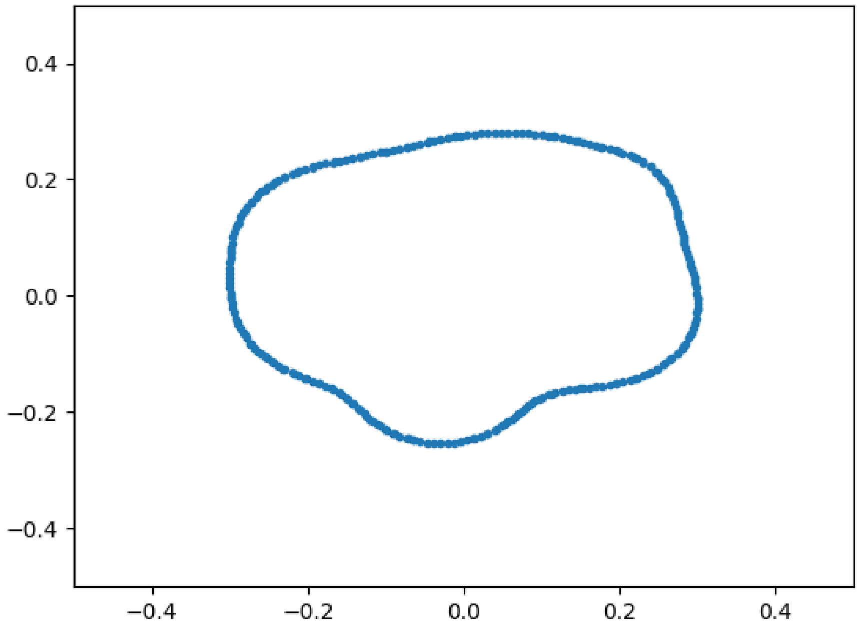
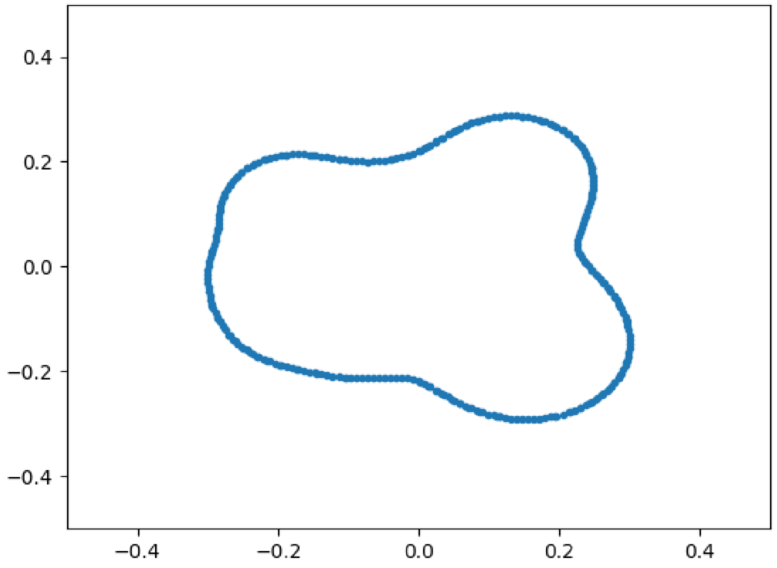
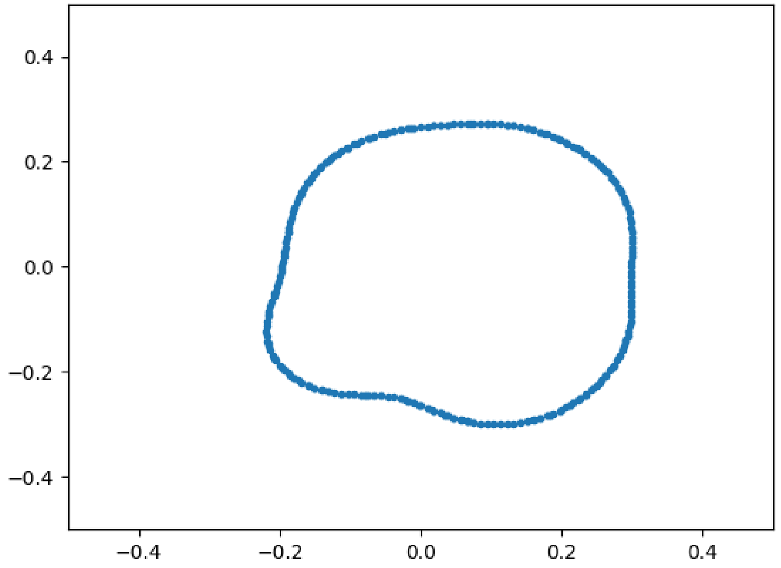
$$\sum_{n=1}^{k} \frac{s_n}{||m_n - p||^g} > r \tag{1}$$

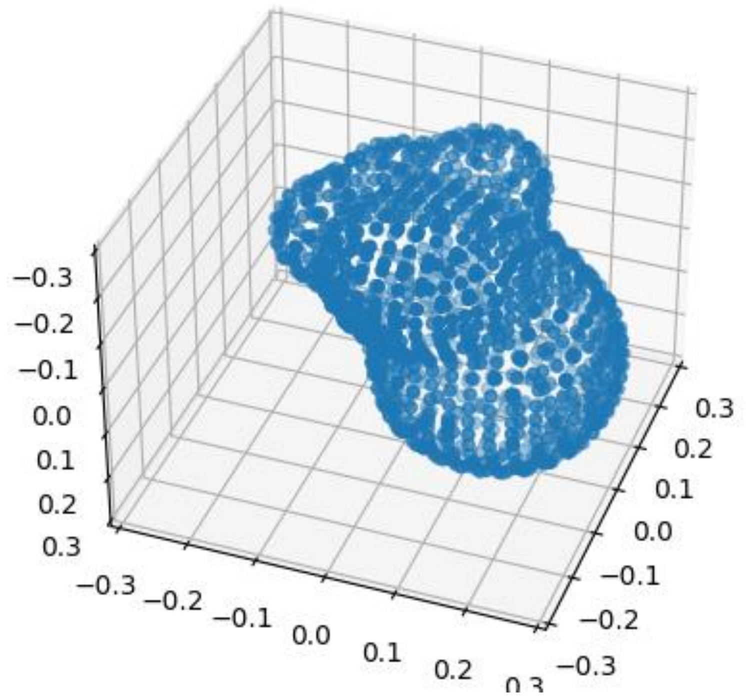
where m_n is location of metaball number n, s_n is size of metaball number n, k is number of balls, g 'goo'-factor, which affects the way how metaballs are drawn, r is the threshold for metaballs, p is the place vector, and $||x||$ indicates the magnitude (length) of vector x.

Normals Metaballs are described by a force field, as shown in Equation 1, so normal for any given point is easy to calculate with the help of gradient, as shown below in Equation 2:

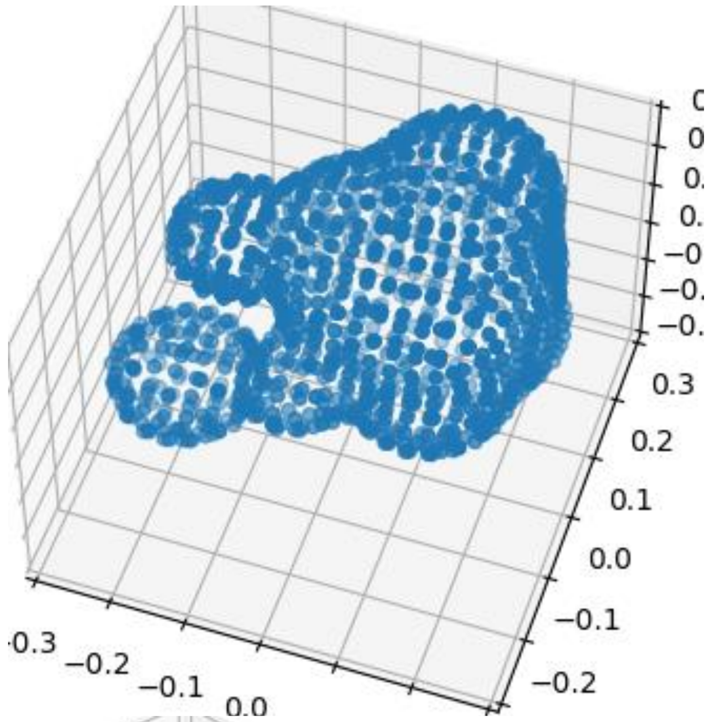
$$\begin{aligned} \text{normal} &= \nabla \sum_{n=1}^{k} \frac{s_n}{||m_n - p||^g} \\ &= \sum_{n=1}^{k} (-g)(s_n) \frac{m_n - p}{||m_n - p||^{2+g}} \end{aligned} \tag{2}$$


0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

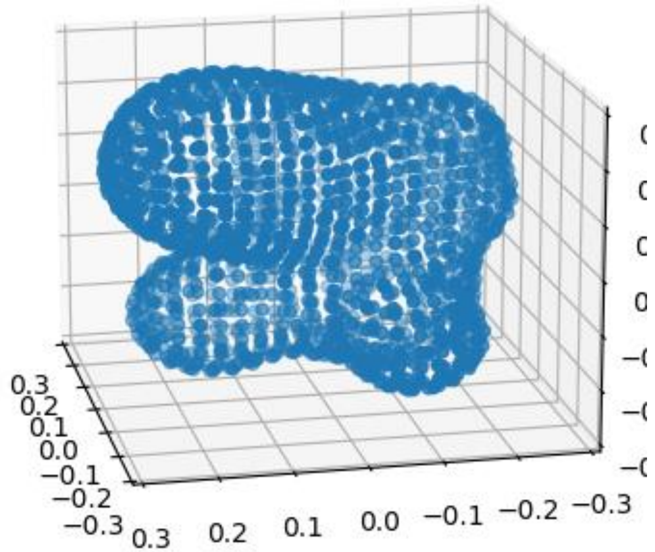

0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4

0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4



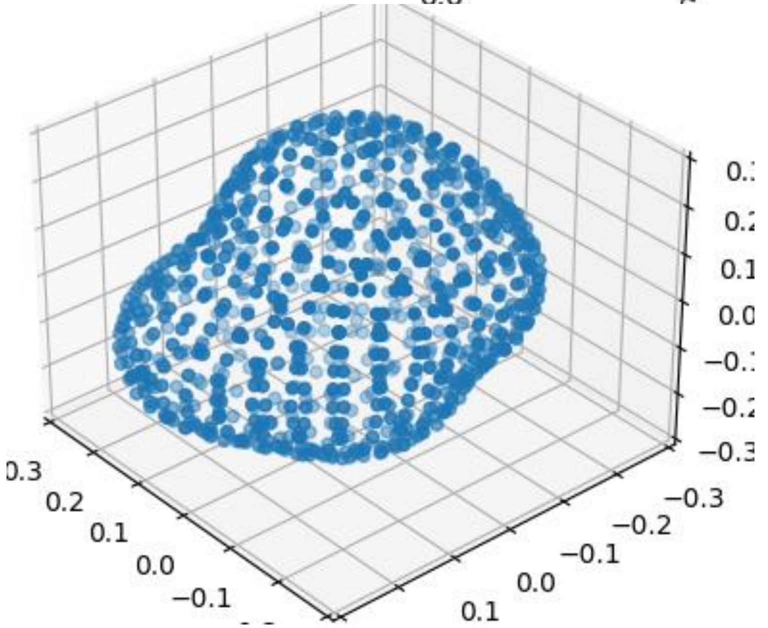
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



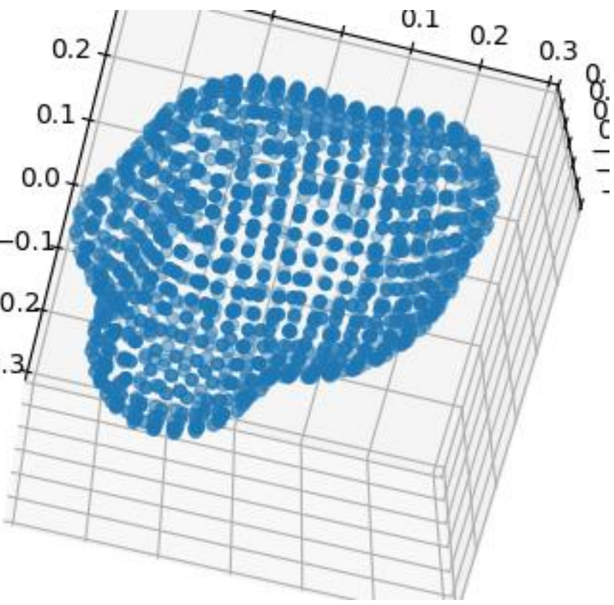
0.3
0.2
0.1
0.0
−0.1
−0.2
·0.3
−0.2
−0.1
0.0



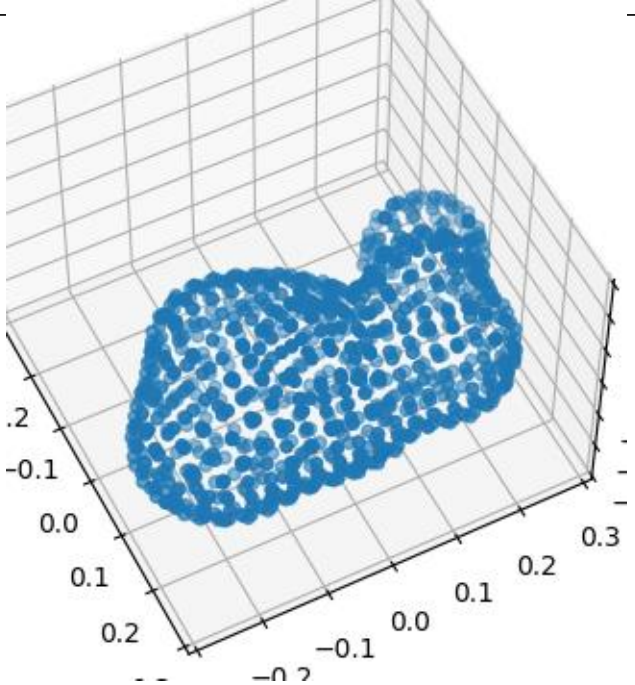
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



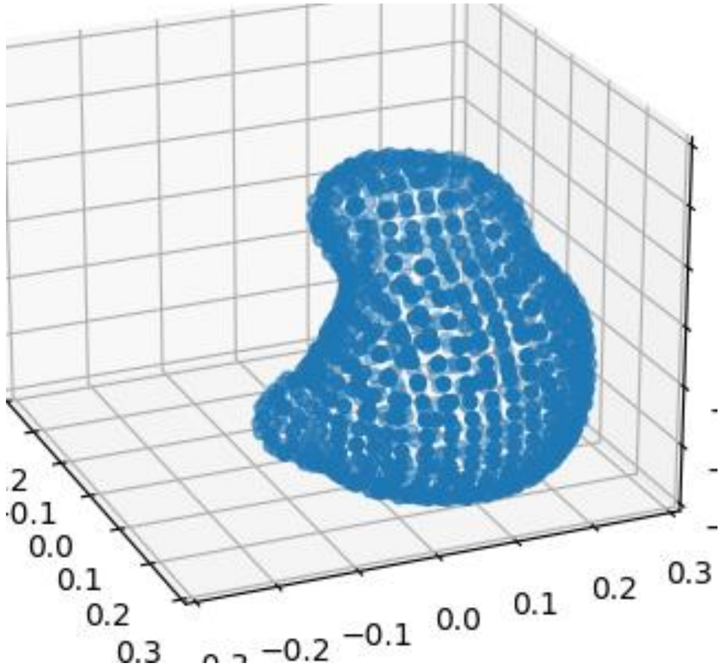
0.3
0.2
0.1
0.0
−0.1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1



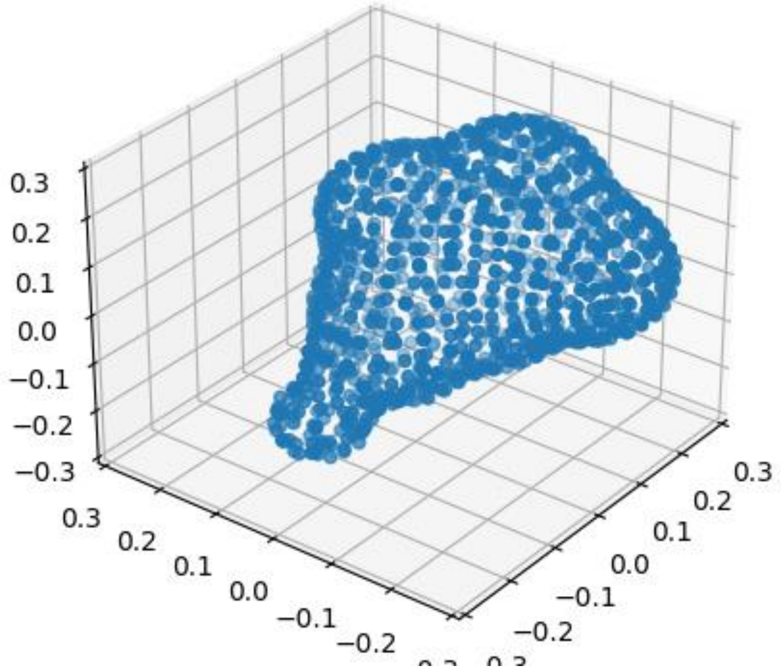
0.1
0.2
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



.2
−0.1
0.0
0.1
0.2
0.3
0.2
0.1
0.0
−0.1
−0.2

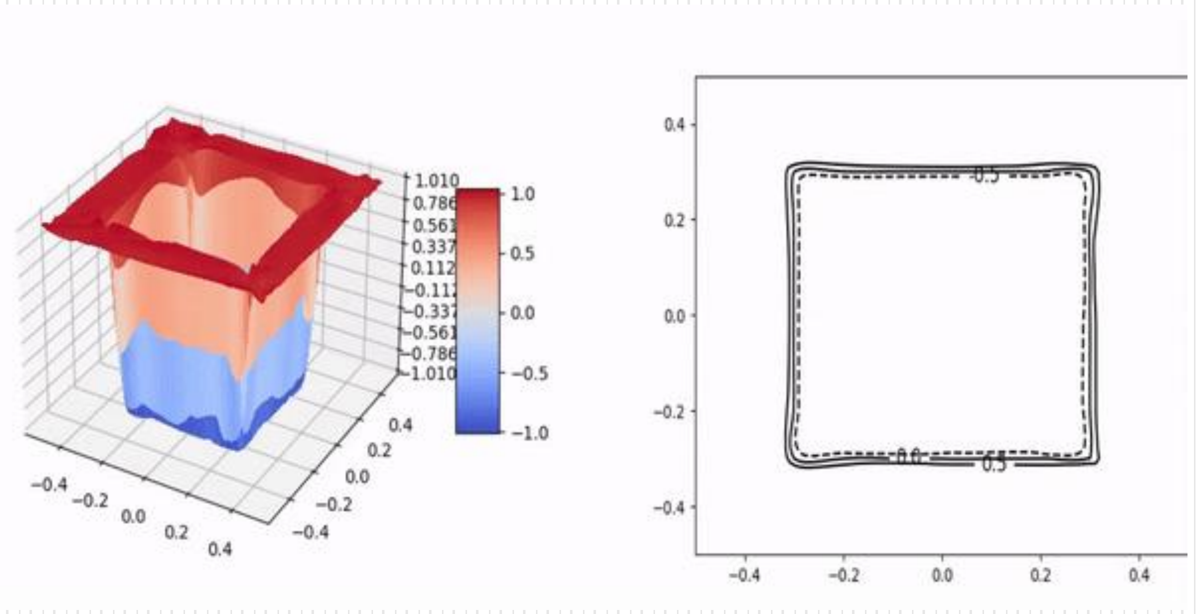


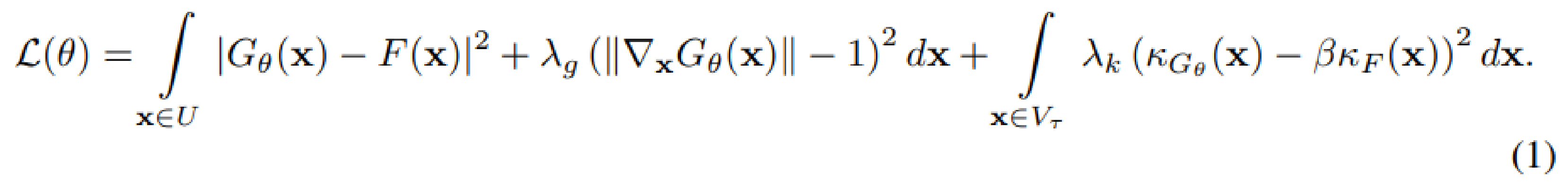
.2
-0.1
0.0
0.1
0.2
0.3
−0.2
−0.1
0.0
0.1
0.2
0.3



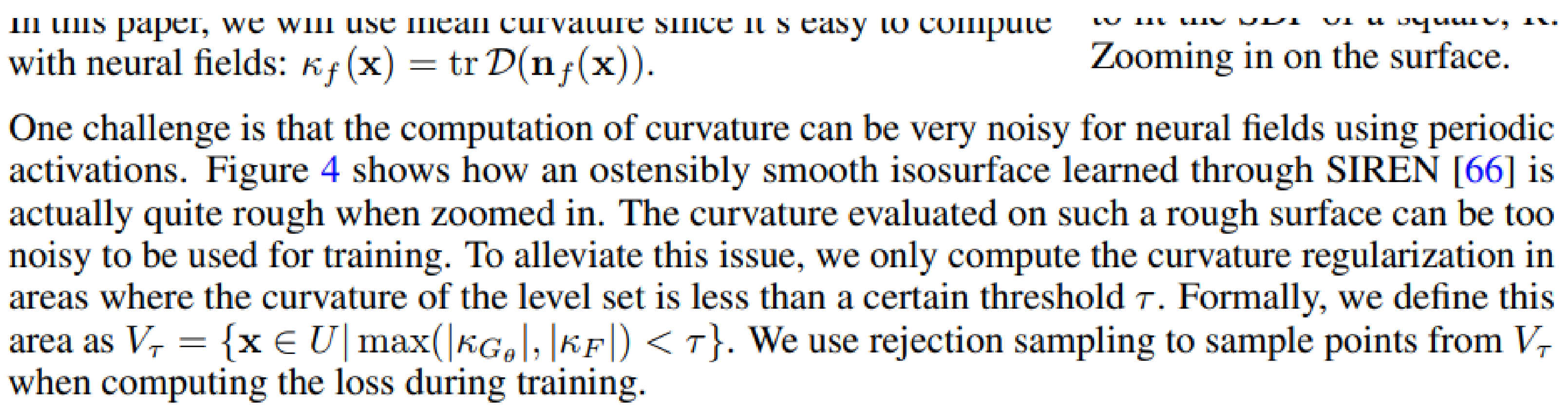
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
0.3
0.2
0.1
0.0
−0.1
−0.2

Transformationen





$$\mathcal{L}(\theta) = \int_{\mathbf{x}\in U} |G_\theta(\mathbf{x}) - F(\mathbf{x})|^2 + \lambda_g \left(\|\nabla_{\mathbf{x}} G_\theta(\mathbf{x})\| - 1\right)^2 d\mathbf{x} + \int_{\mathbf{x}\in V_\tau} \lambda_k \left(\kappa_{G_\theta}(\mathbf{x}) - \beta\kappa_F(\mathbf{x})\right)^2 d\mathbf{x}. \quad (1)$$

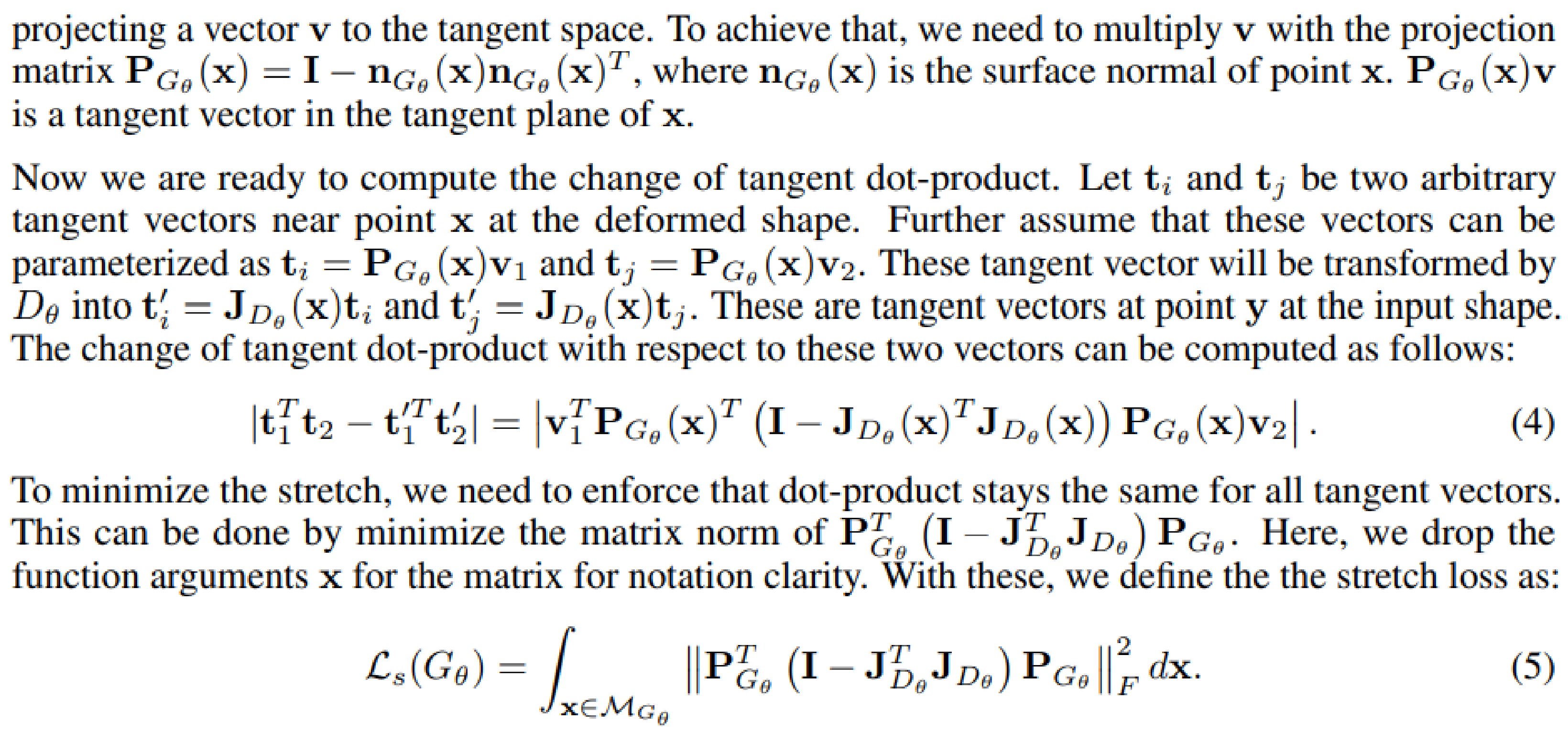


In this paper, we will use mean curvature since it's easy to compute with neural fields: $\kappa_f(\mathbf{x}) = \operatorname{tr}\mathcal{D}(\mathbf{n}_f(\mathbf{x}))$.

to fit the SDF of a square, R: Zooming in on the surface.

One challenge is that the computation of curvature can be very noisy for neural fields using periodic activations. Figure 4 shows how an ostensibly smooth isosurface learned through SIREN [66] is actually quite rough when zoomed in. The curvature evaluated on such a rough surface can be too noisy to be used for training. To alleviate this issue, we only compute the curvature regularization in areas where the curvature of the level set is less than a certain threshold τ. Formally, we define this area as $V_\tau = \{\mathbf{x} \in U \,|\, \max(|\kappa_{G_\theta}|, |\kappa_F|) < \tau\}$. We use rejection sampling to sample points from V_τ when computing the loss during training.

maybe



projecting a vector $\mathbf{v}$ to the tangent space. To achieve that, we need to multiply $\mathbf{v}$ with the projection matrix $\mathbf{P}_{G_\theta}(\mathbf{x}) = \mathbf{I} - \mathbf{n}_{G_\theta}(\mathbf{x})\mathbf{n}_{G_\theta}(\mathbf{x})^T$, where $\mathbf{n}_{G_\theta}(\mathbf{x})$ is the surface normal of point $\mathbf{x}$. $\mathbf{P}_{G_\theta}(\mathbf{x})\mathbf{v}$ is a tangent vector in the tangent plane of $\mathbf{x}$.

Now we are ready to compute the change of tangent dot-product. Let $\mathbf{t}_i$ and $\mathbf{t}_j$ be two arbitrary tangent vectors near point $\mathbf{x}$ at the deformed shape. Further assume that these vectors can be parameterized as $\mathbf{t}_i = \mathbf{P}_{G_\theta}(\mathbf{x})\mathbf{v}_1$ and $\mathbf{t}_j = \mathbf{P}_{G_\theta}(\mathbf{x})\mathbf{v}_2$. These tangent vector will be transformed by D_θ into $\mathbf{t}'_i = \mathbf{J}_{D_\theta}(\mathbf{x})\mathbf{t}_i$ and $\mathbf{t}'_j = \mathbf{J}_{D_\theta}(\mathbf{x})\mathbf{t}_j$. These are tangent vectors at point $\mathbf{y}$ at the input shape. The change of tangent dot-product with respect to these two vectors can be computed as follows:

$$|\mathbf{t}_1^T\mathbf{t}_2 - \mathbf{t}_1'^T\mathbf{t}_2'| = \left|\mathbf{v}_1^T\mathbf{P}_{G_\theta}(\mathbf{x})^T\left(\mathbf{I} - \mathbf{J}_{D_\theta}(\mathbf{x})^T\mathbf{J}_{D_\theta}(\mathbf{x})\right)\mathbf{P}_{G_\theta}(\mathbf{x})\mathbf{v}_2\right|. \tag{4}$$

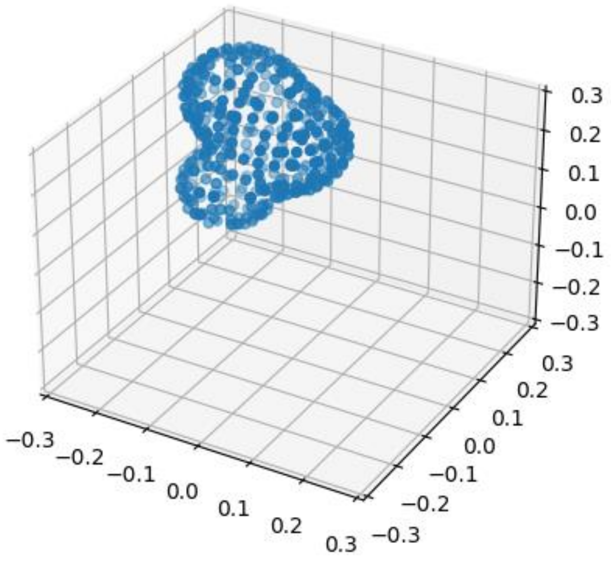
To minimize the stretch, we need to enforce that dot-product stays the same for all tangent vectors. This can be done by minimize the matrix norm of $\mathbf{P}_{G_\theta}^T\left(\mathbf{I} - \mathbf{J}_{D_\theta}^T\mathbf{J}_{D_\theta}\right)\mathbf{P}_{G_\theta}$. Here, we drop the function arguments $\mathbf{x}$ for the matrix for notation clarity. With these, we define the the stretch loss as:

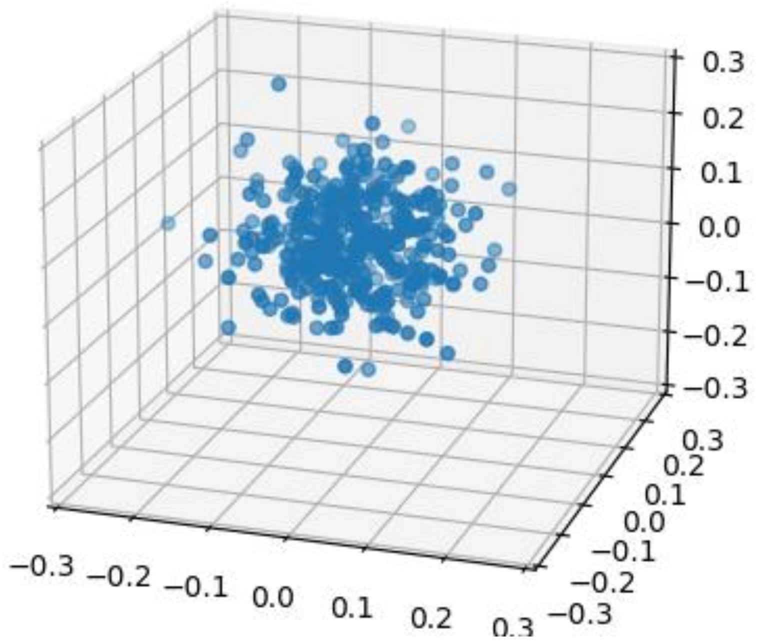
$$\mathcal{L}_s(G_\theta) = \int_{\mathbf{x}\in\mathcal{M}_{G_\theta}} \left\|\mathbf{P}_{G_\theta}^T\left(\mathbf{I} - \mathbf{J}_{D_\theta}^T\mathbf{J}_{D_\theta}\right)\mathbf{P}_{G_\theta}\right\|_F^2 d\mathbf{x}. \tag{5}$$

not

Autoencoder

- Does not work??





Loss functional

Chamfer distance is

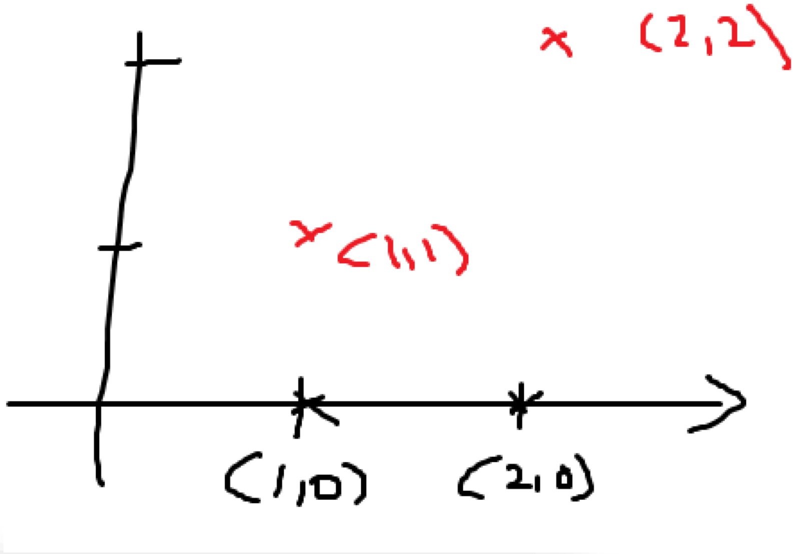
$$\mathrm{d_C}\left(\mathcal{X}_1, \mathcal{X}_2\right) = \frac{1}{2}\left(\mathrm{d}_{\overrightarrow{\mathrm{C}}}\left(\mathcal{X}_1, \mathcal{X}_2\right) + \mathrm{d}_{\overrightarrow{\mathrm{C}}}\left(\mathcal{X}_2, \mathcal{X}_1\right)\right) \tag{17}$$

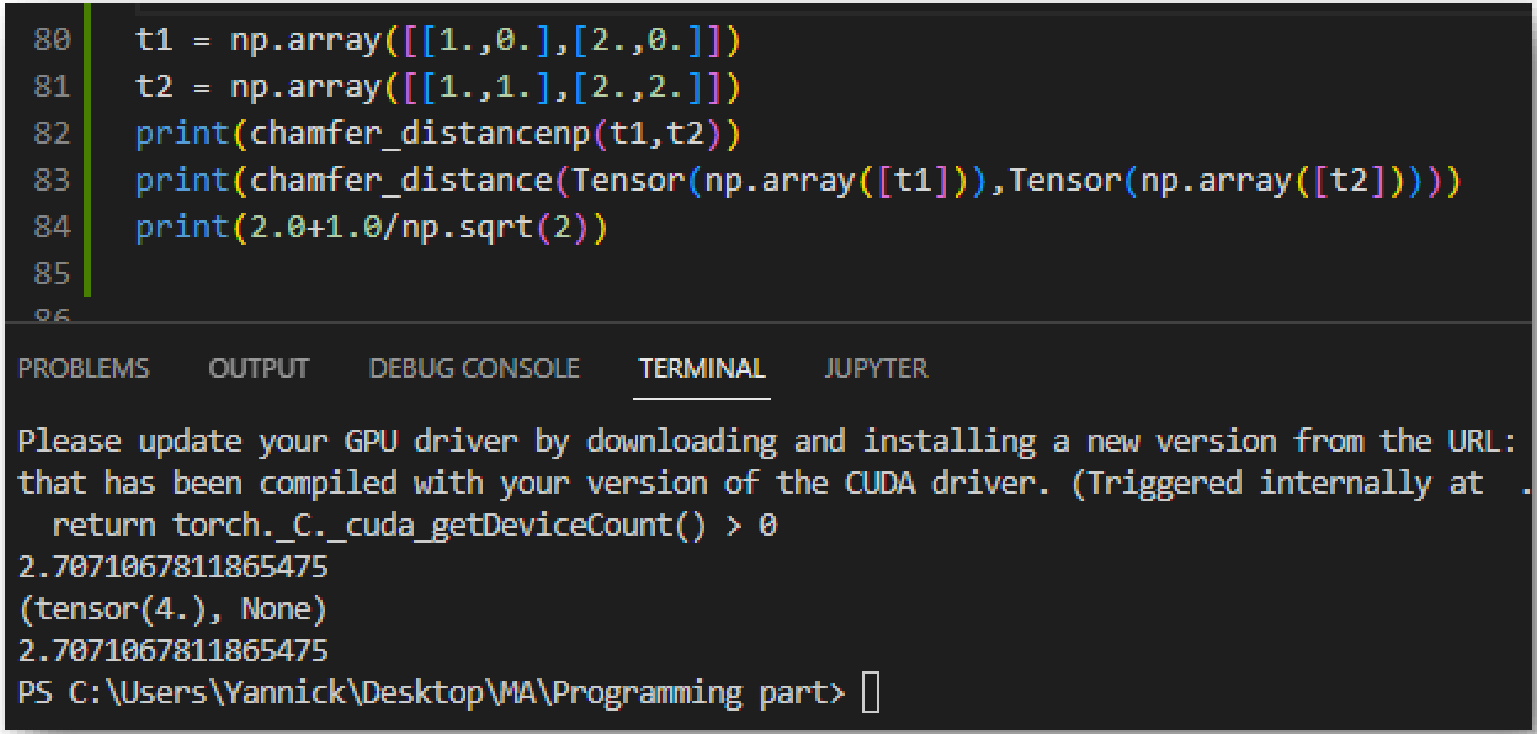
where

$$\mathrm{d}_{\overrightarrow{\mathrm{C}}}\left(\mathcal{X}_1, \mathcal{X}_2\right) = \frac{1}{|\mathcal{X}_1|} \sum_{\boldsymbol{x}_1 \in \mathcal{X}_1} \min_{\boldsymbol{x}_2 \in \mathcal{X}_2} \left\|\boldsymbol{x}_1 - \boldsymbol{x}_2\right\|^2 . \tag{18}$$

Cd(P1,P2) = Cd(P1^T,P2^T)

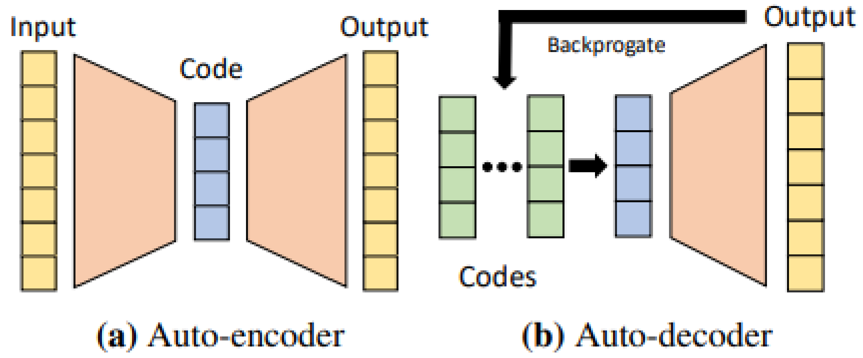
??





```
t1 = np.array([[1.,0.],[2.,0.]])
t2 = np.array([[1.,1.],[2.,2.]])
print(chamfer_distancenp(t1,t2))
print(chamfer_distance(Tensor(np.array([t1])),Tensor(np.array([t2]))))
print(2.0+1.0/np.sqrt(2))
```

```
Please update your GPU driver by downloading and installing a new version from the URL:
that has been compiled with your version of the CUDA driver. (Triggered internally at  .
  return torch._C._cuda_getDeviceCount() > 0
2.7071067811865475
(tensor(4.), None)
2.7071067811865475
PS C:\Users\Yannick\Desktop\MA\Programming part> 
```



$$\underset{\theta,\{z_i\}_{i=1}^N}{\arg\min} \sum_{i=1}^{N} \left(\sum_{j=1}^{K} \mathcal{L}(f_\theta(z_i, x_j), s_j) + \frac{1}{\sigma^2}||z_i||_2^2 \right). \quad (9)$$

Figure 4: Different from an auto-encoder whose latent code is produced by the encoder, an auto-decoder directly accepts a latent vector as an input. A randomly initialized latent vector is assigned to each data point in the beginning of training, and the latent vectors are optimized along with the decoder weights through standard backpropagation. During inference, decoder weights are fixed, and an optimal latent vector is estimated.

$$\hat{z} = \underset{z}{\arg\min} \sum_{(x_j, s_j)\in X} \mathcal{L}(f_\theta(z, x_j), s_j) + \frac{1}{\sigma^2}||z||_2^2 .$$

by a continuous SDF. Formally, for some shape indexed by i, f_θ is now a function of a latent code z_i and a query 3D location x, and outputs the shape's approximate SDF:

$$f_\theta(z_i, x) \approx SDF^i(x). \quad (5)$$

By conditioning the network output on a latent vector, this formulation allows modeling multiple SDFs with a single neural network. Given the decoding model f_θ, the continuous surface associated with a latent vector z is similarly represented with the decision boundary of $f_\theta(z, x)$, and the shape can again be discretized for visualization by, for ex-

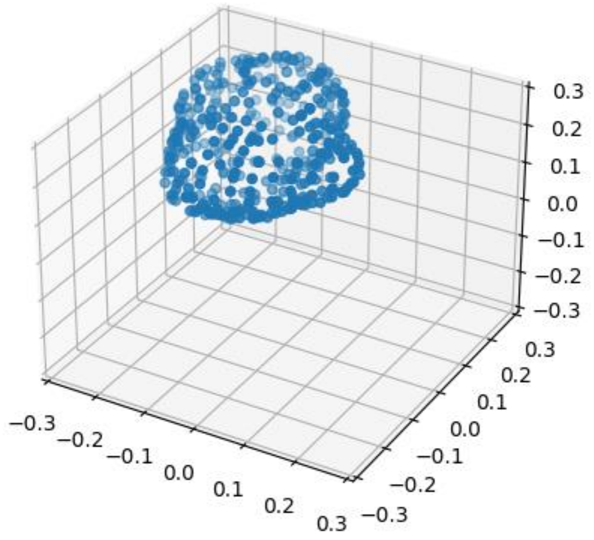
initialized randomly from $\mathcal{N}(0, 0.01^2)$.

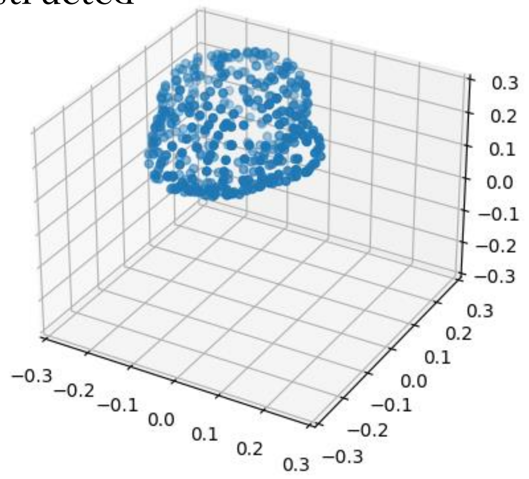
Note that while both VAE and the proposed auto-decoder formulation share the zero-mean Gaussian prior on the latent codes, we found that the the stochastic nature of the VAE optimization did not lead to good training results.

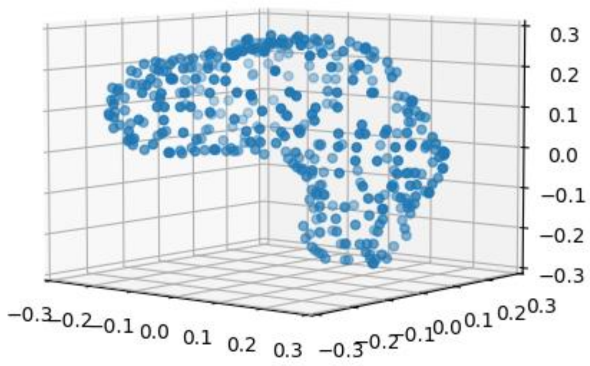
5. Data Preparation

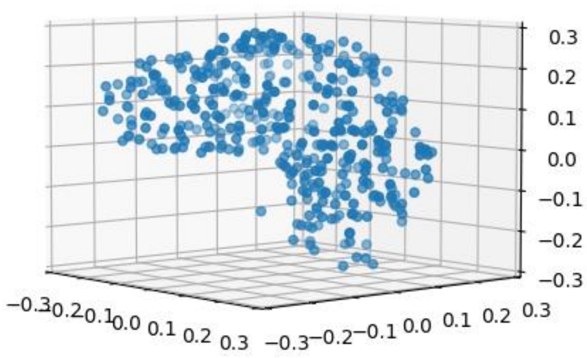
original

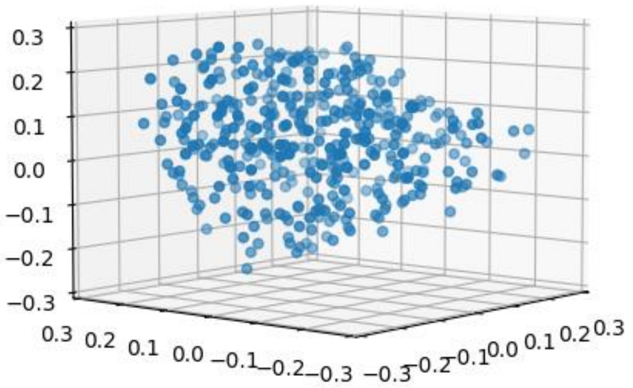
reconstructed

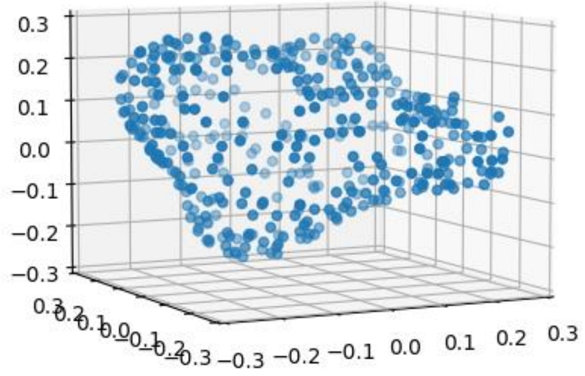


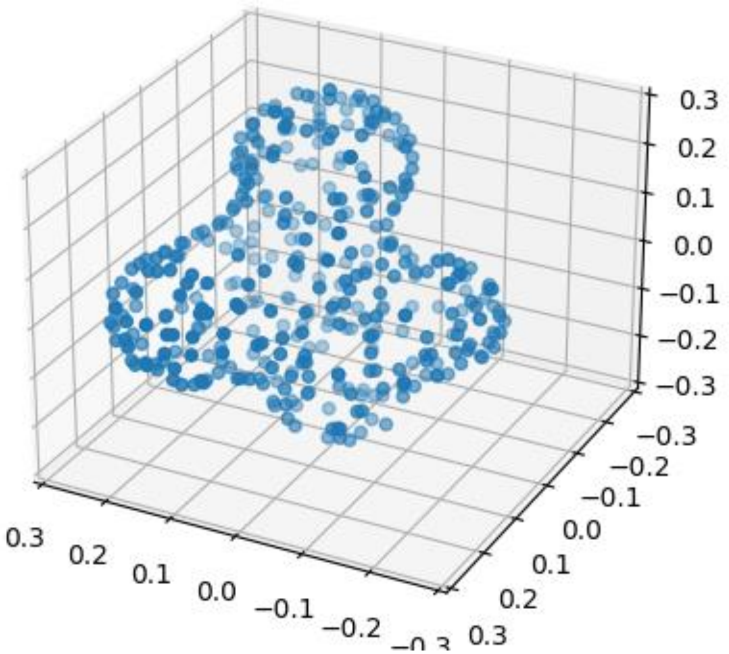


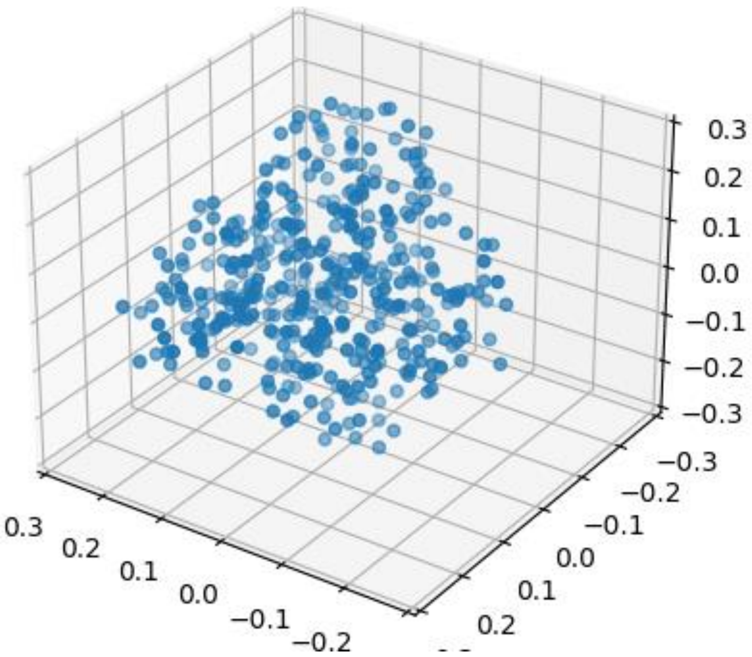


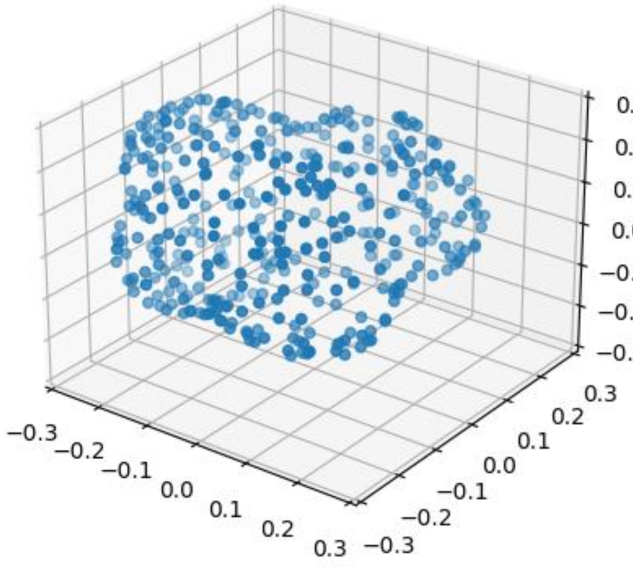
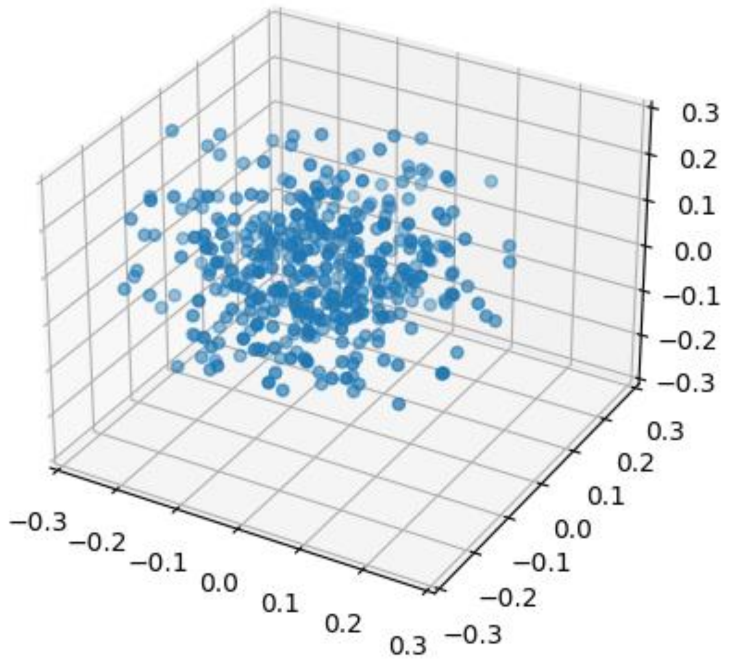
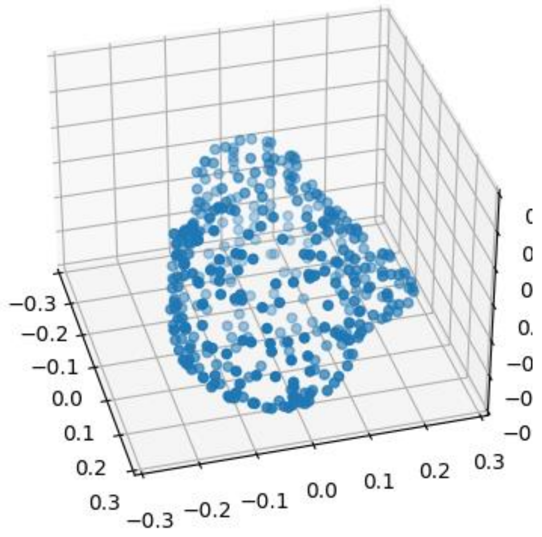
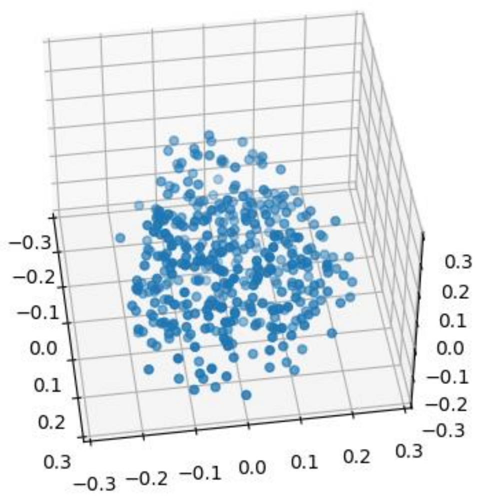


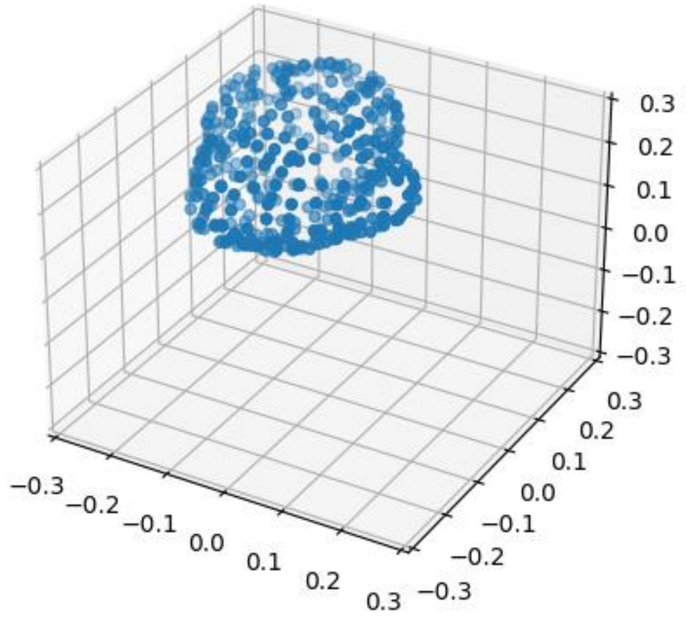
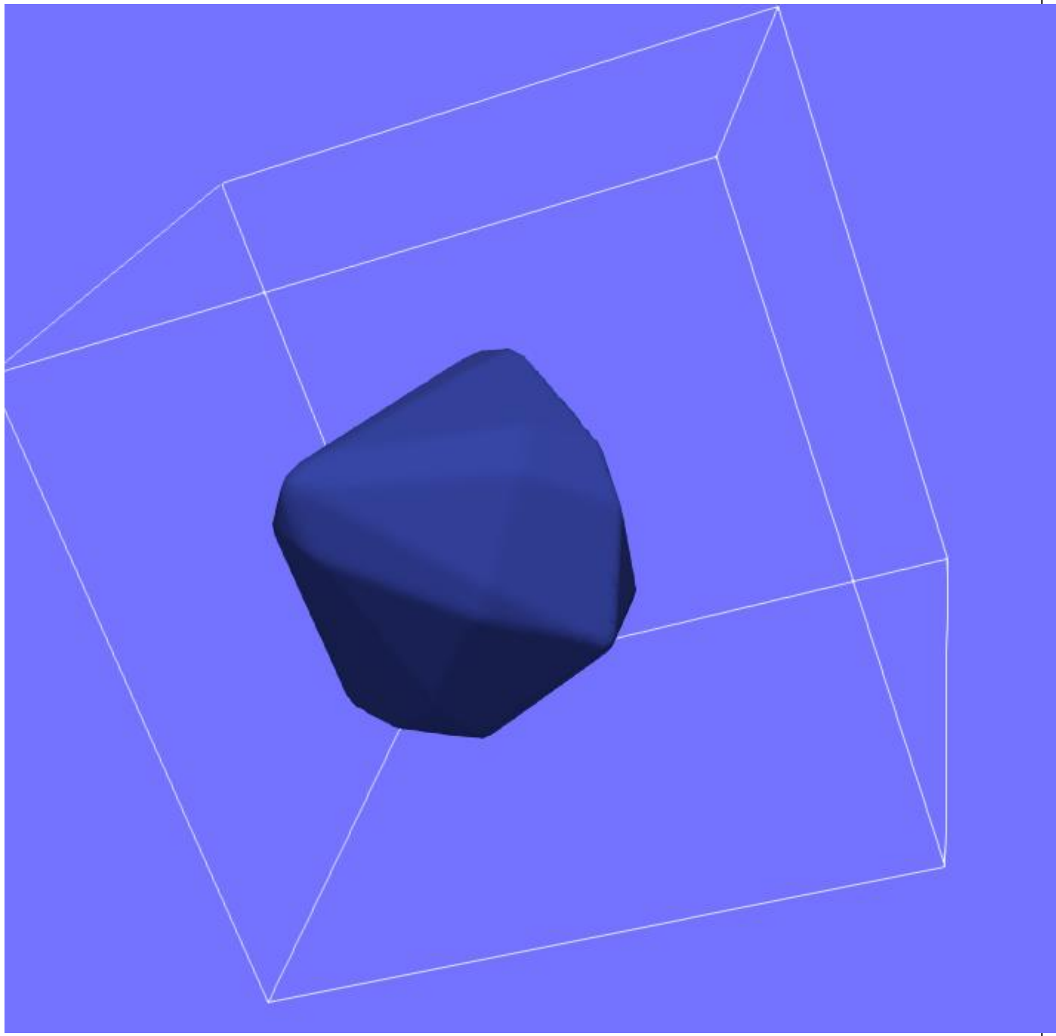

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

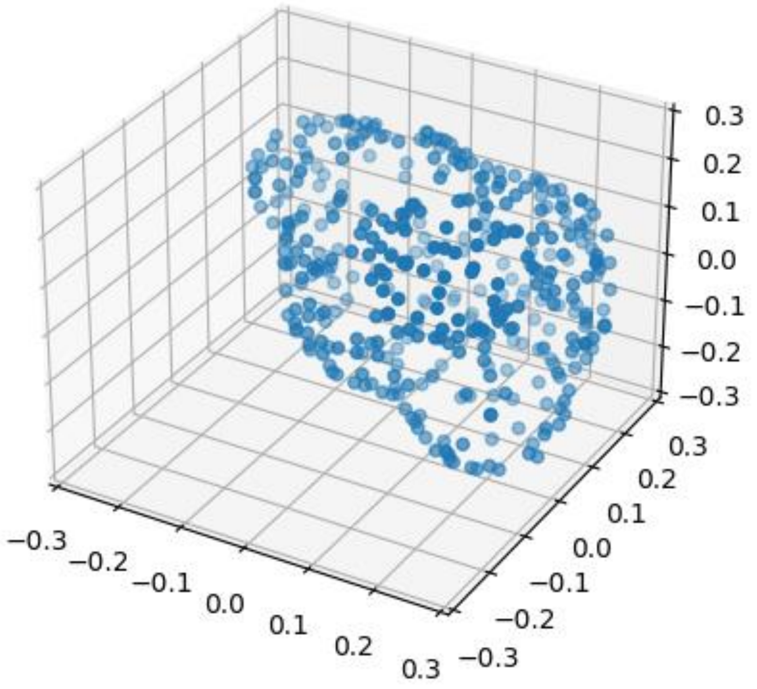

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

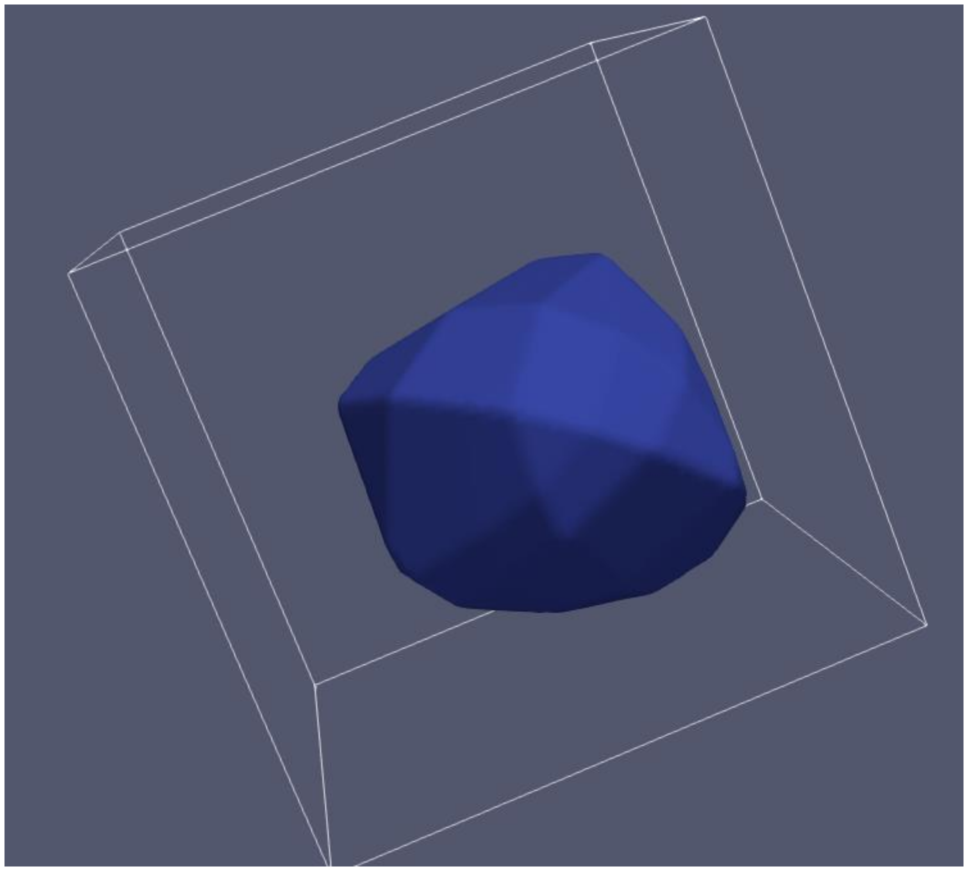

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2
−0.3 −0.2 −0.1 0.0 0.1 0.2

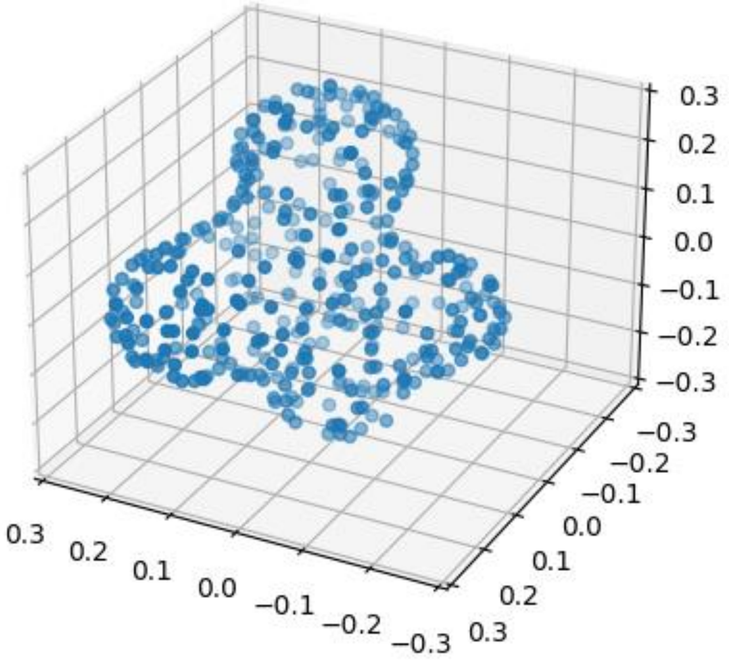





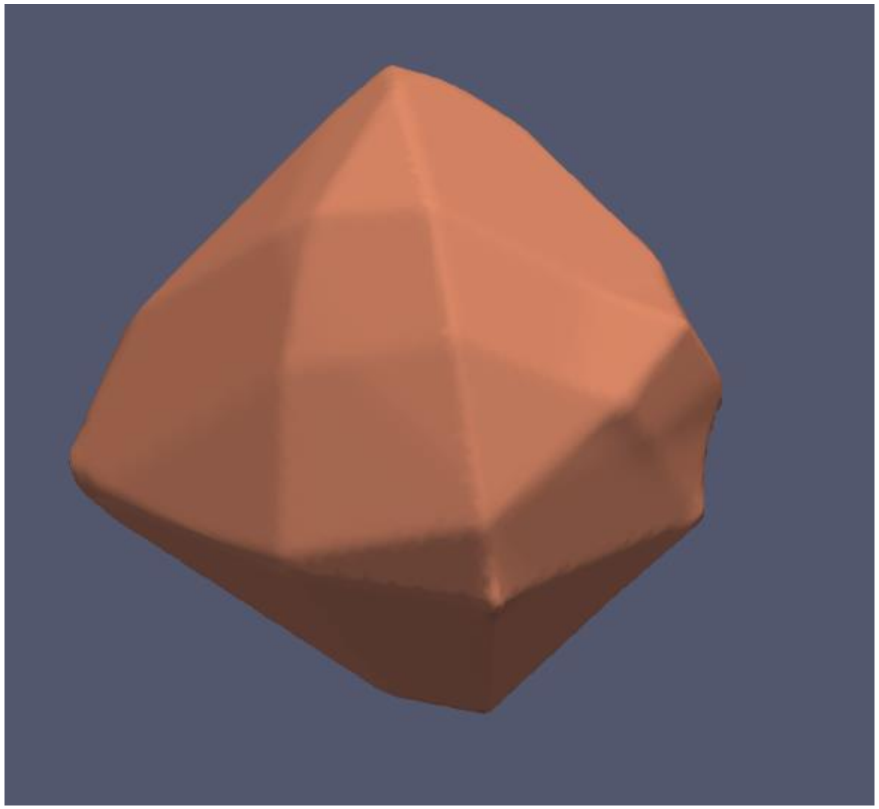

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


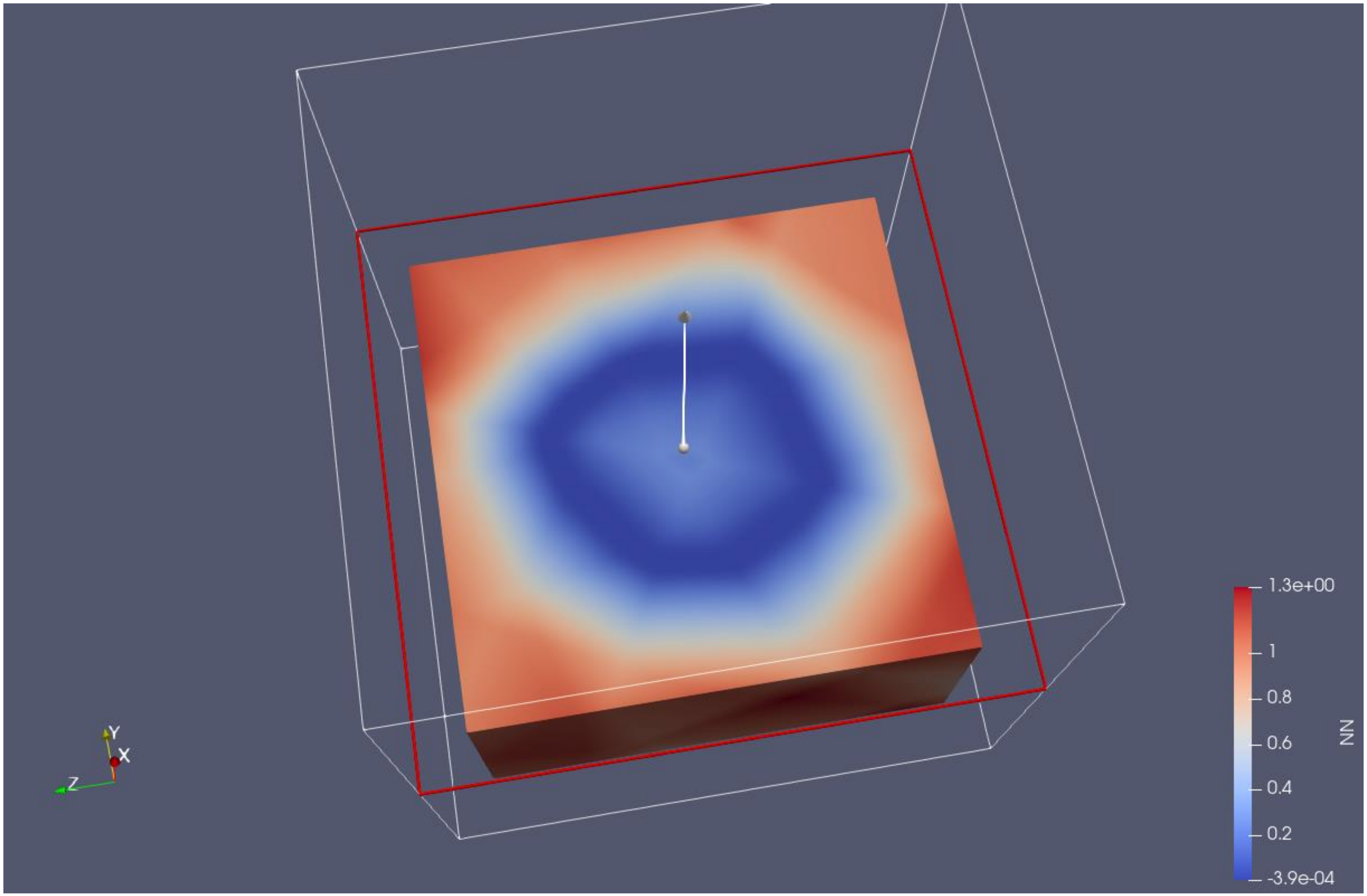


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

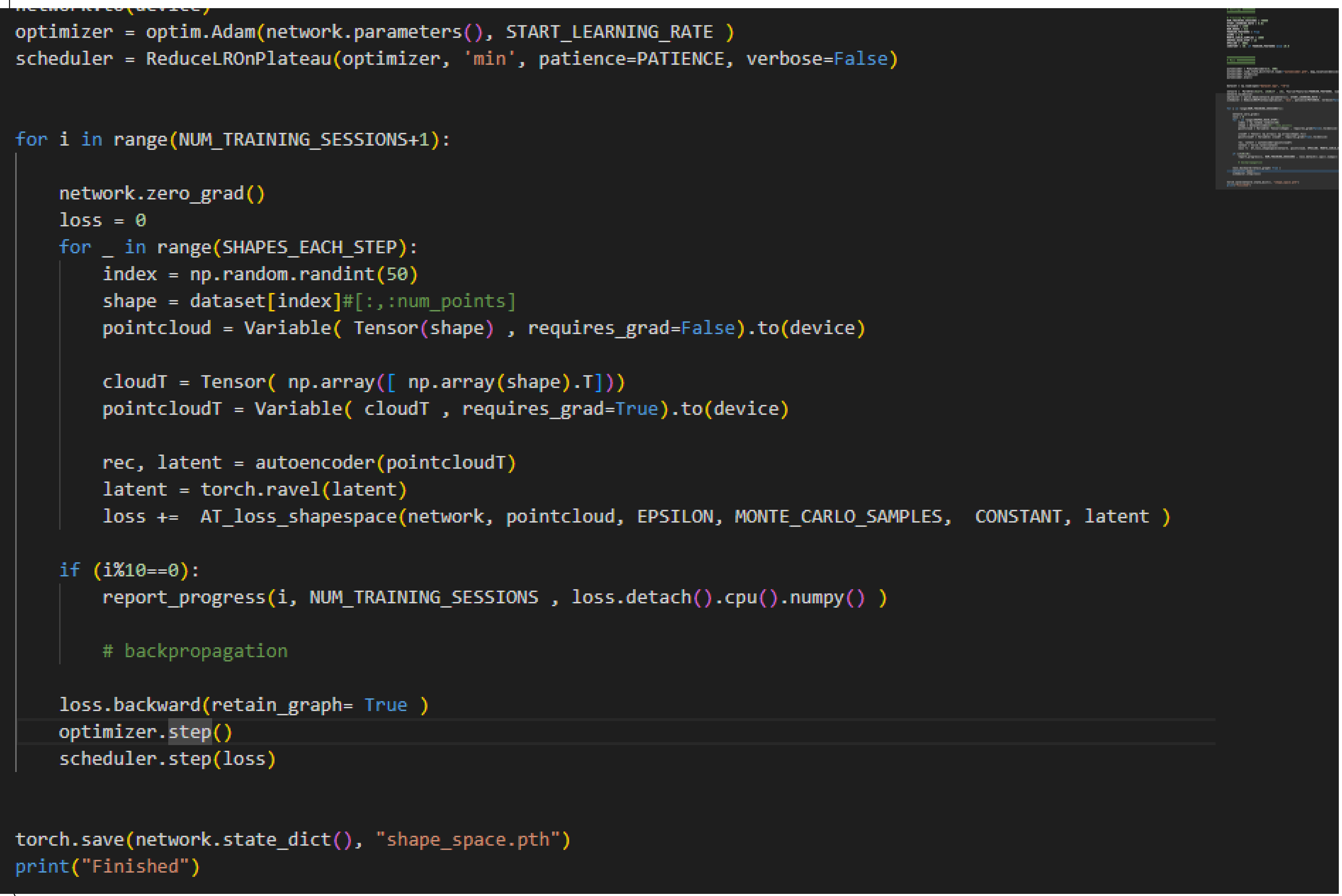



0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

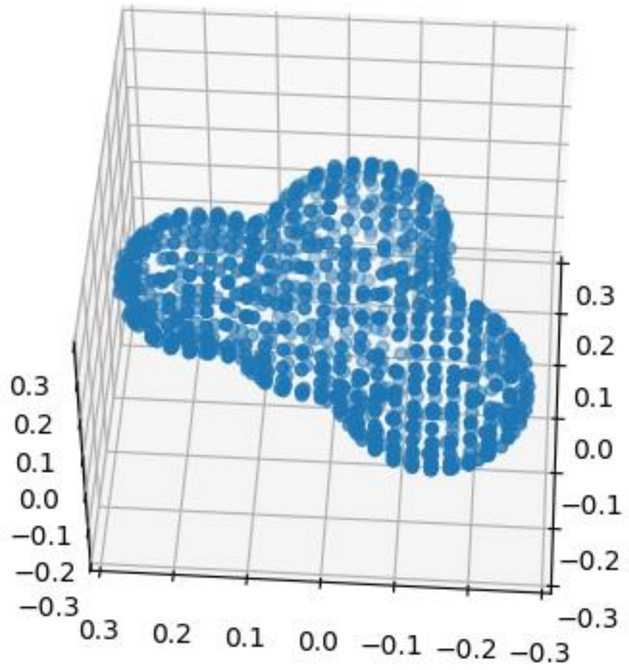


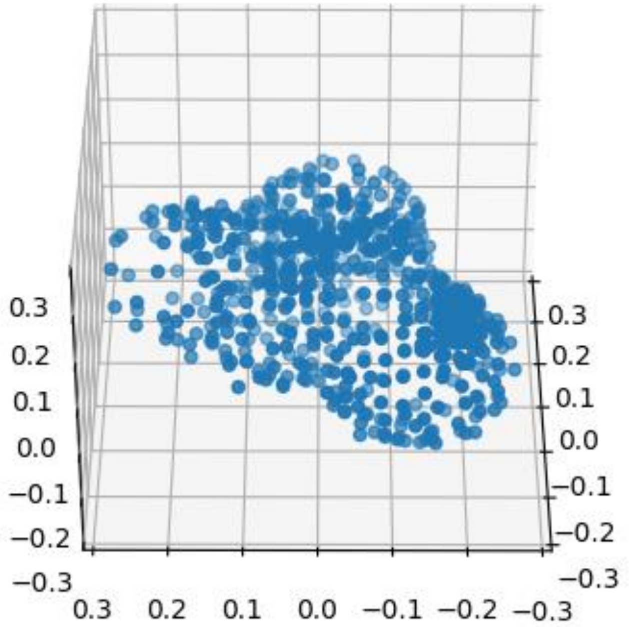


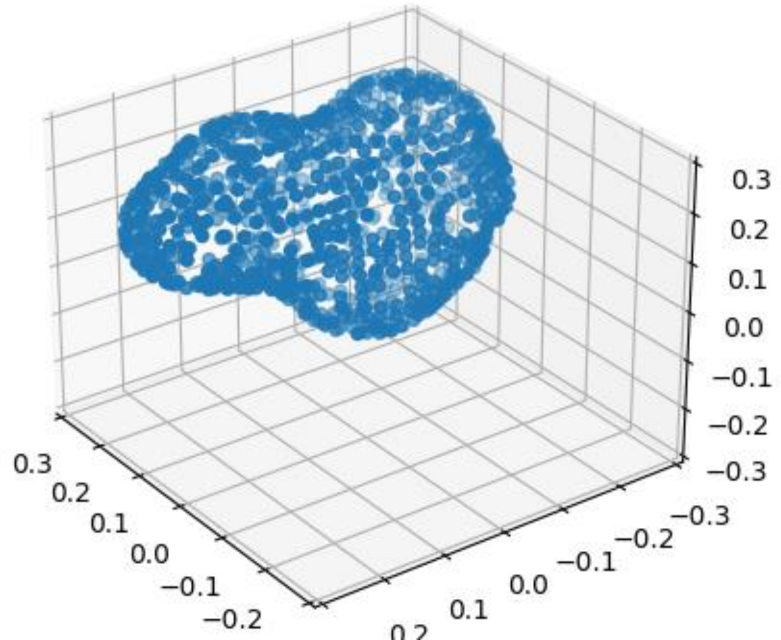
Y
X
Z
1.3e+00
1
0.8
0.6
0.4
0.2
-3.9e-04
NN

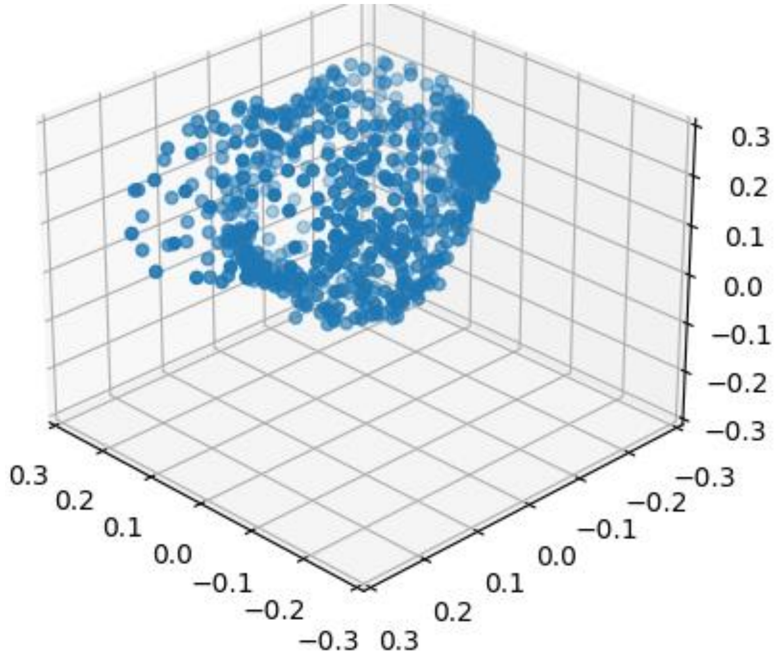


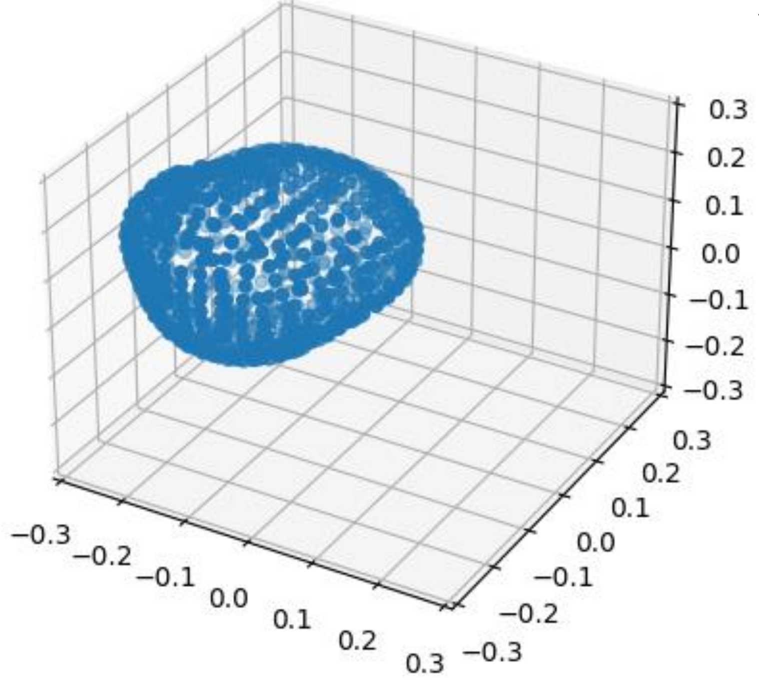
```
optimizer = optim.Adam(network.parameters(), START_LEARNING_RATE )
scheduler = ReduceLROnPlateau(optimizer, 'min', patience=PATIENCE, verbose=False)


for i in range(NUM_TRAINING_SESSIONS+1):

    network.zero_grad()
    loss = 0
    for _ in range(SHAPES_EACH_STEP):
        index = np.random.randint(50)
        shape = dataset[index]#[:,:num_points]
        pointcloud = Variable( Tensor(shape) , requires_grad=False).to(device)

        cloudT = Tensor( np.array([ np.array(shape).T]))
        pointcloudT = Variable( cloudT , requires_grad=True).to(device)

        rec, latent = autoencoder(pointcloudT)
        latent = torch.ravel(latent)
        loss +=  AT_loss_shapespace(network, pointcloud, EPSILON, MONTE_CARLO_SAMPLES,  CONSTANT, latent )

    if (i%10==0):
        report_progress(i, NUM_TRAINING_SESSIONS , loss.detach().cpu().numpy() )

        # backpropagation

    loss.backward(retain_graph= True )
    optimizer.step()
    scheduler.step(loss)


torch.save(network.state_dict(), "shape_space.pth")
print("Finished")
```

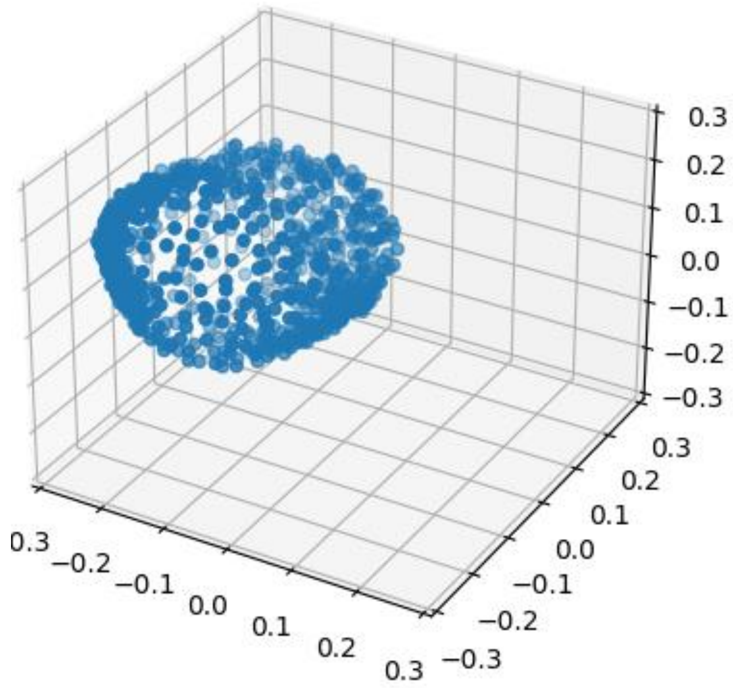

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

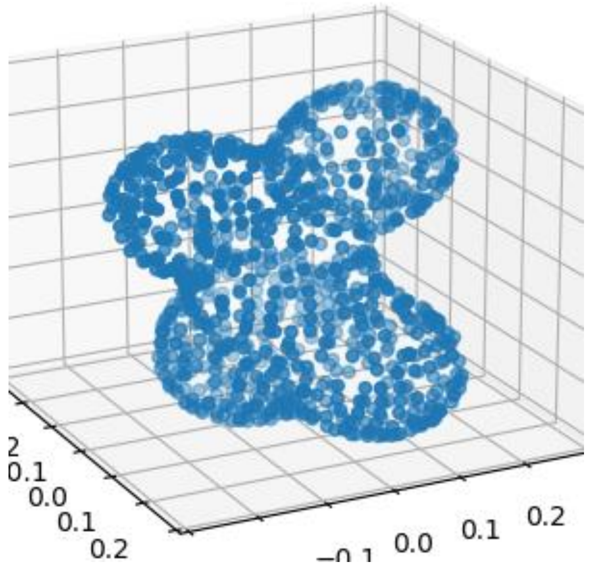

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

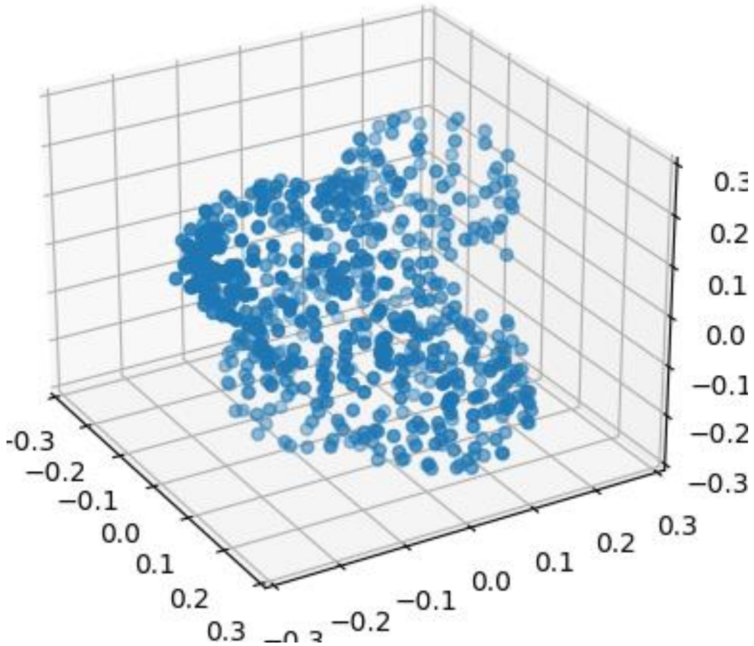

0.3
0.2
0.1
0.0
−0.1
−0.2
0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

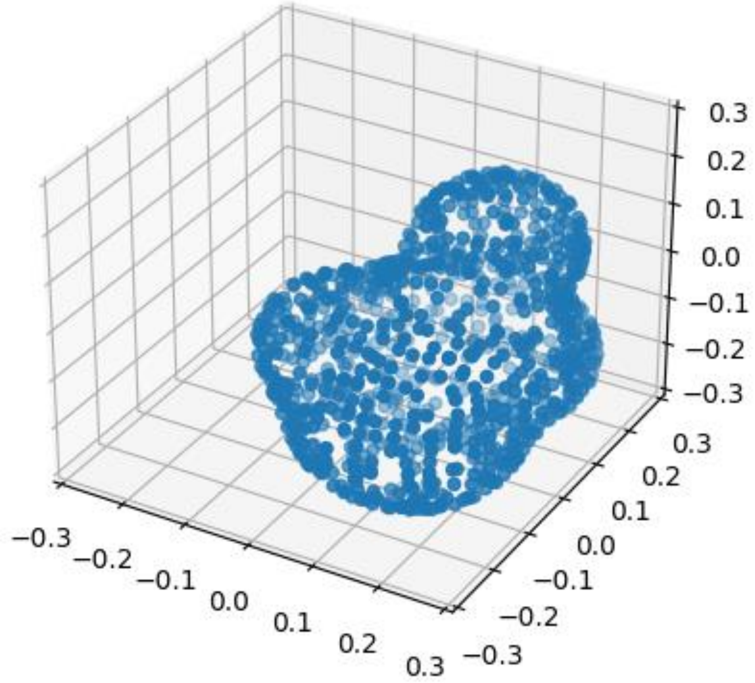

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

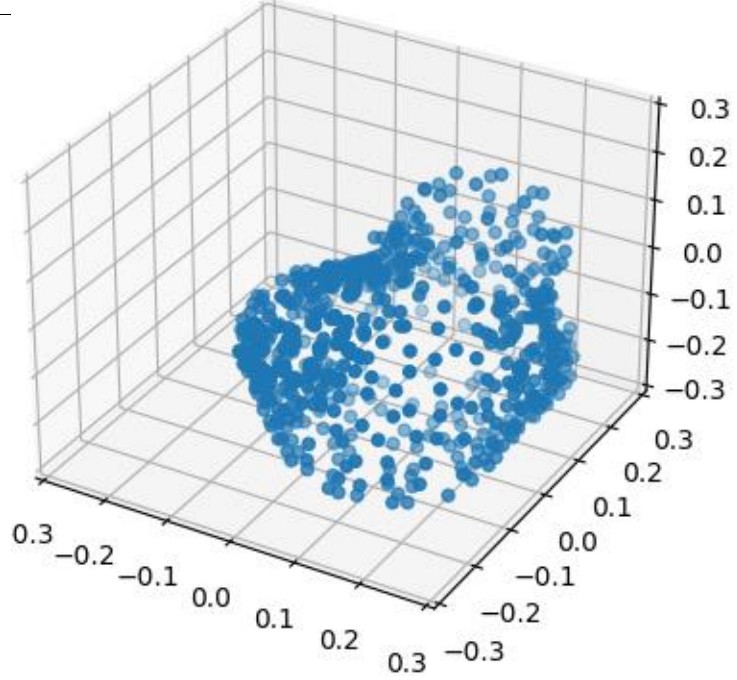

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


2
0.1
0.0
0.1
0.2
−0.1
0.0
0.1
0.2

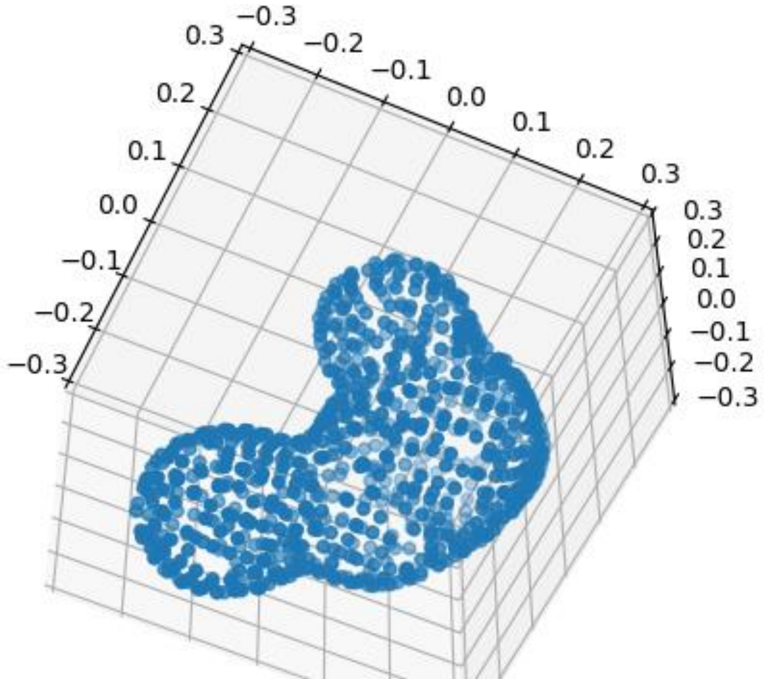

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
-0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2



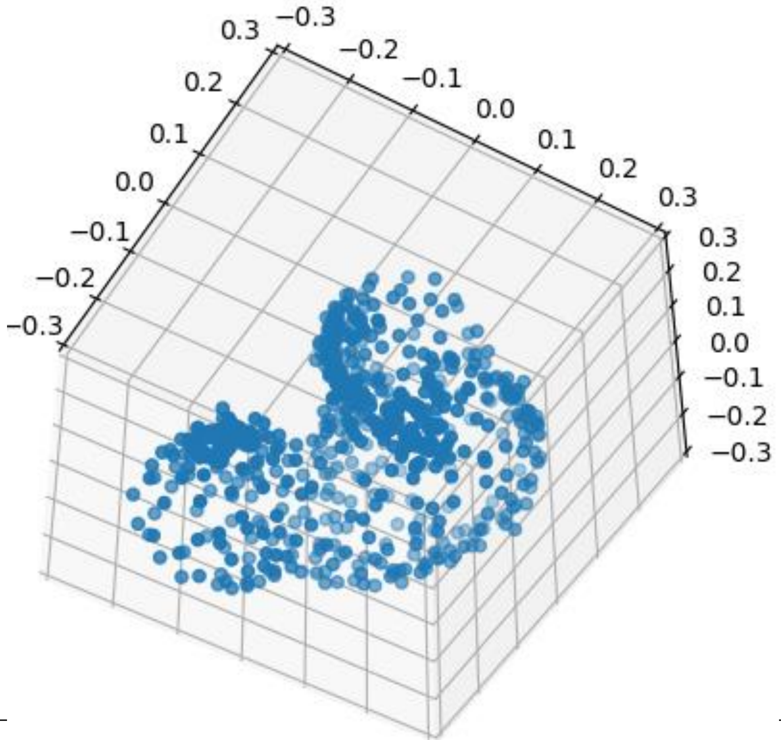
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



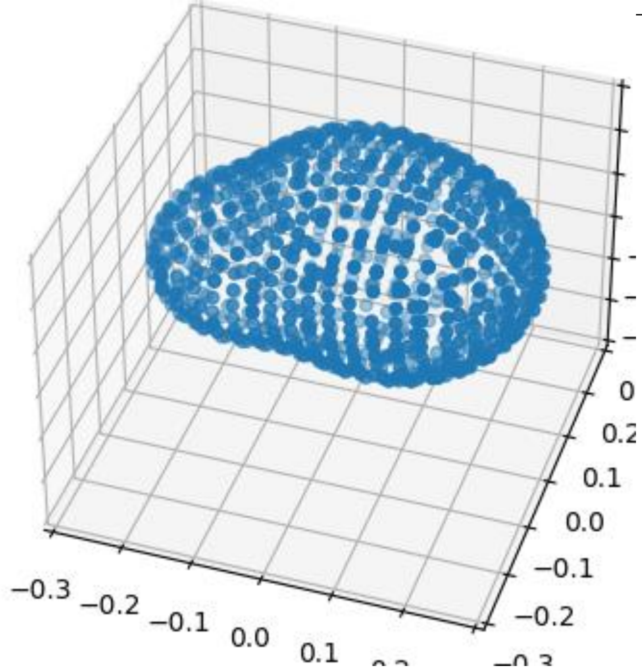
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



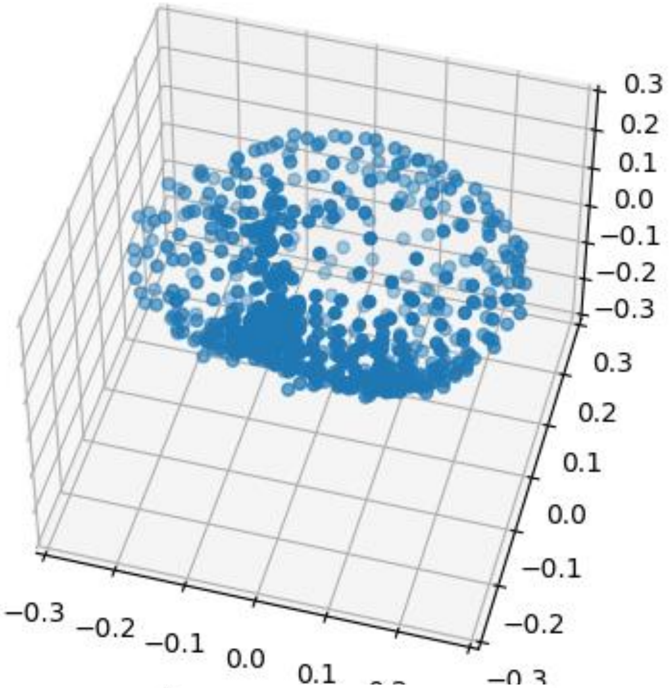
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3



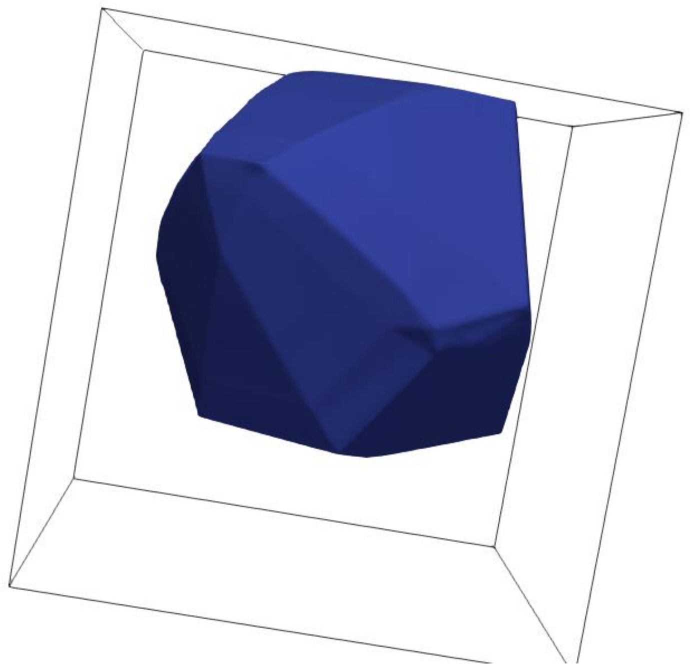
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

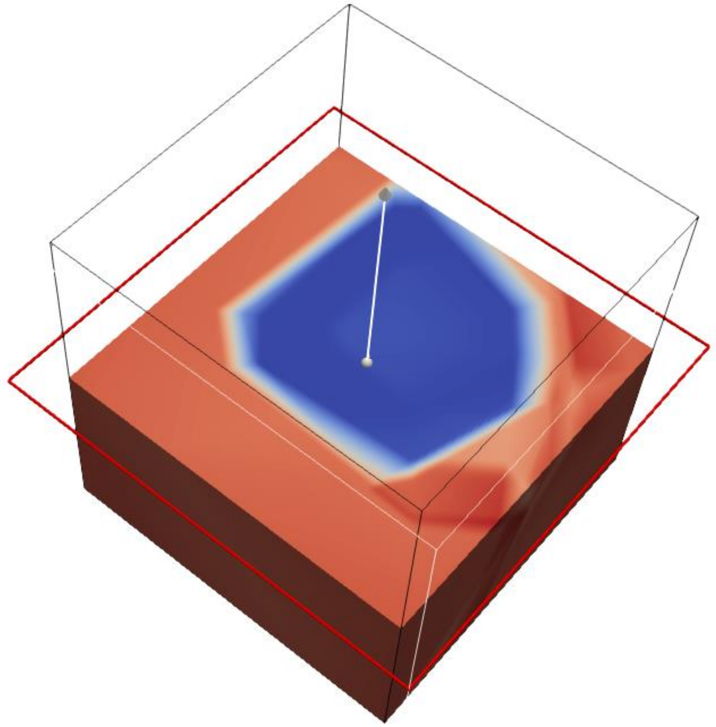


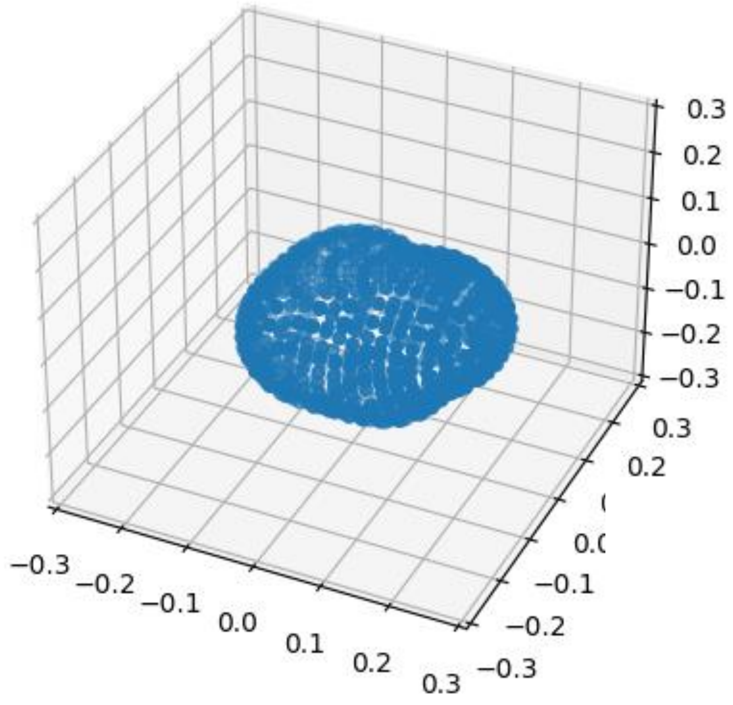
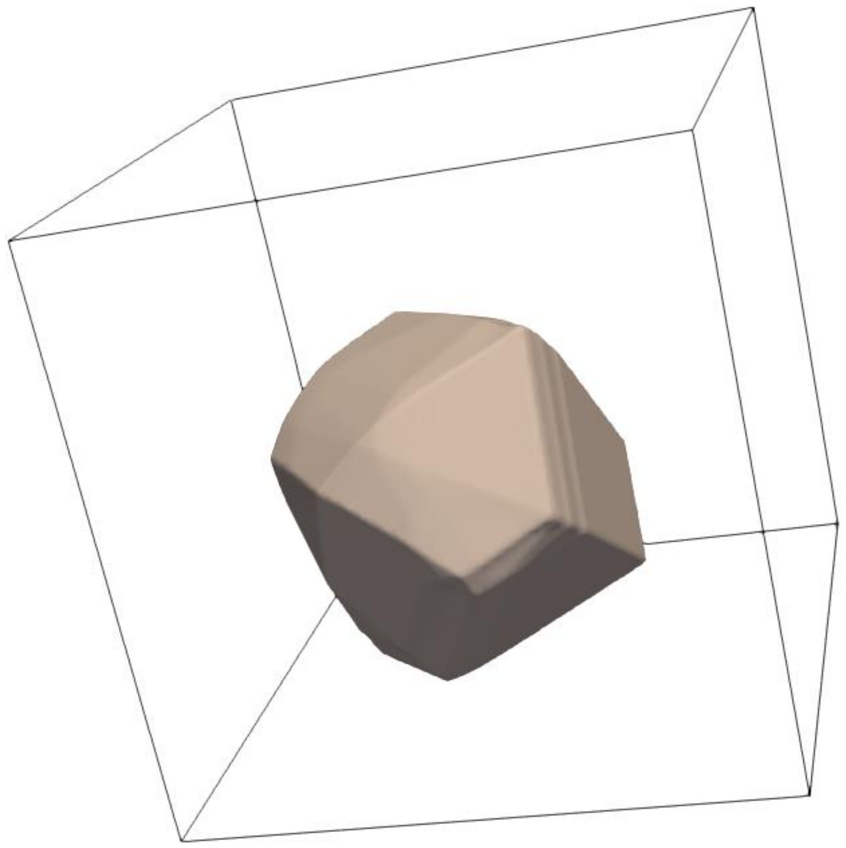
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.1
0.0
−0.1
−0.2

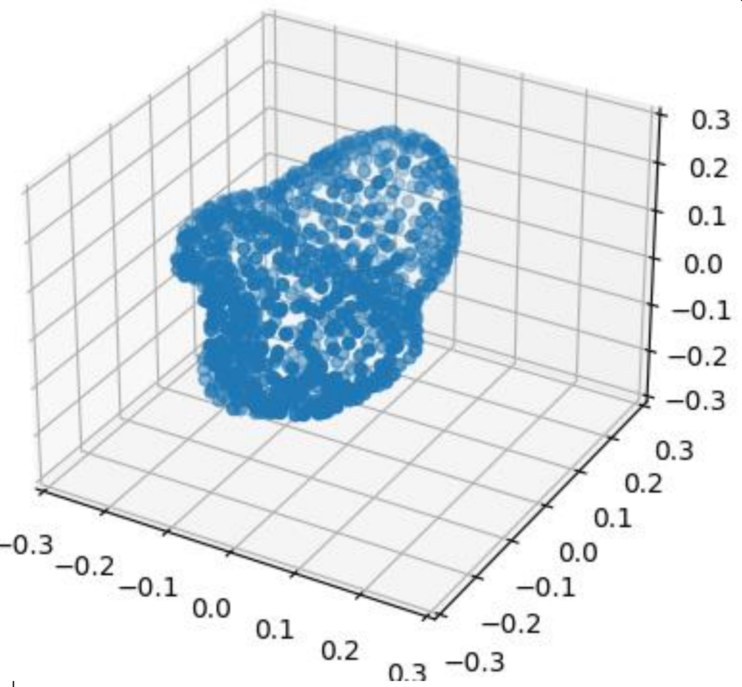


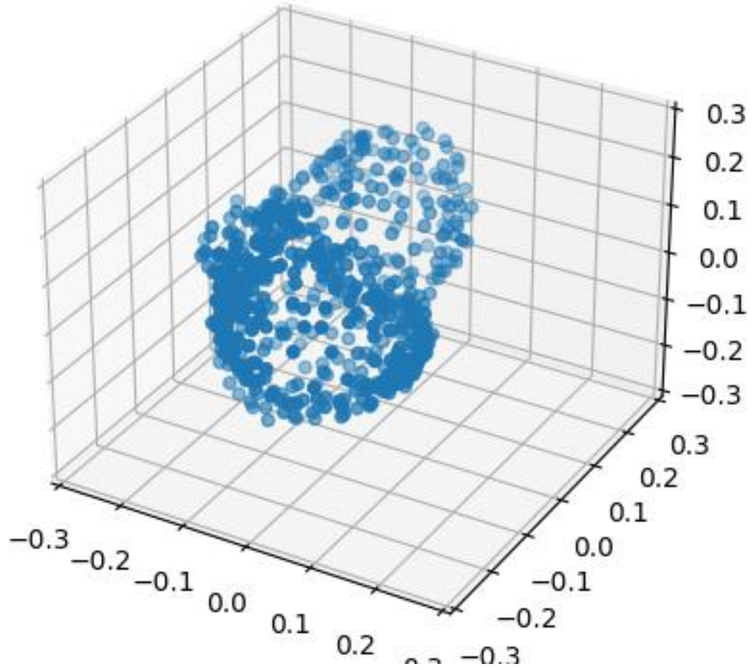
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.2
−0.1
0.0
0.1

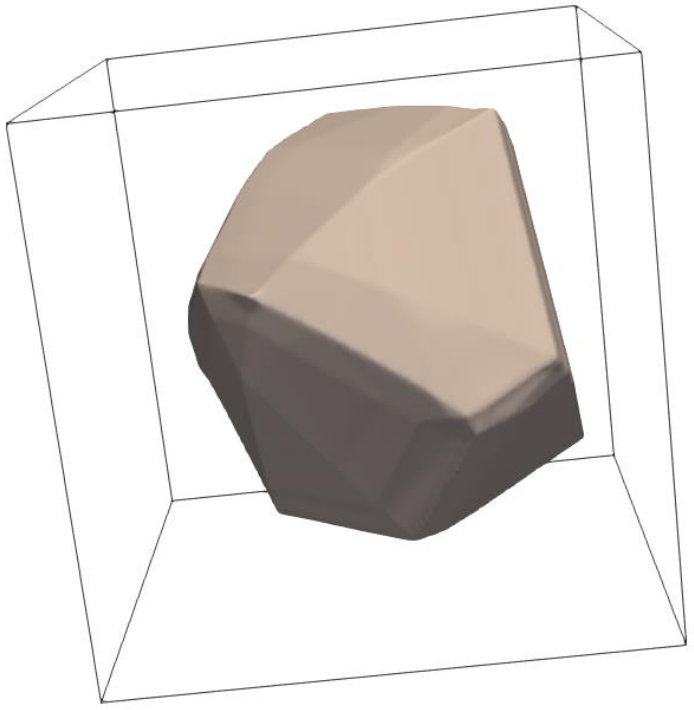


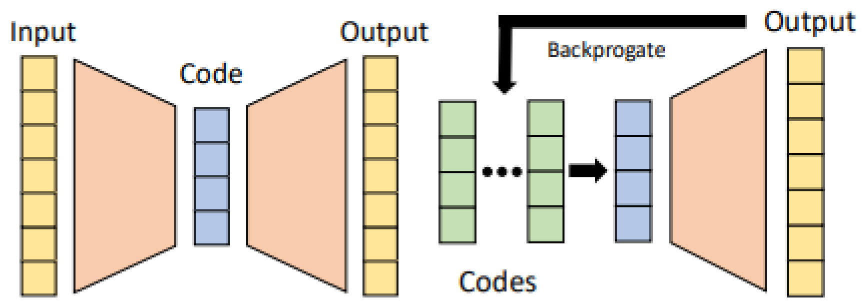



0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3



0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3





(a) Auto-encoder **(b)** Auto-decoder

Figure 4: Different from an auto-encoder whose latent code is produced by the encoder, an auto-decoder directly accepts a latent vector as an input. A randomly initialized latent vector is assigned to each data point in the beginning of training, and the latent vectors are optimized along with the decoder weights through standard backpropagation. During inference, decoder weights are fixed, and an optimal latent vector is estimated.

by a continuous SDF. Formally, for some shape indexed by i, f_θ is now a function of a latent code $\boldsymbol{z}_i$ and a query 3D location $\boldsymbol{x}$, and outputs the shape's approximate SDF:

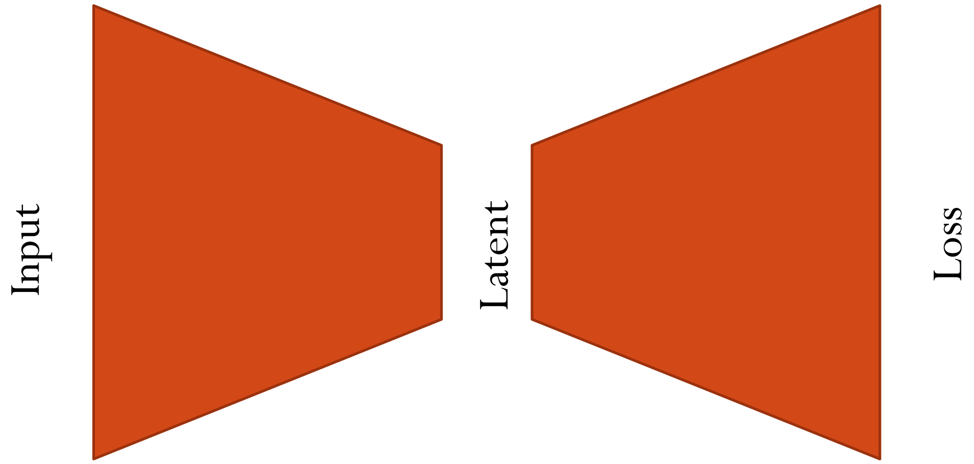
$$f_\theta(\boldsymbol{z}_i, \boldsymbol{x}) \approx SDF^i(\boldsymbol{x}). \tag{5}$$

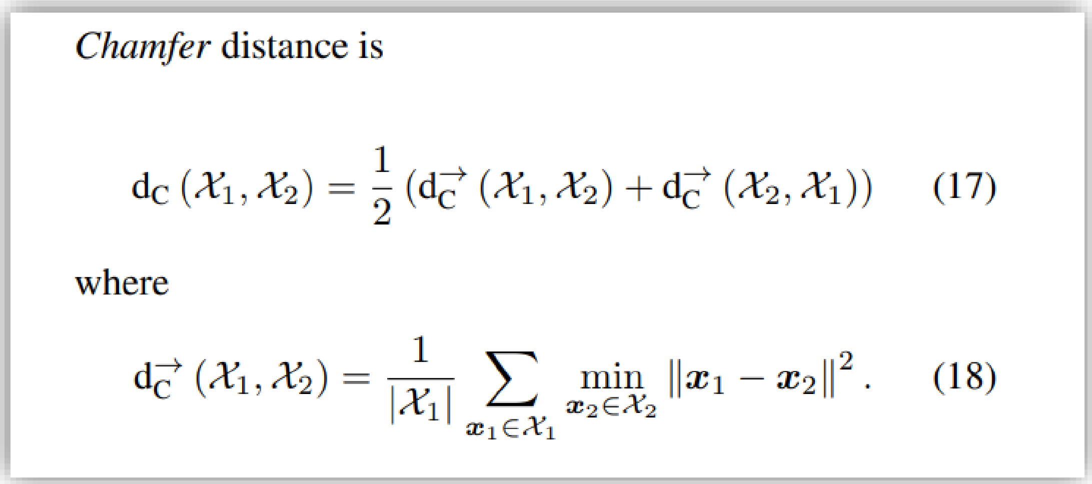
By conditioning the network output on a latent vector, this formulation allows modeling multiple SDFs with a single neural network. Given the decoding model f_θ, the continuous surface associated with a latent vector $\boldsymbol{z}$ is similarly represented with the decision boundary of $f_\theta(\boldsymbol{z}, \boldsymbol{x})$, and the shape can again be discretized for visualization by, for ex-

$$\underset{\theta,\{\boldsymbol{z}_i\}_{i=1}^N}{\arg\min} \sum_{i=1}^N \left(\sum_{j=1}^K \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}_i||_2^2 \right). \quad (9)$$

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z}}{\arg\min} \sum_{(\boldsymbol{x}_j, \boldsymbol{s}_j)\in X} \mathcal{L}(f_\theta(\boldsymbol{z}, \boldsymbol{x}_j), s_j) + \frac{1}{\sigma^2}||\boldsymbol{z}||_2^2. \quad (1$$

Shape Space learning: AE

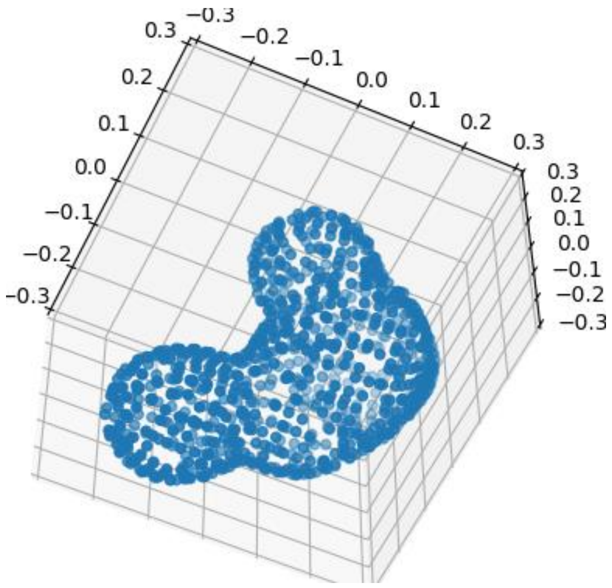


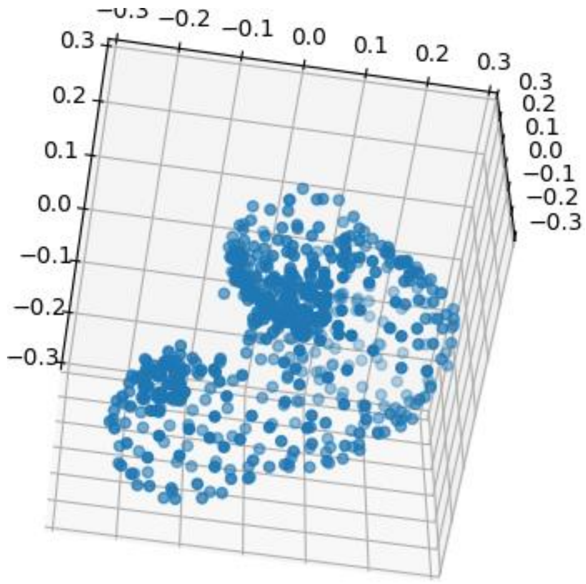

Chamfer distance is

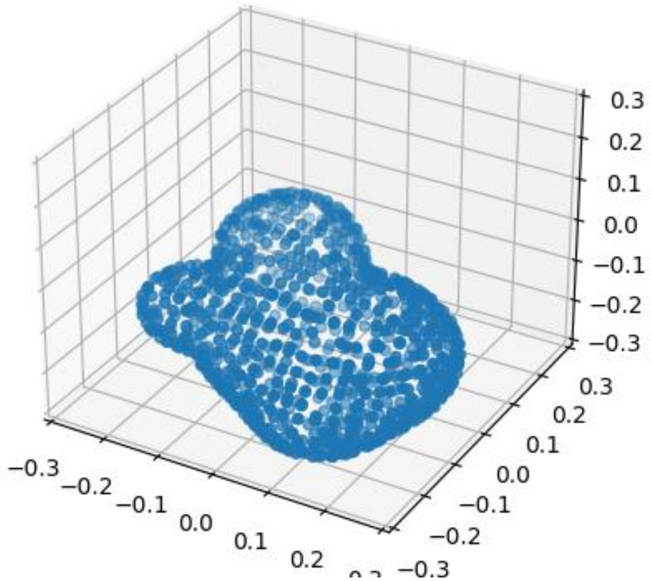
$$\mathrm{d_C}(\mathcal{X}_1, \mathcal{X}_2) = \frac{1}{2}\left(\mathrm{d}_{\mathrm{C}}^{\rightarrow}(\mathcal{X}_1, \mathcal{X}_2) + \mathrm{d}_{\mathrm{C}}^{\rightarrow}(\mathcal{X}_2, \mathcal{X}_1)\right) \quad (17)$$

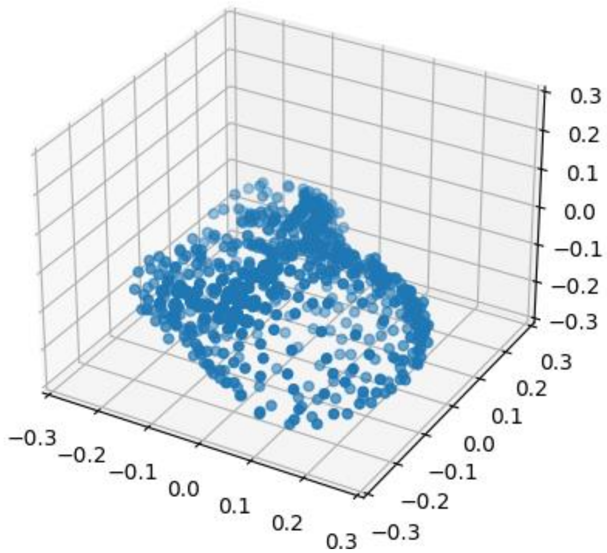
where

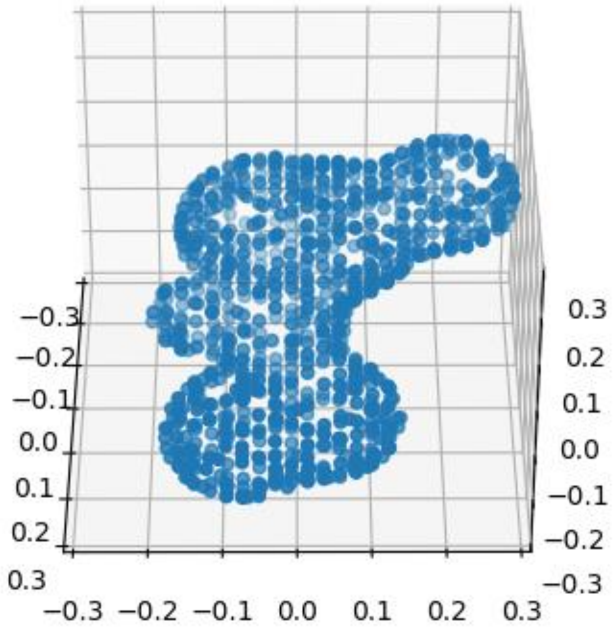
$$\mathrm{d}_{\mathrm{C}}^{\rightarrow}(\mathcal{X}_1, \mathcal{X}_2) = \frac{1}{|\mathcal{X}_1|}\sum_{\boldsymbol{x}_1 \in \mathcal{X}_1} \min_{\boldsymbol{x}_2 \in \mathcal{X}_2} \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 . \quad (18)$$

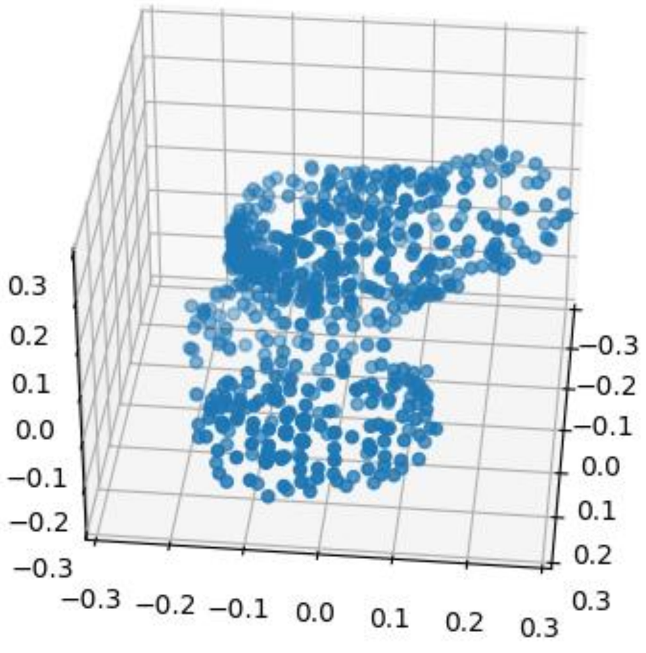

−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

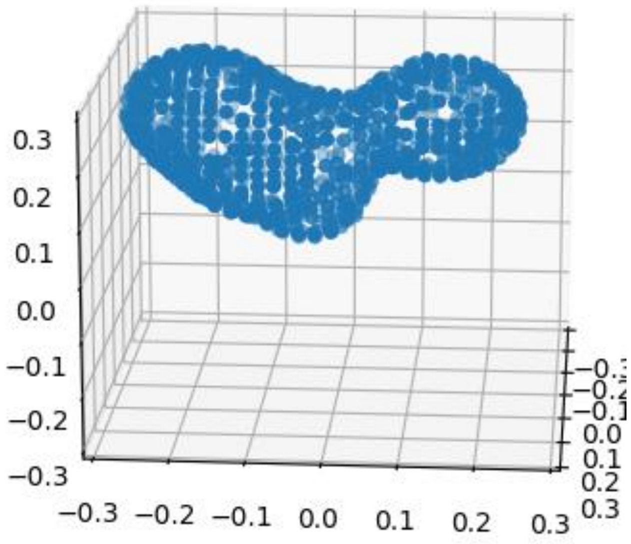

−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

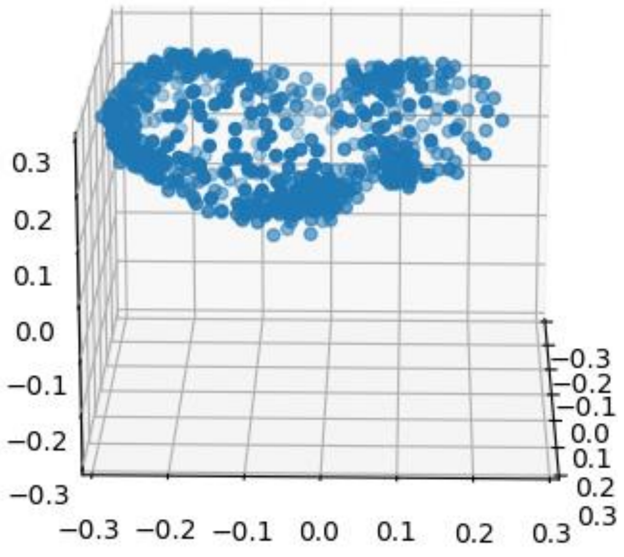

−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

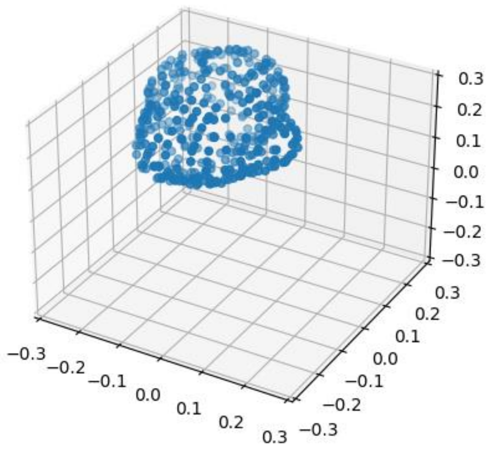

−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3


−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3

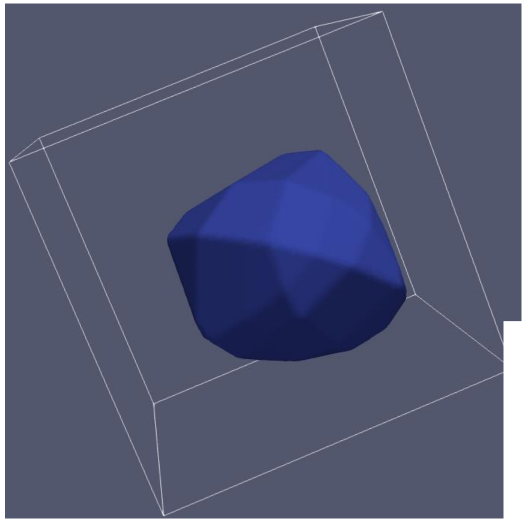
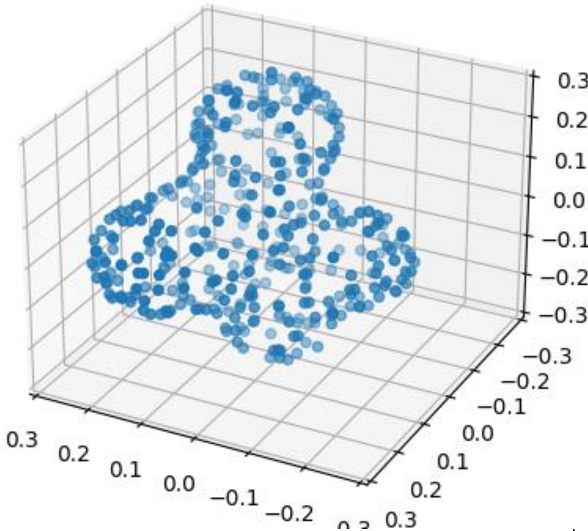

0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3


0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

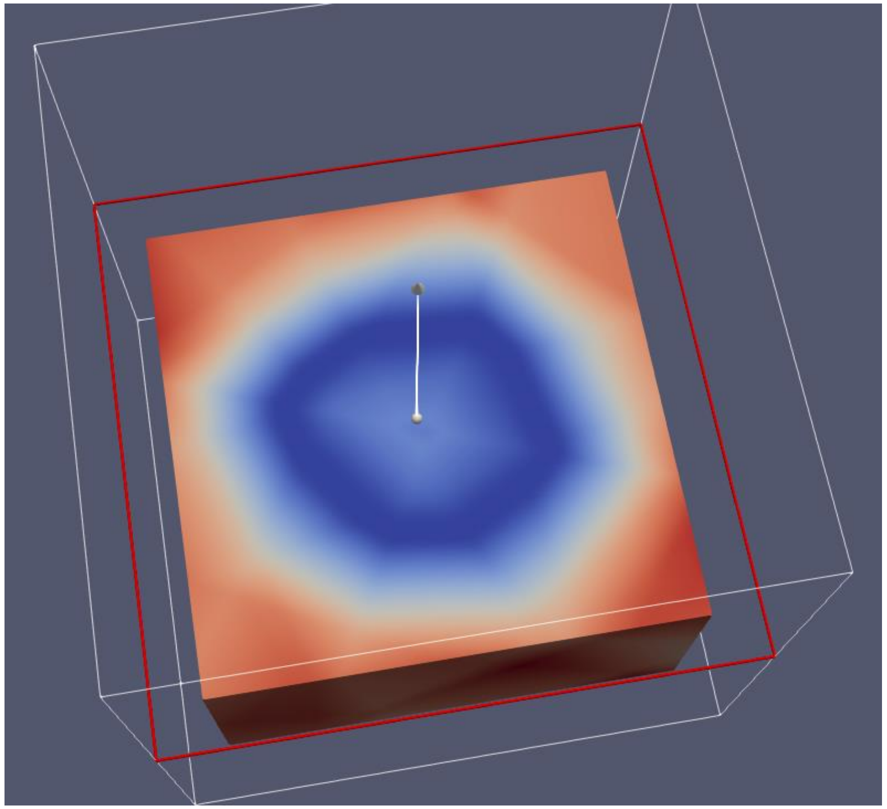
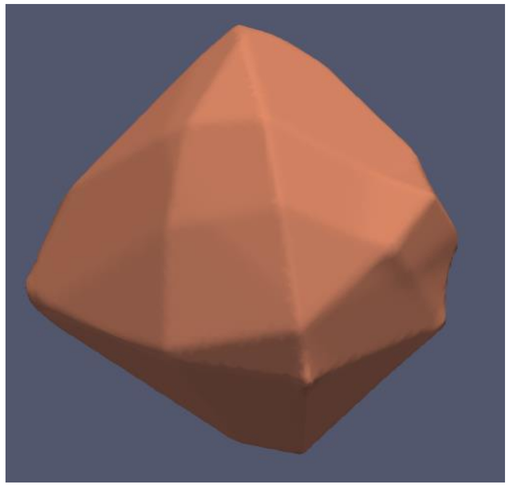

0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3



0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

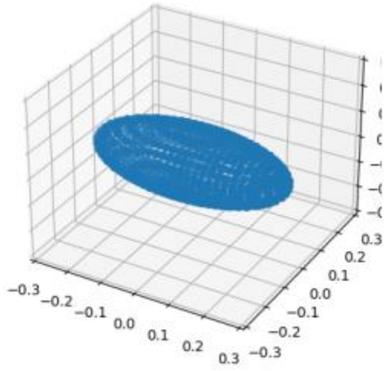



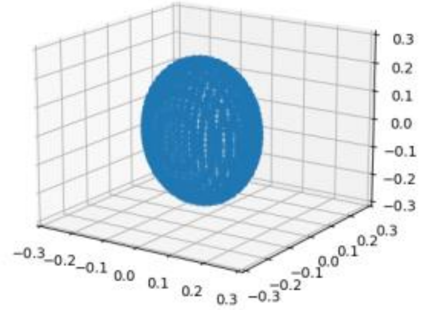
Ellipsoids

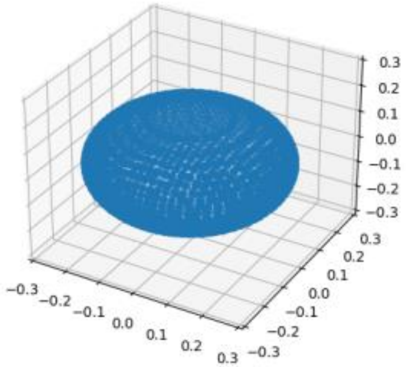
manifold of 3D objects is the space of **ellipsoids**. An ellipsoid is defin

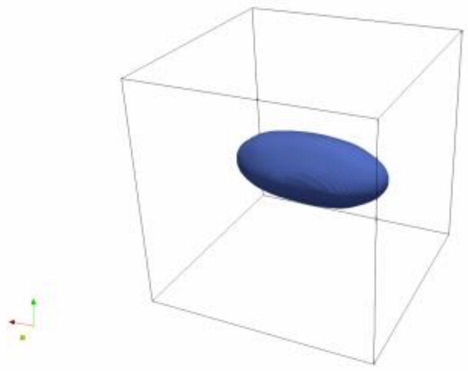
$$\mathcal{E}\ell\ell := \left\{ (x,y,z)^T \in \Omega \mid \left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1 \right\}$$

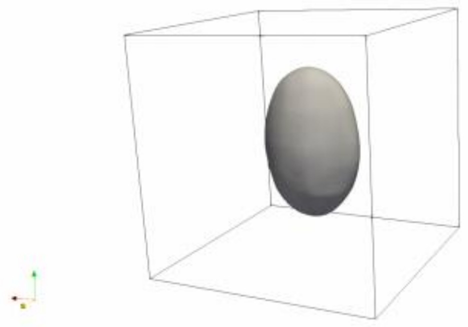
ation one could also directly compute the signed distance to the object

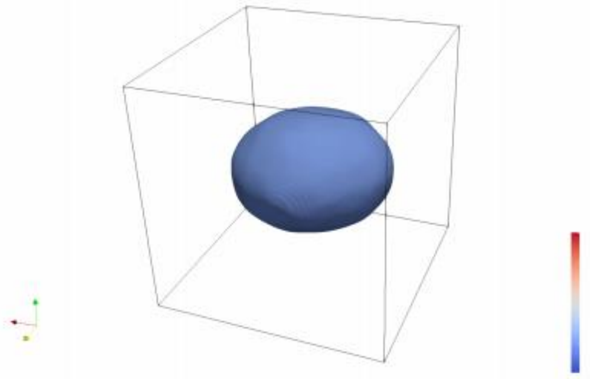

(a) GT $(0.29, 0.12, 0.12)$


(b) GT $(0.17, 0.12, 0.26)$

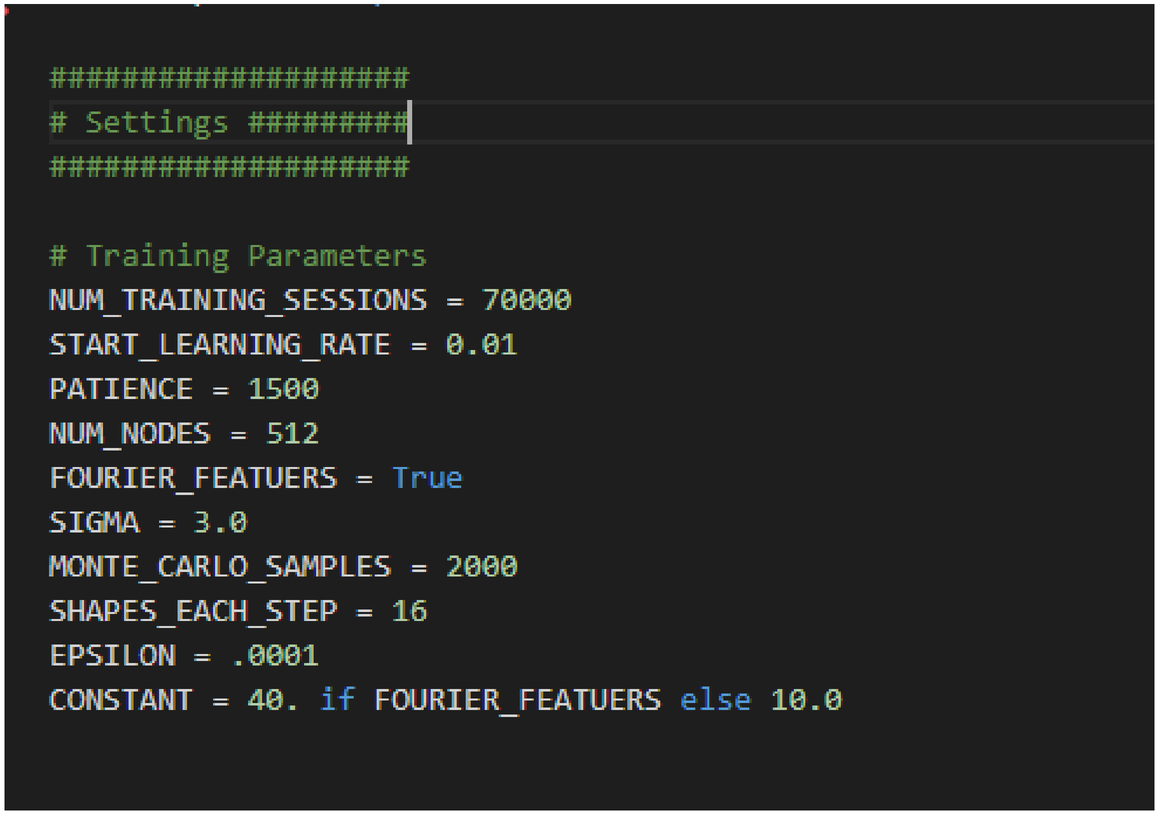

(c) GT $(0.28, 0.28, 0.19)$


(d) Result $(0.29, 0.12, 0.12)$

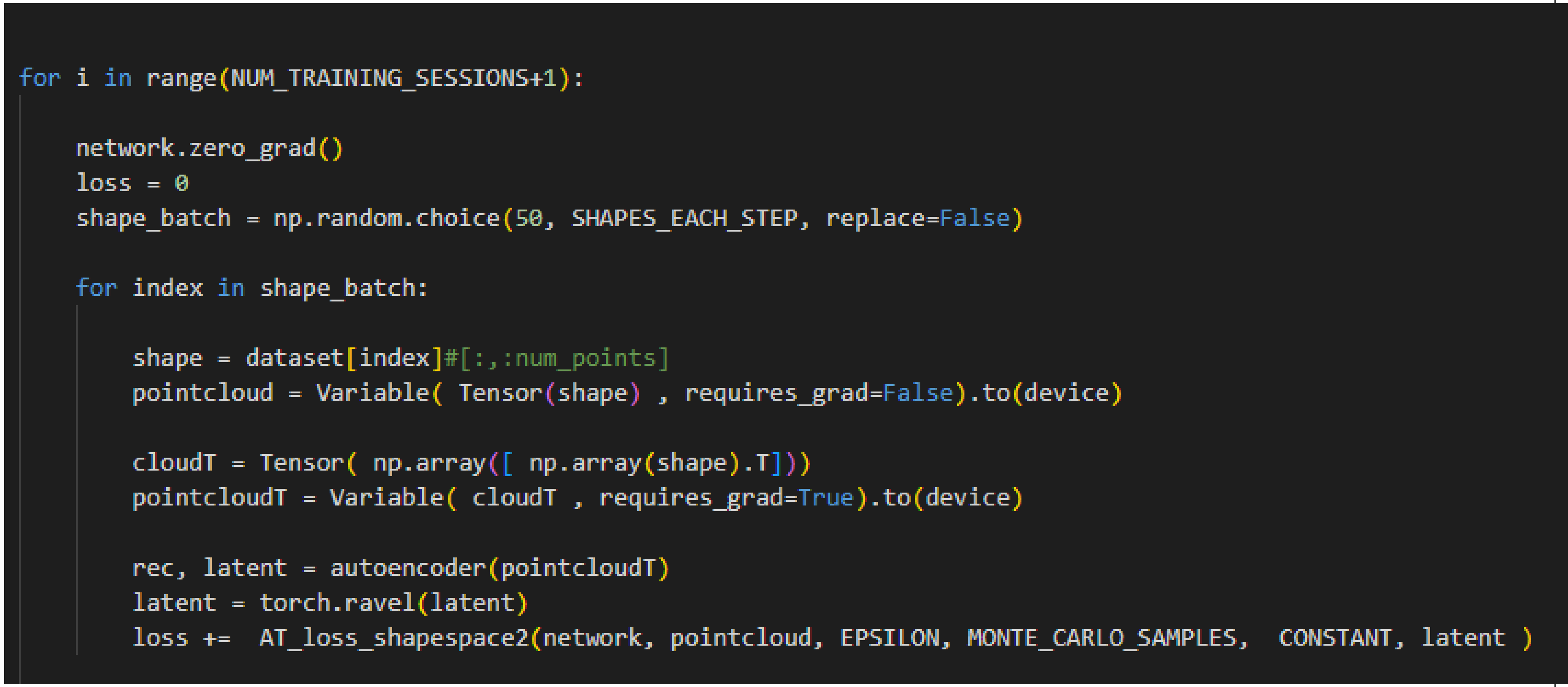

(e) Result $(0.17, 0.12, 0.26)$


(f) Result $(0.28, 0.28, 0.19)$

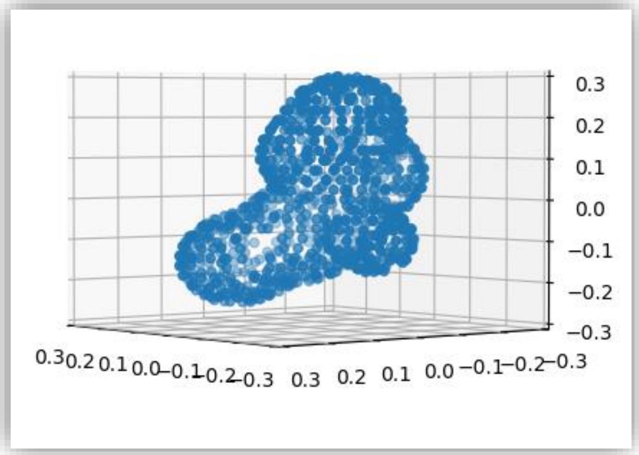
Feature saved manually



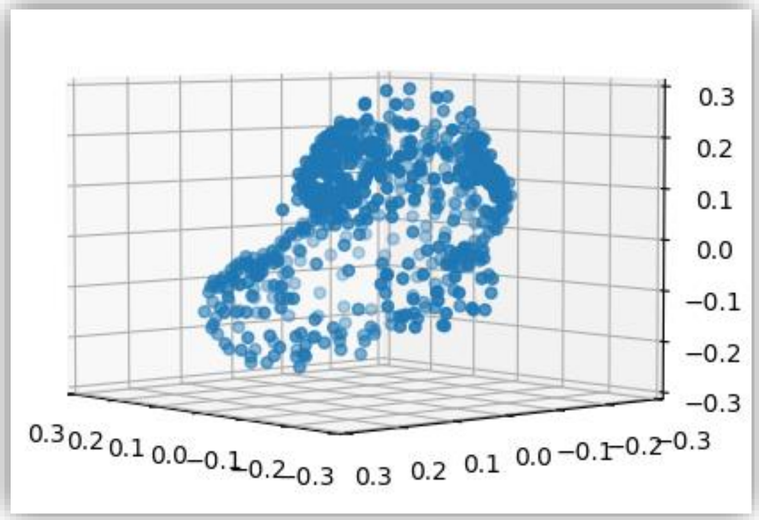
```
####################
# Settings #########
####################

# Training Parameters
NUM_TRAINING_SESSIONS = 70000
START_LEARNING_RATE = 0.01
PATIENCE = 1500
NUM_NODES = 512
FOURIER_FEATUERS = True
SIGMA = 3.0
MONTE_CARLO_SAMPLES = 2000
SHAPES_EACH_STEP = 16
EPSILON = .0001
CONSTANT = 40. if FOURIER_FEATUERS else 10.0
```



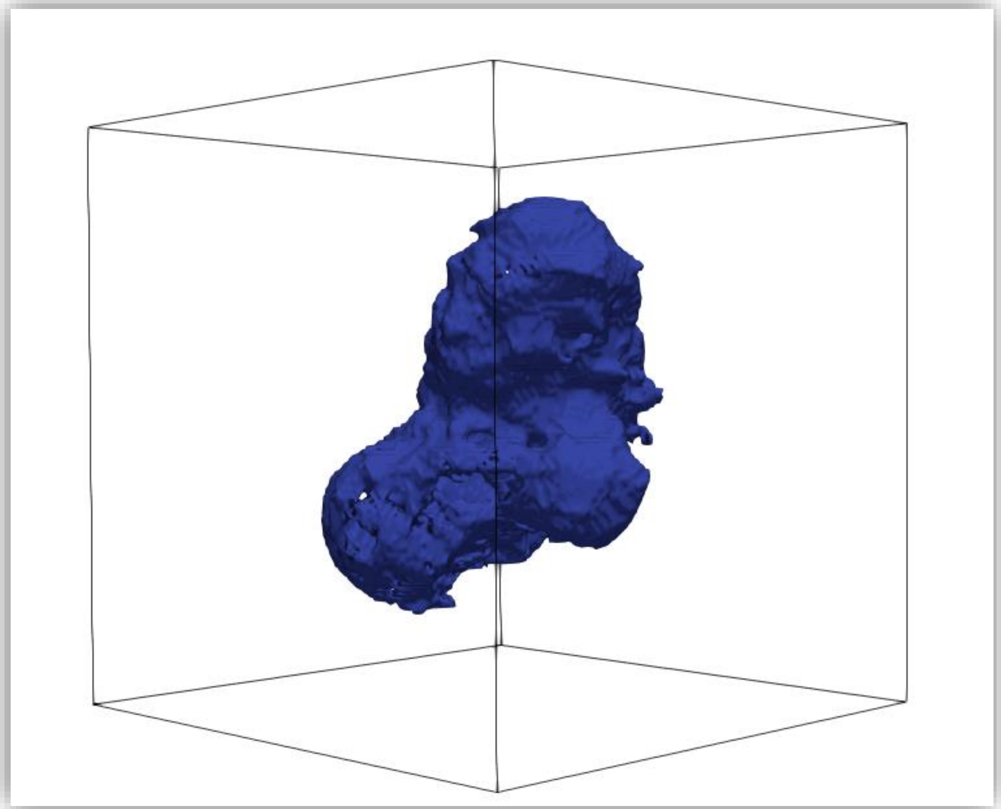
```
for i in range(NUM_TRAINING_SESSIONS+1):

    network.zero_grad()
    loss = 0
    shape_batch = np.random.choice(50, SHAPES_EACH_STEP, replace=False)

    for index in shape_batch:

        shape = dataset[index]#[:,:num_points]
        pointcloud = Variable( Tensor(shape) , requires_grad=False).to(device)

        cloudT = Tensor( np.array([ np.array(shape).T]))
        pointcloudT = Variable( cloudT , requires_grad=True).to(device)

        rec, latent = autoencoder(pointcloudT)
        latent = torch.ravel(latent)
        loss +=  AT_loss_shapespace2(network, pointcloud, EPSILON, MONTE_CARLO_SAMPLES,  CONSTANT, latent )
```

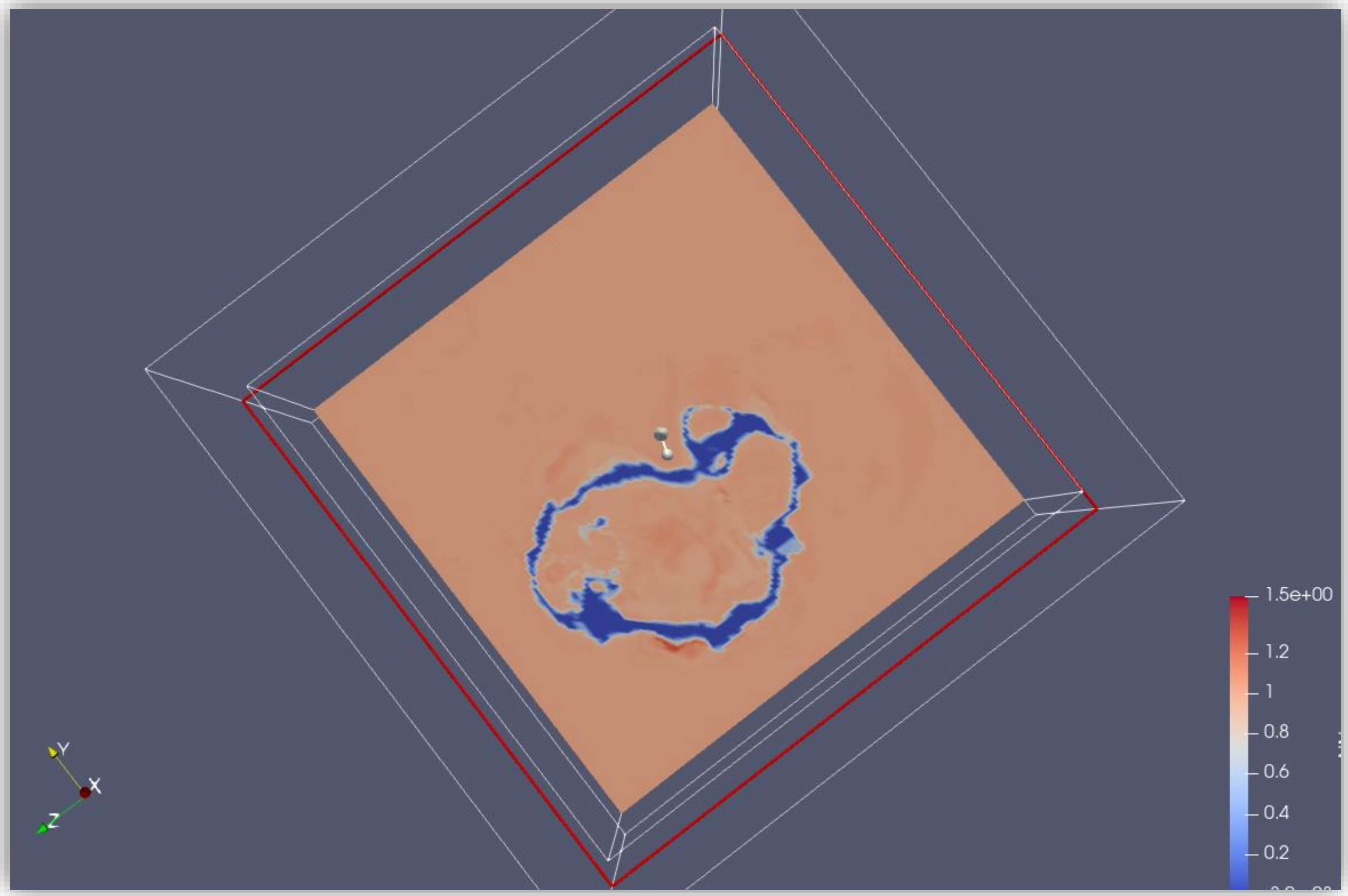


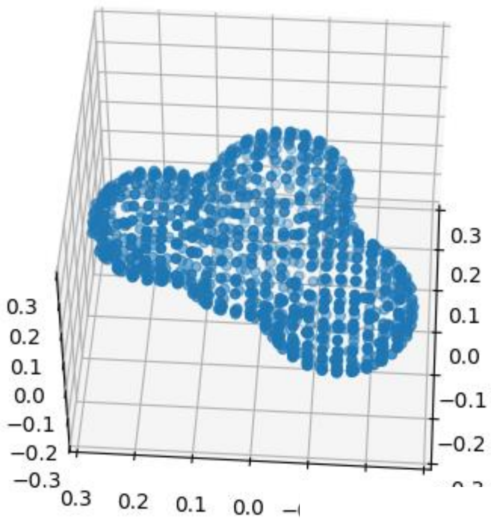
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3

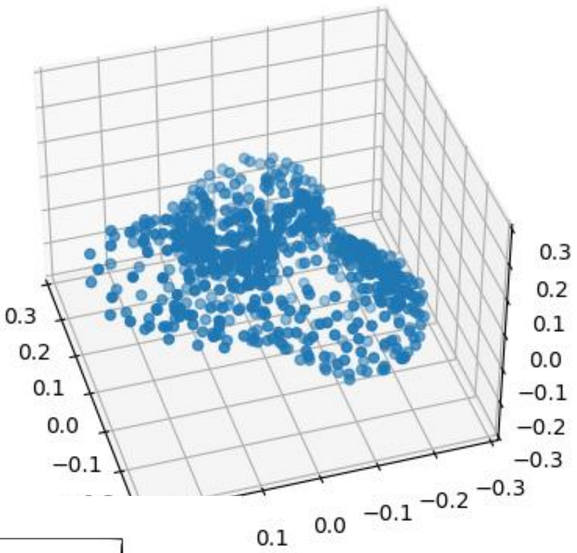


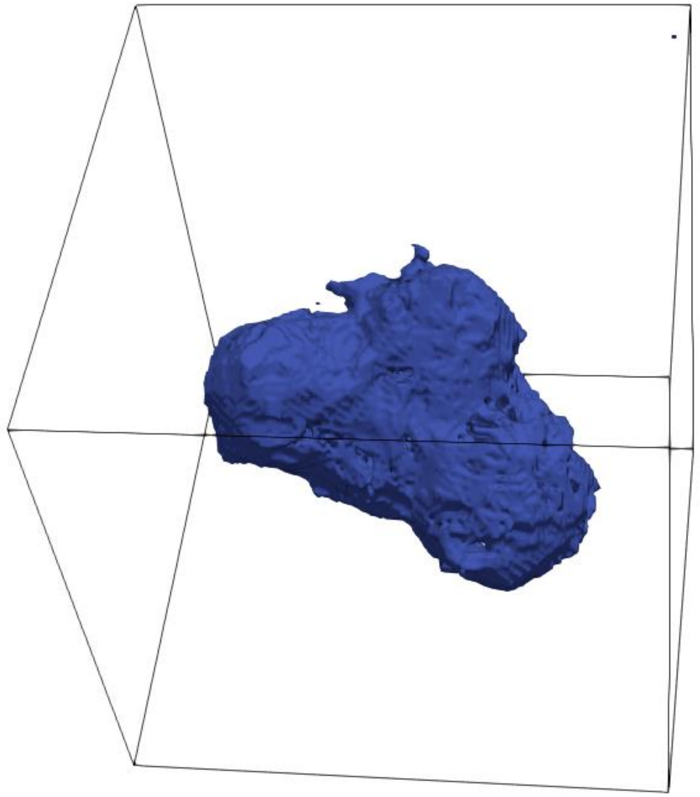
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3
0.3 0.2 0.1 0.0 −0.1 −0.2 −0.3

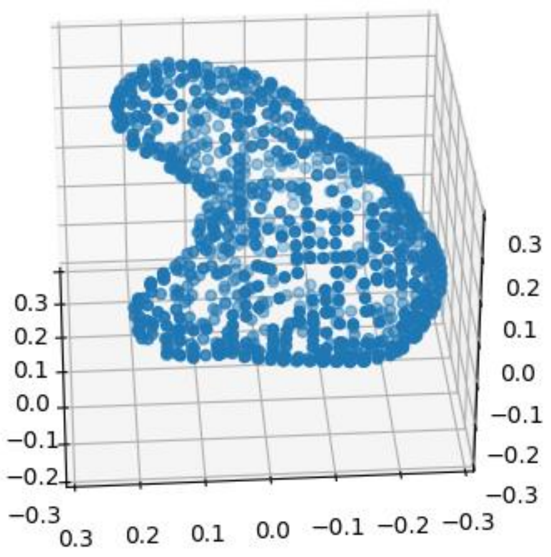


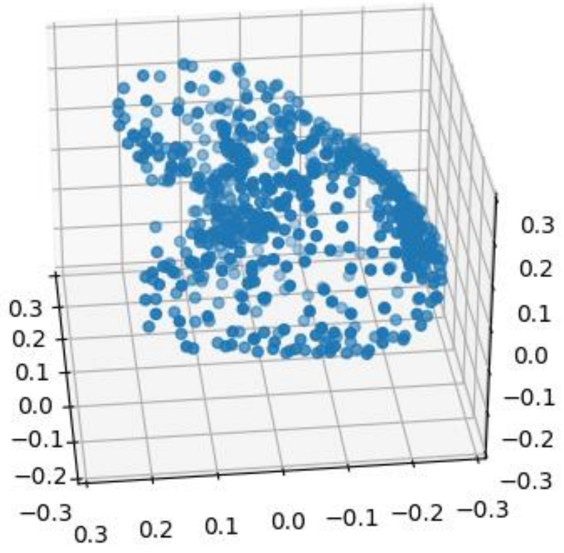

1.5e+00
1.2
1
0.8
0.6
0.4
0.2
Y
X
Z

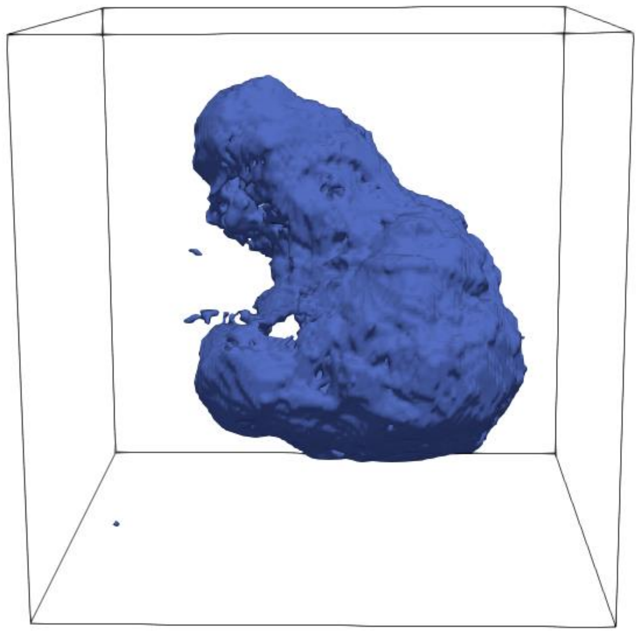

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0

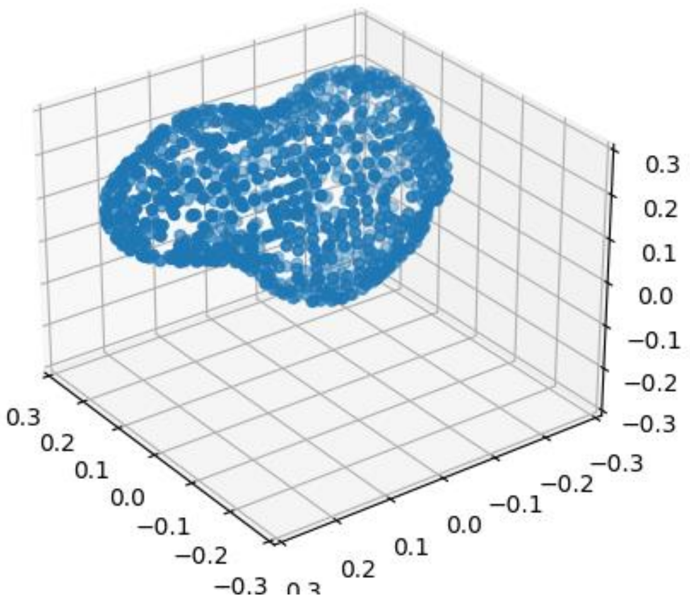

0.3
0.2
0.1
0.0
−0.1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.1
0.0
−0.1
−0.2
−0.3

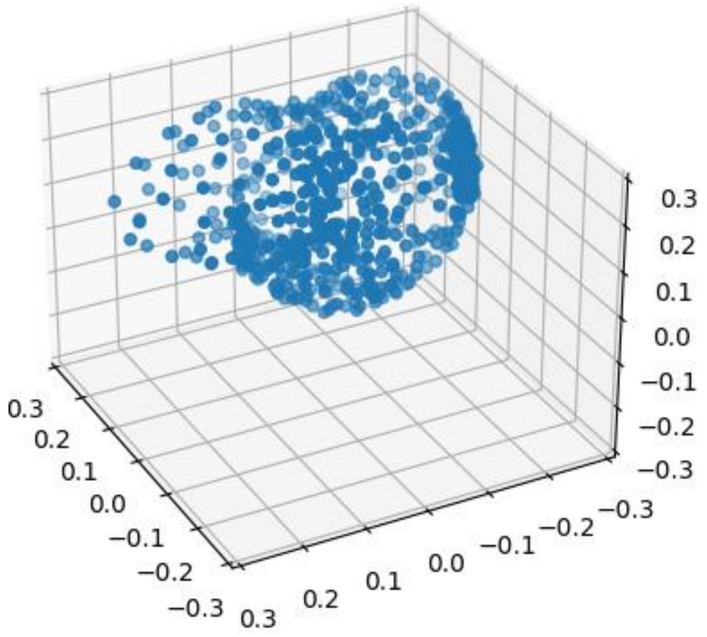


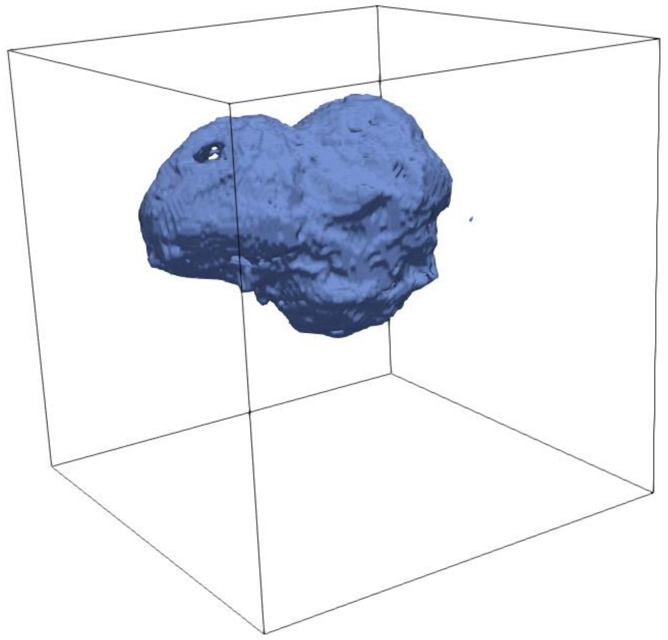

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

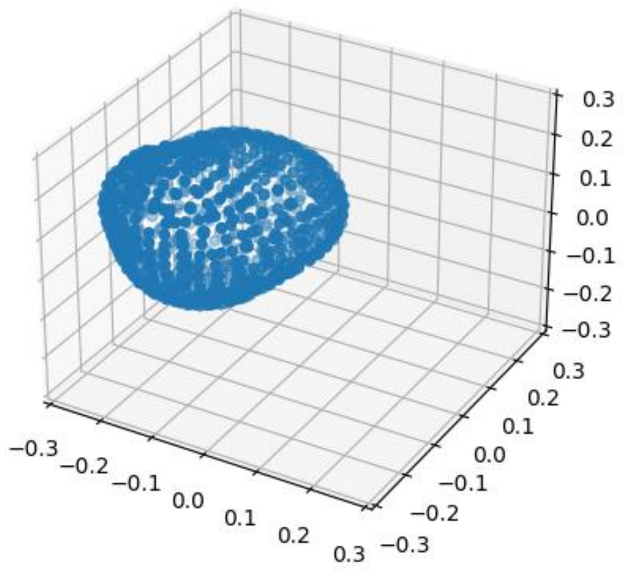

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

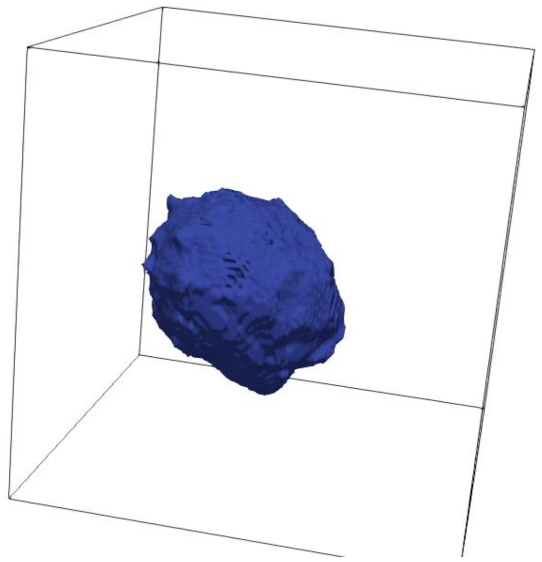


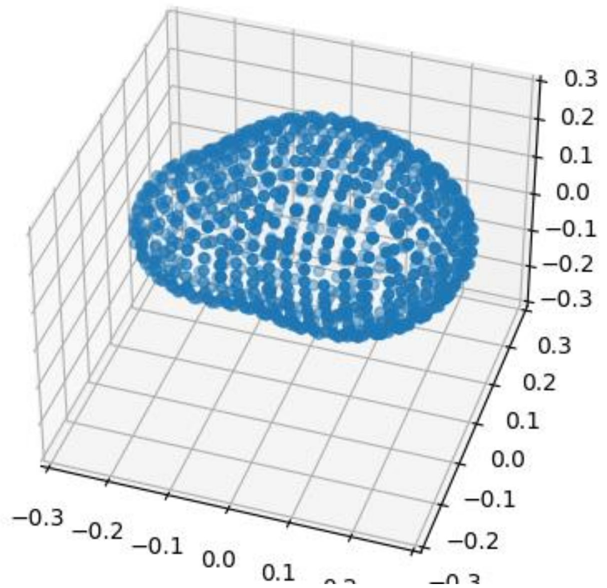

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

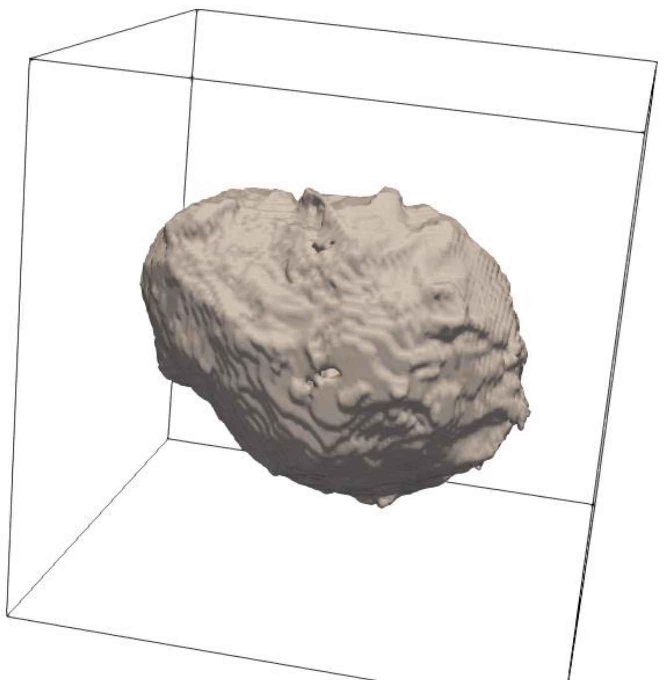

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

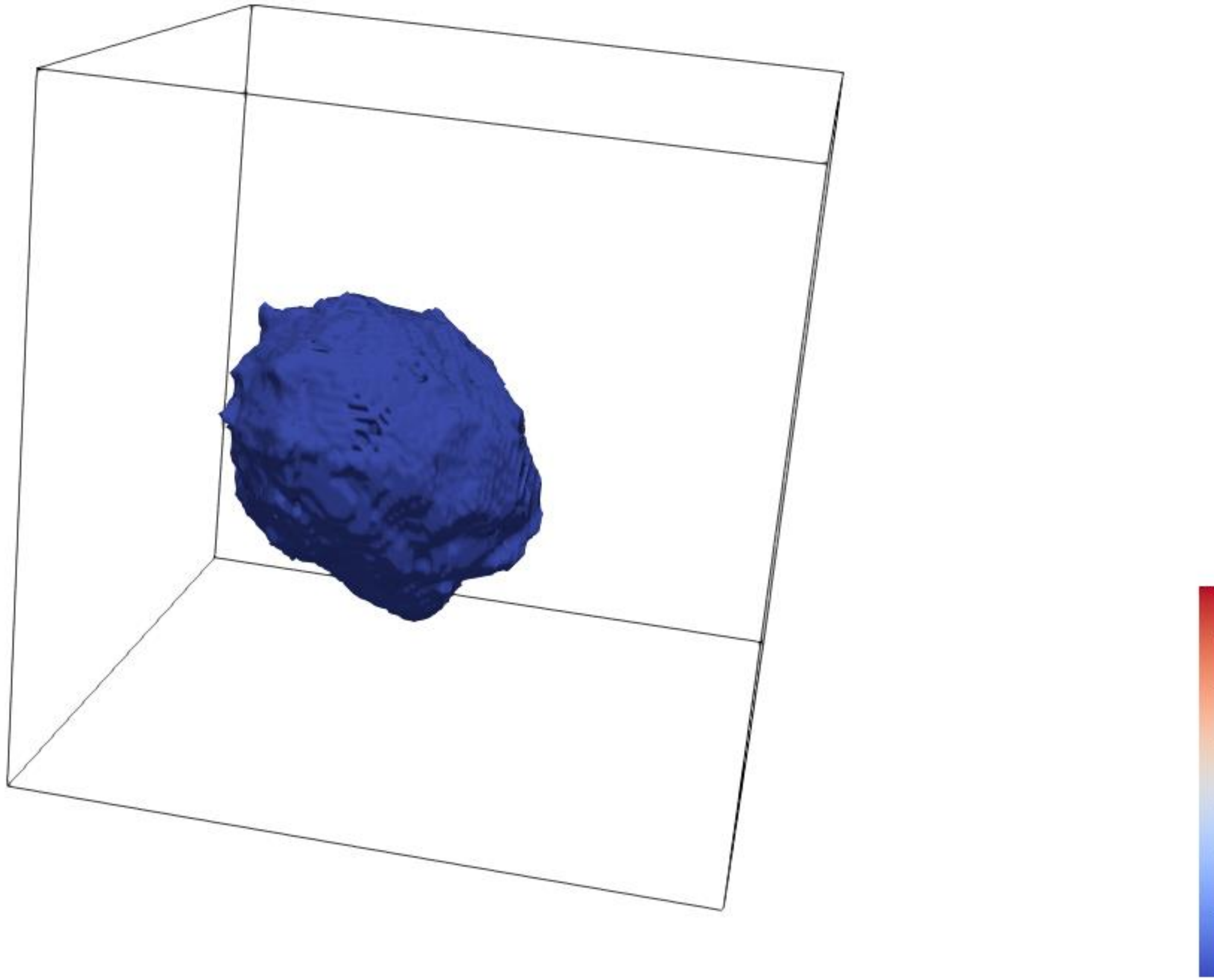


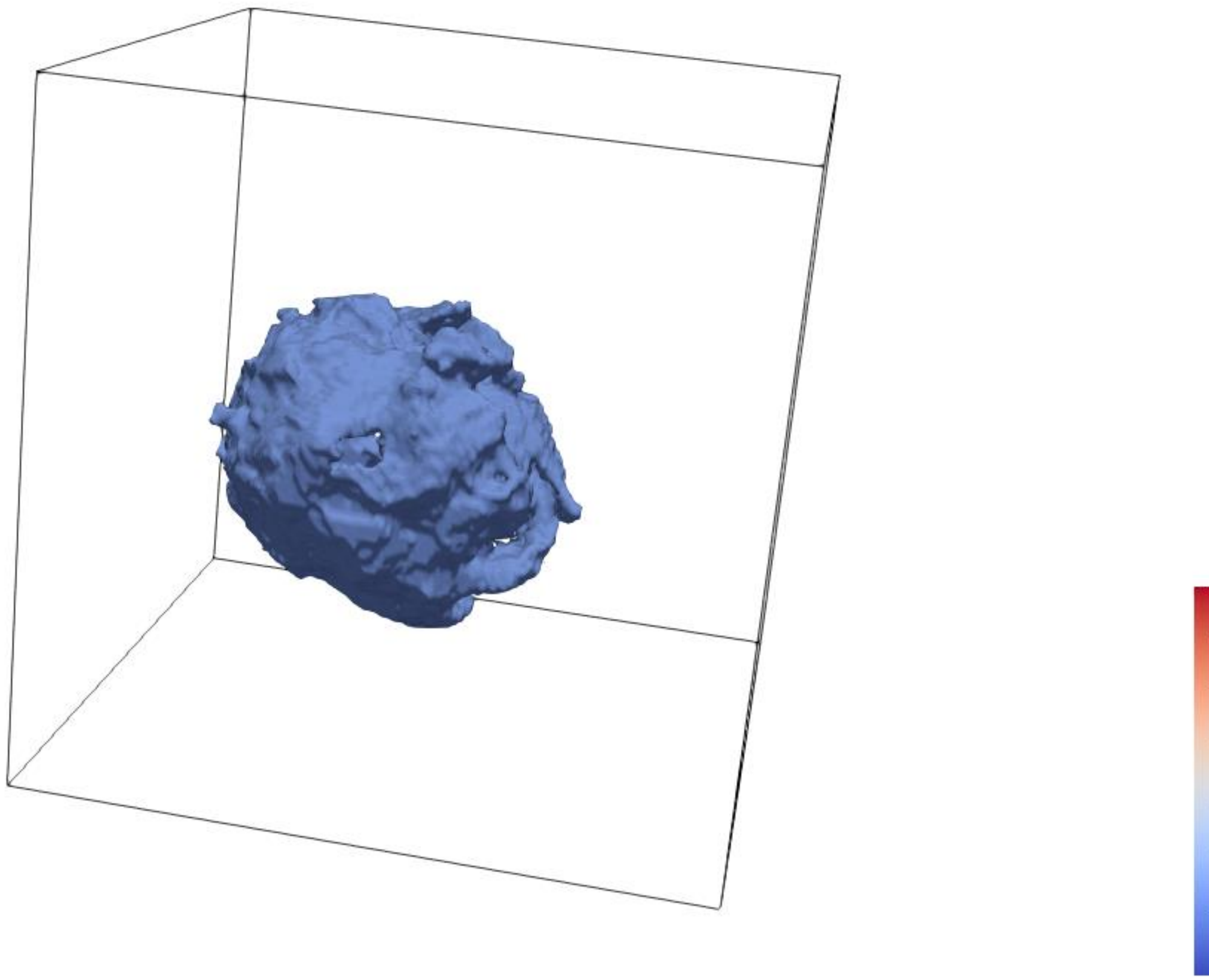

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

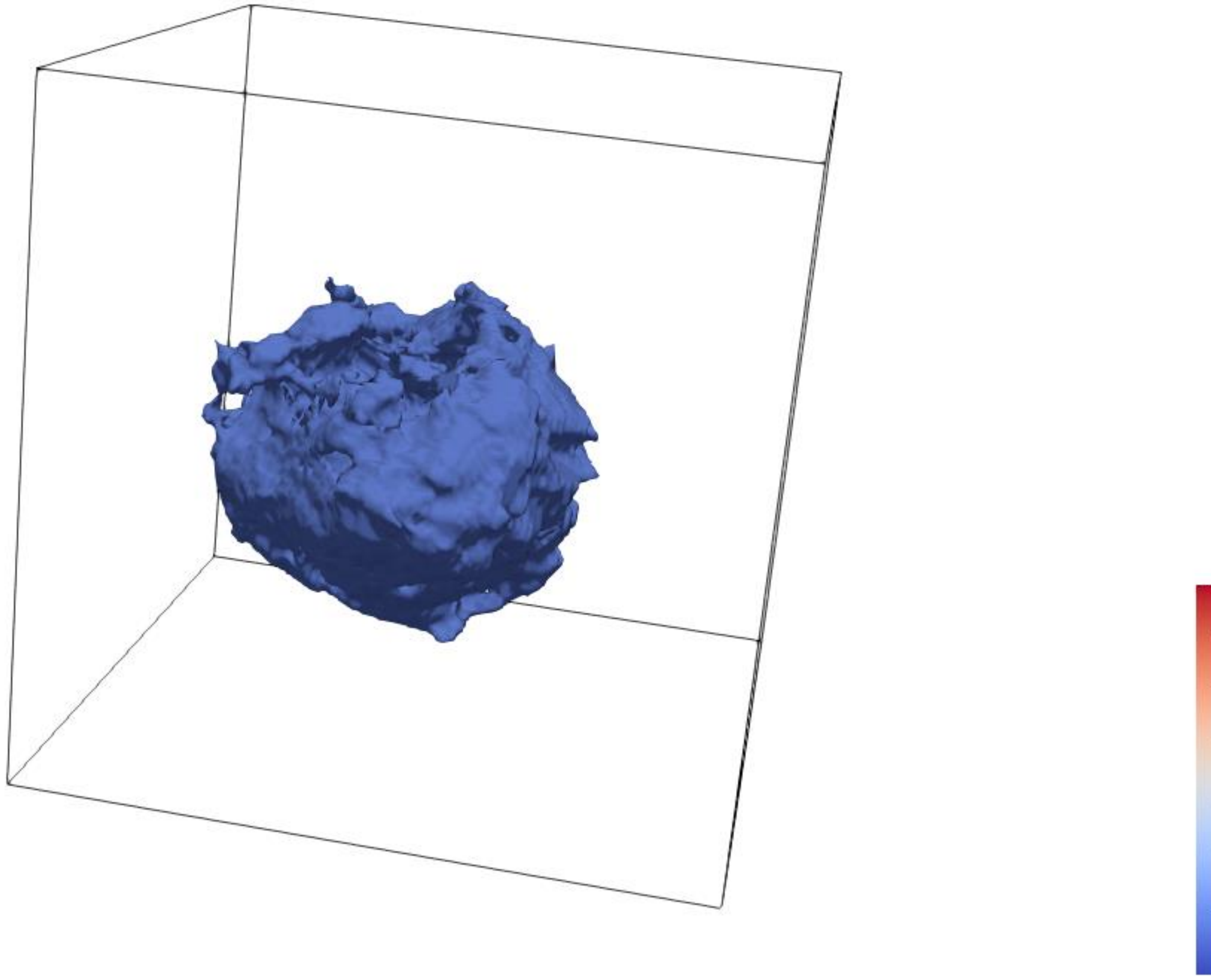


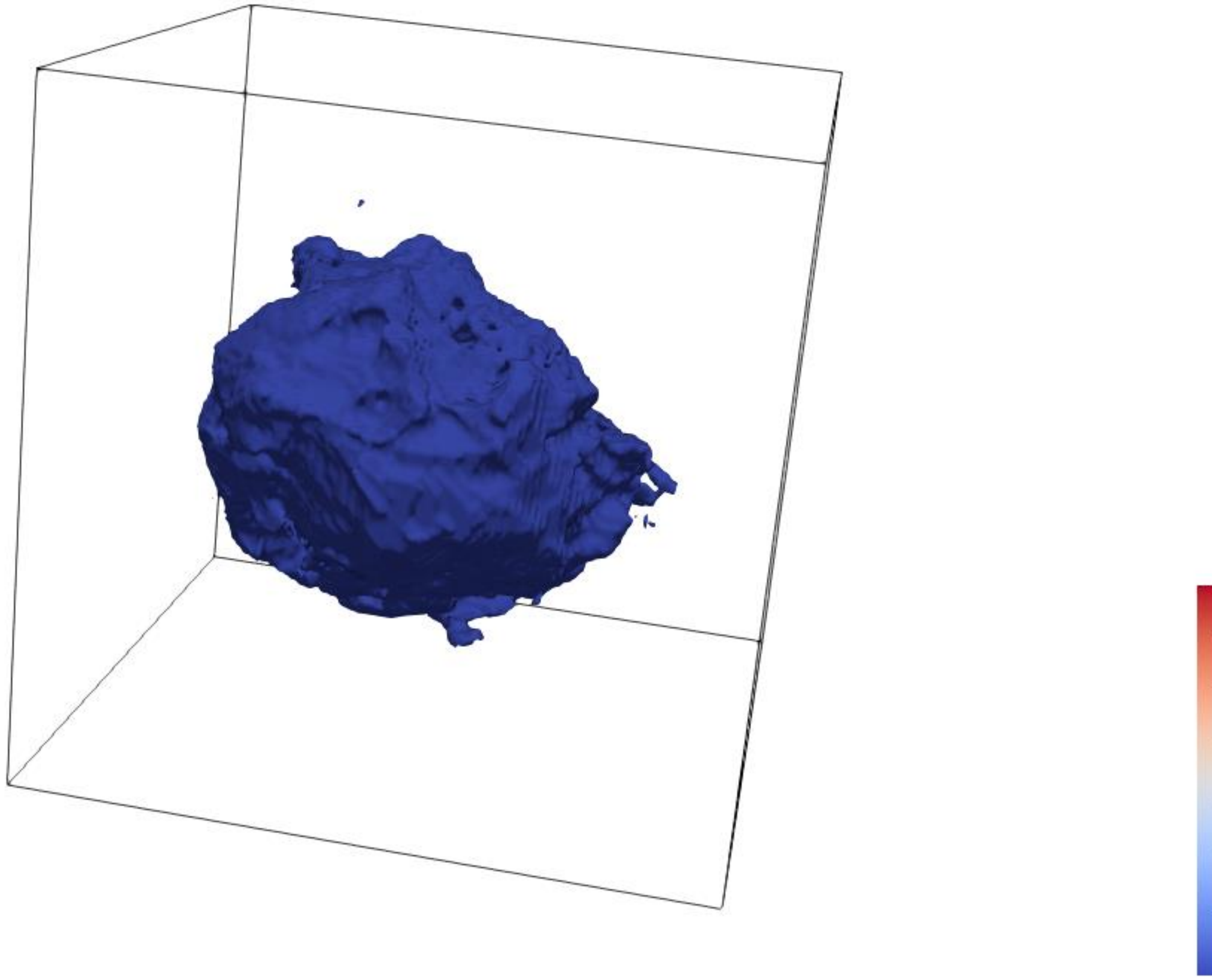

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.3
0.2
0.1
0.0
−0.1
−0.2

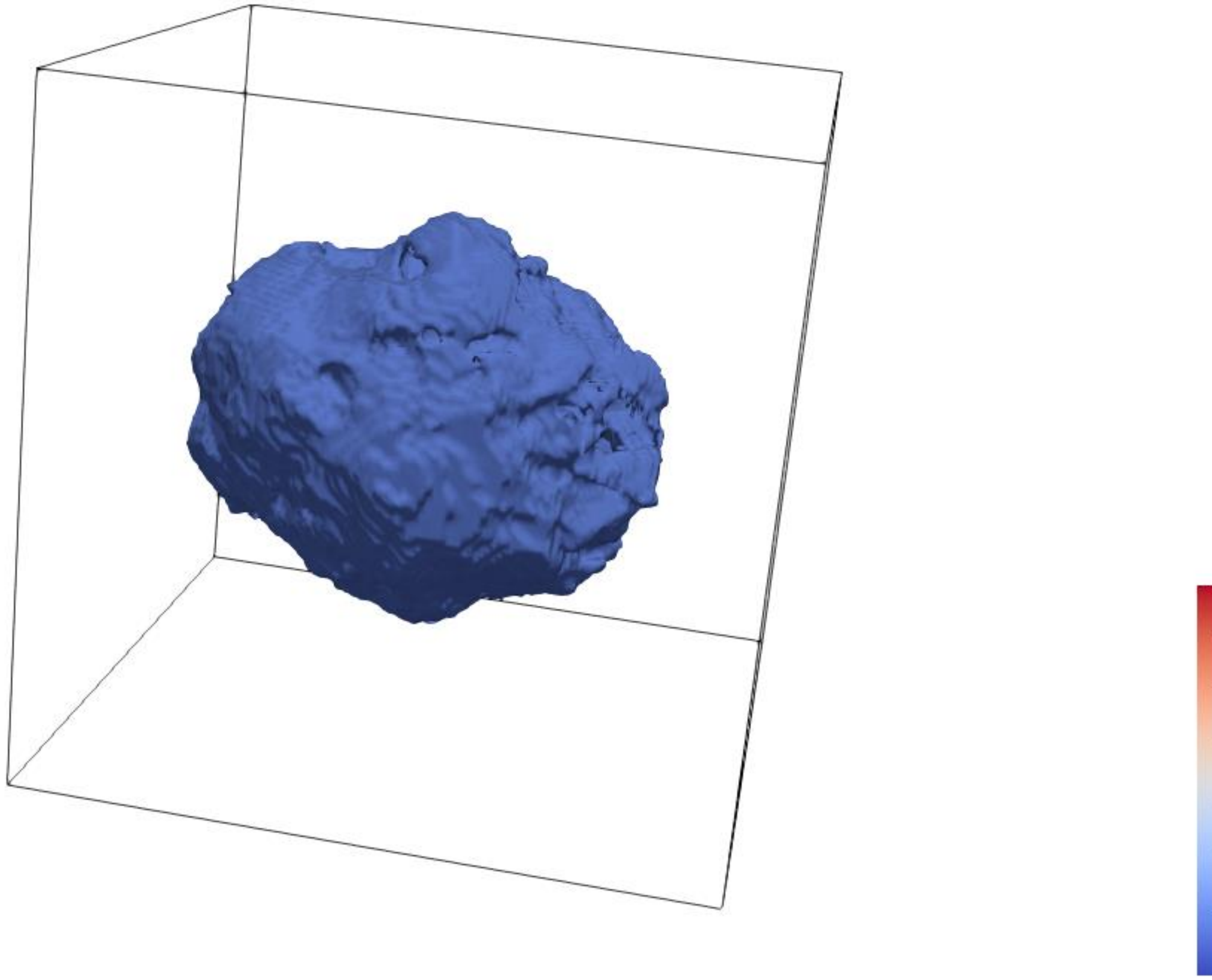


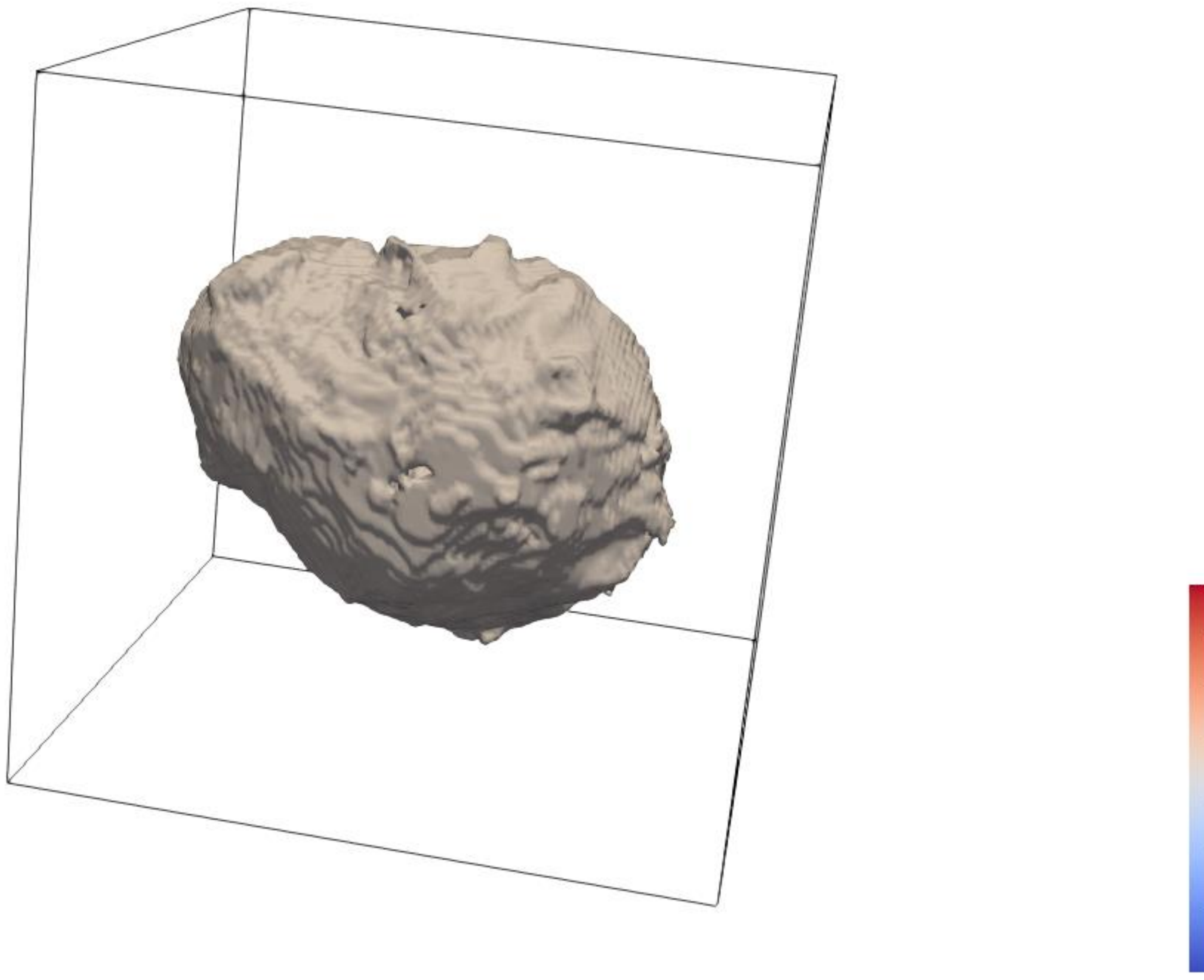


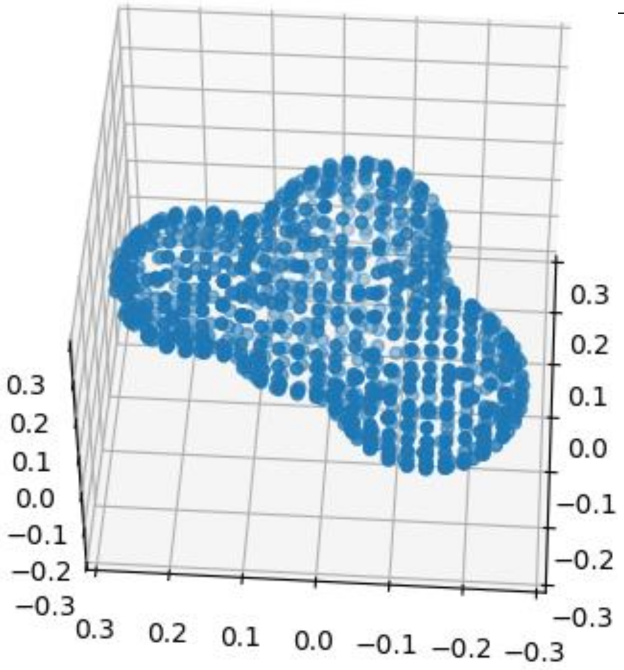


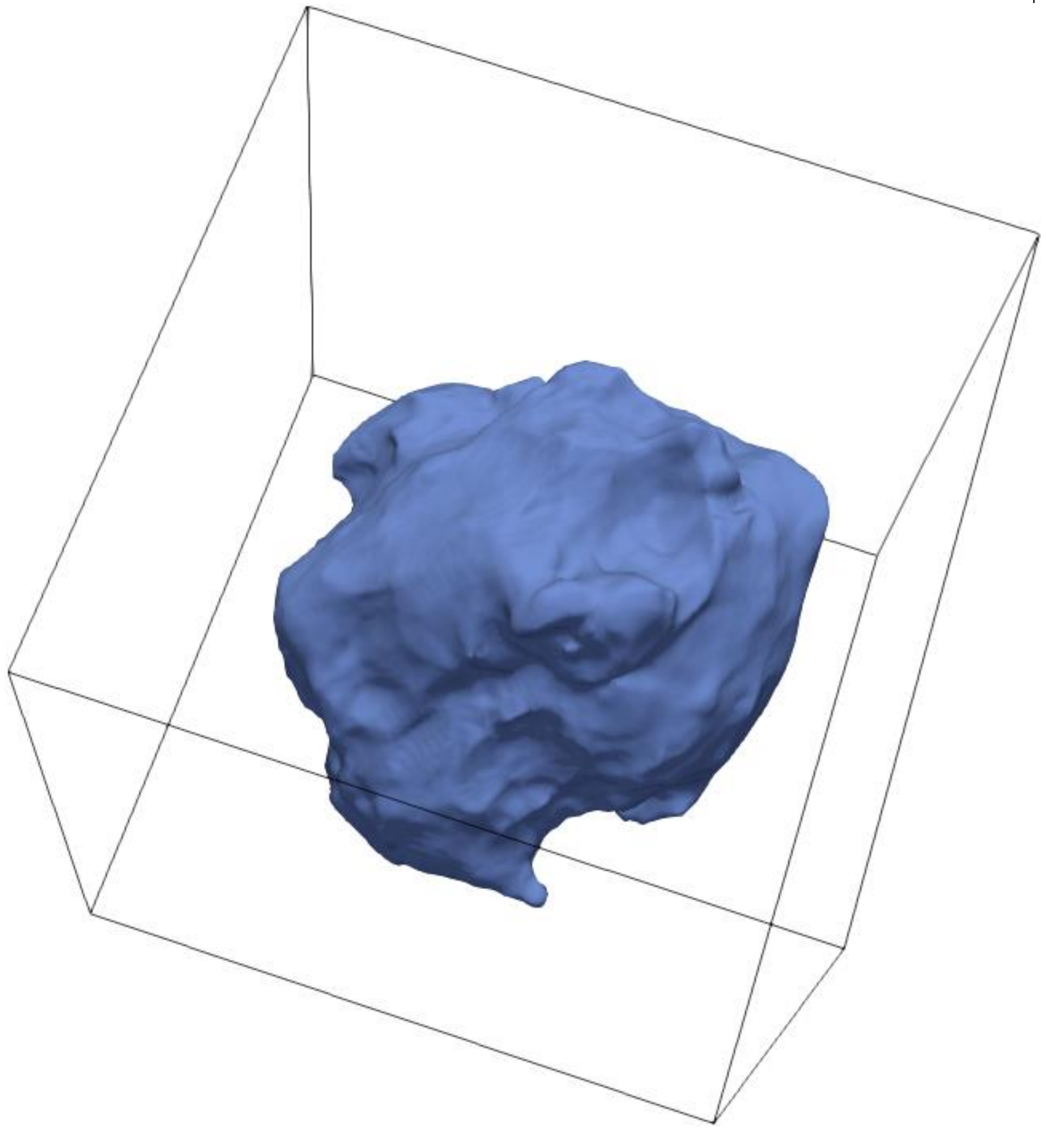




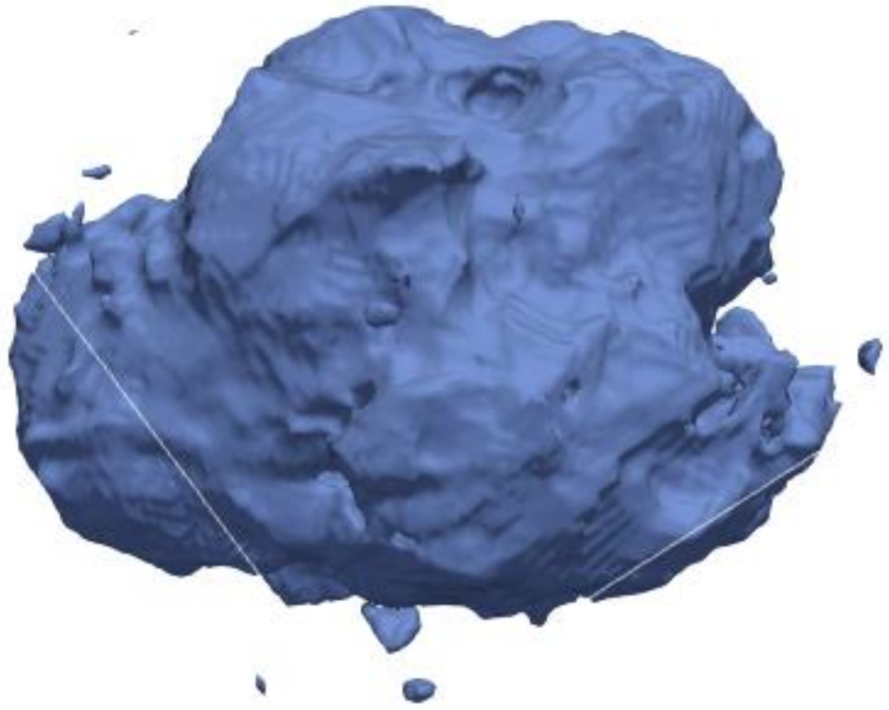
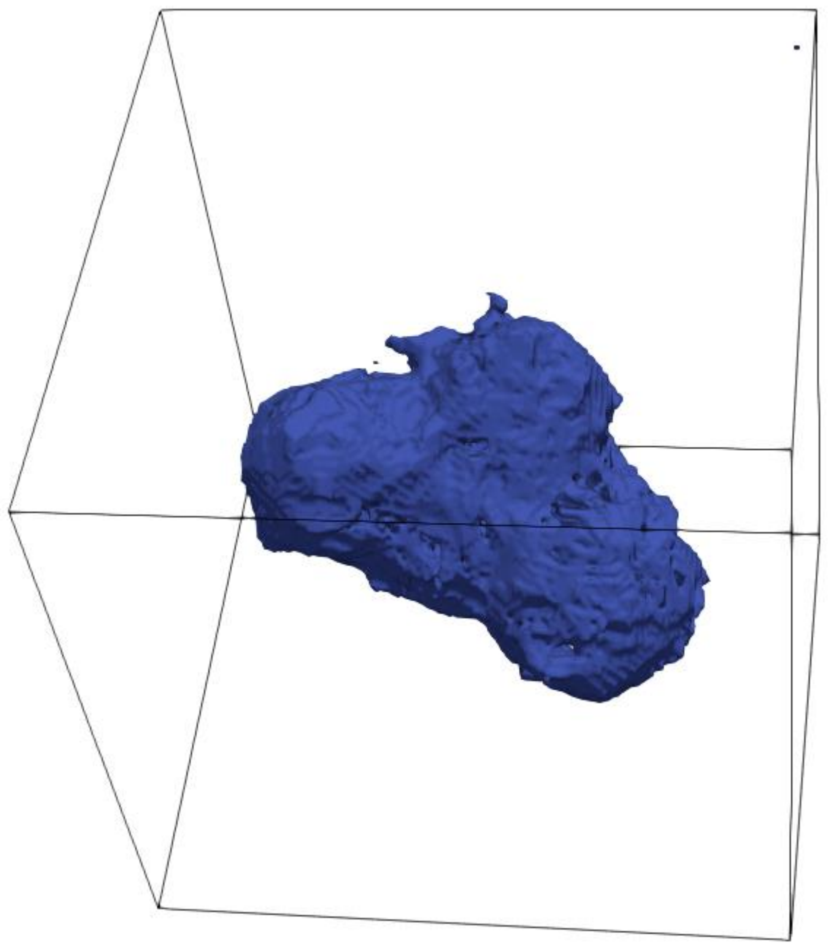


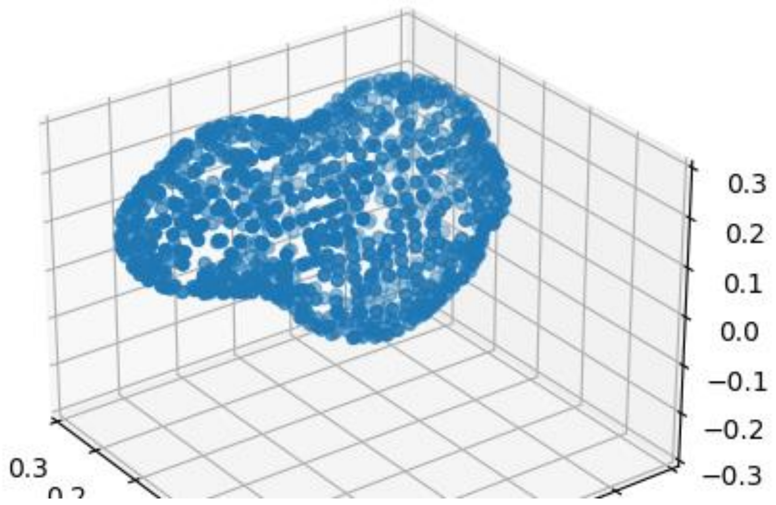


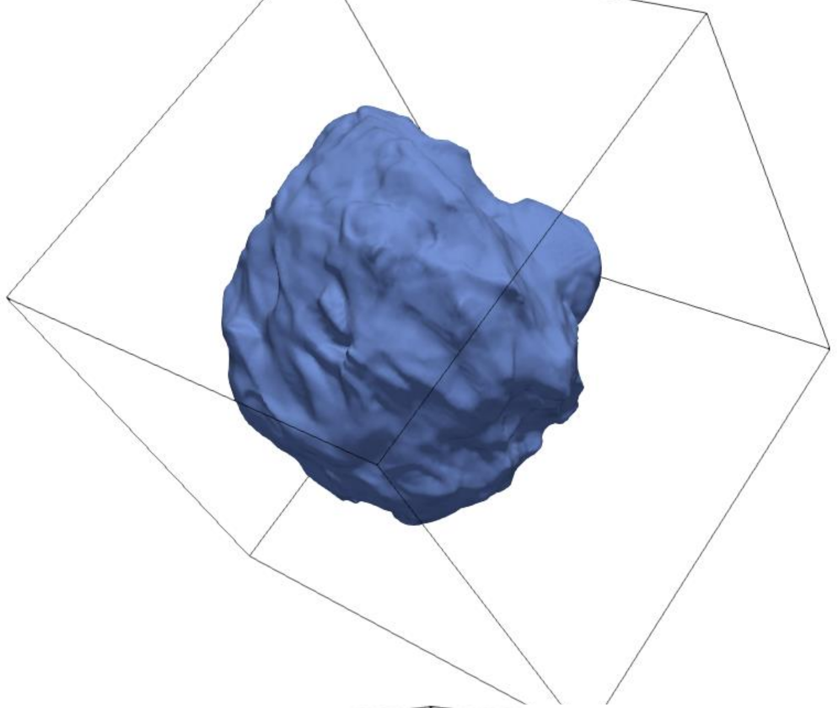


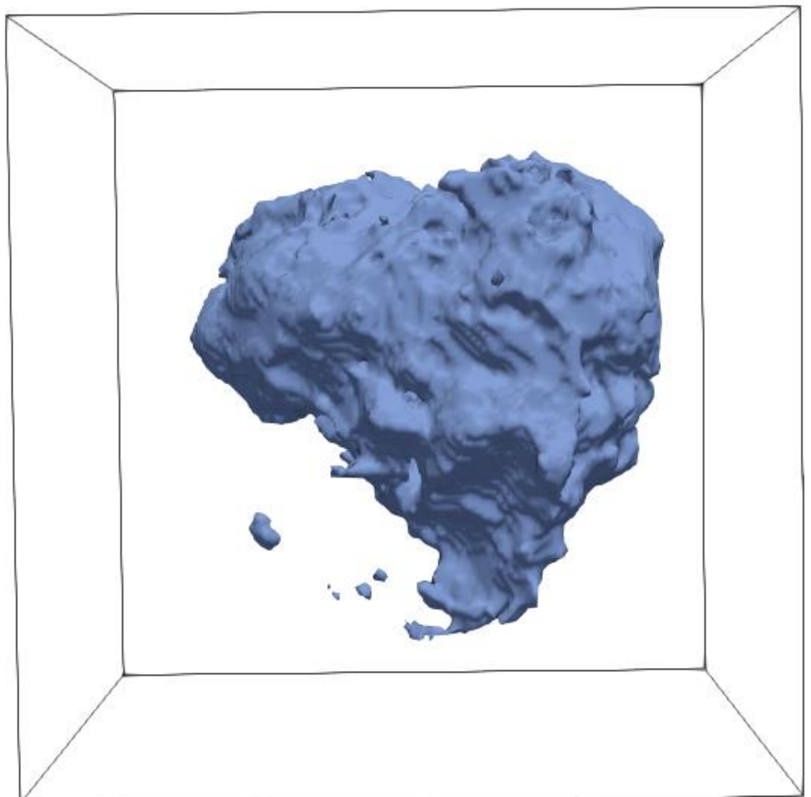


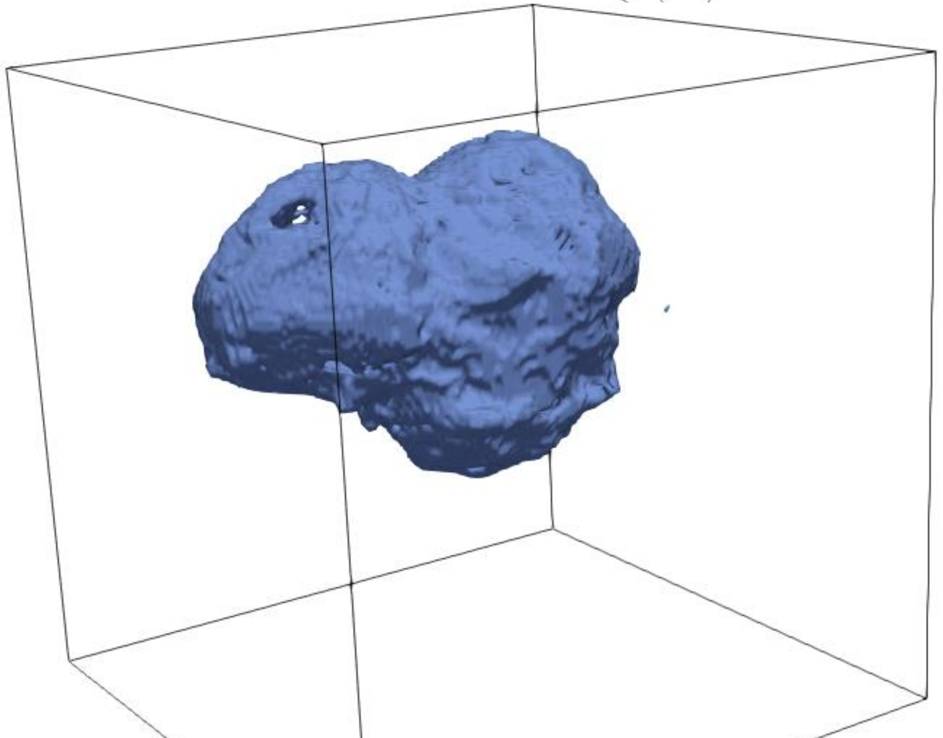
Eps= 10^-2

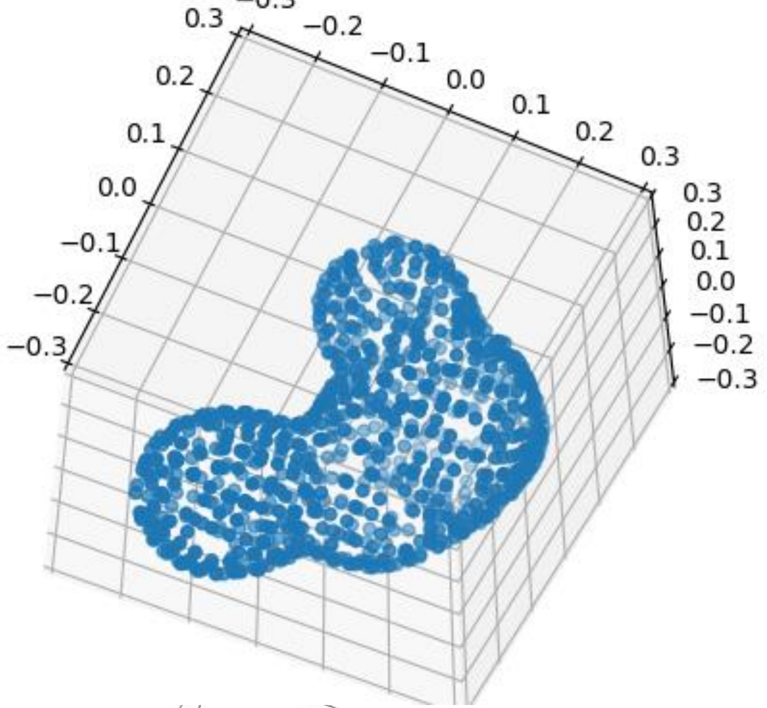
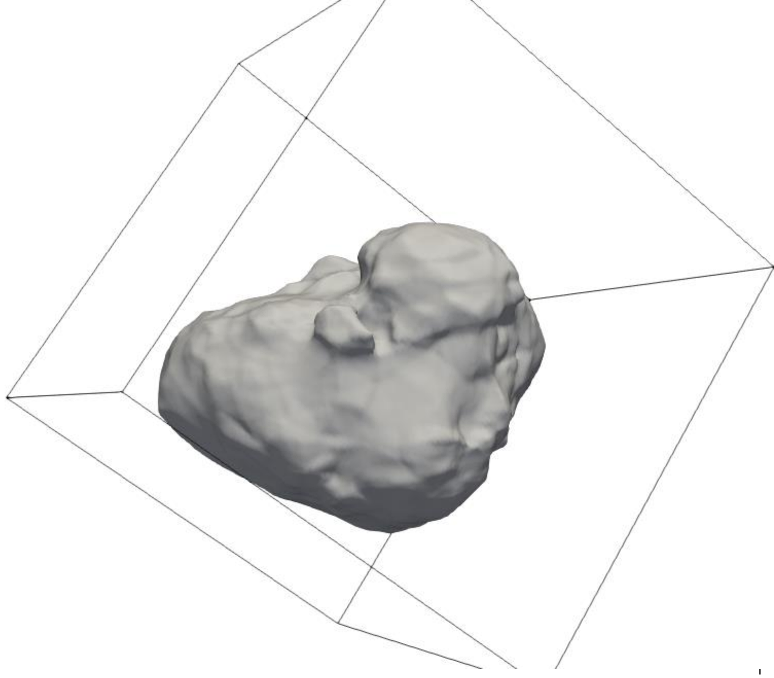
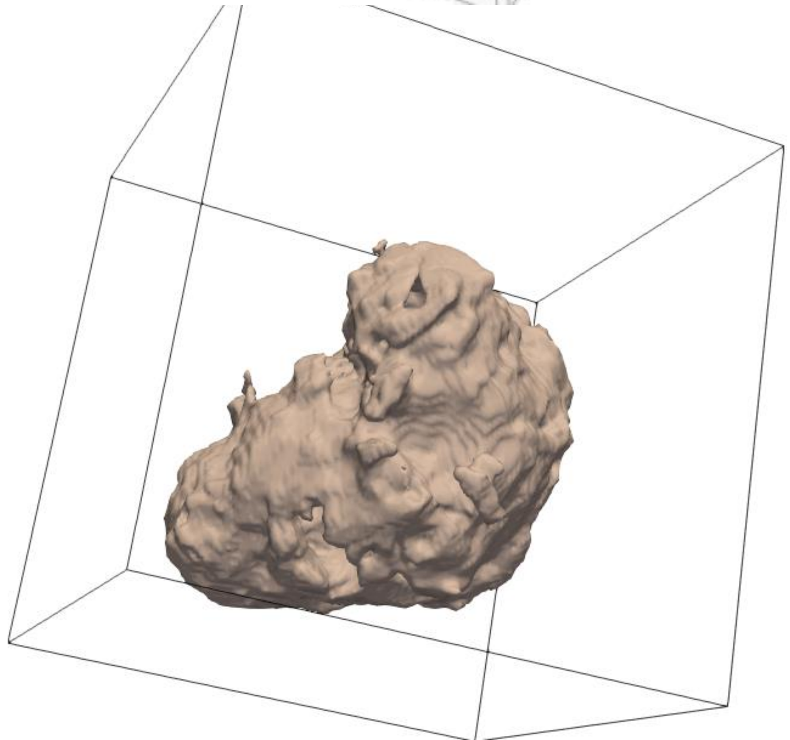
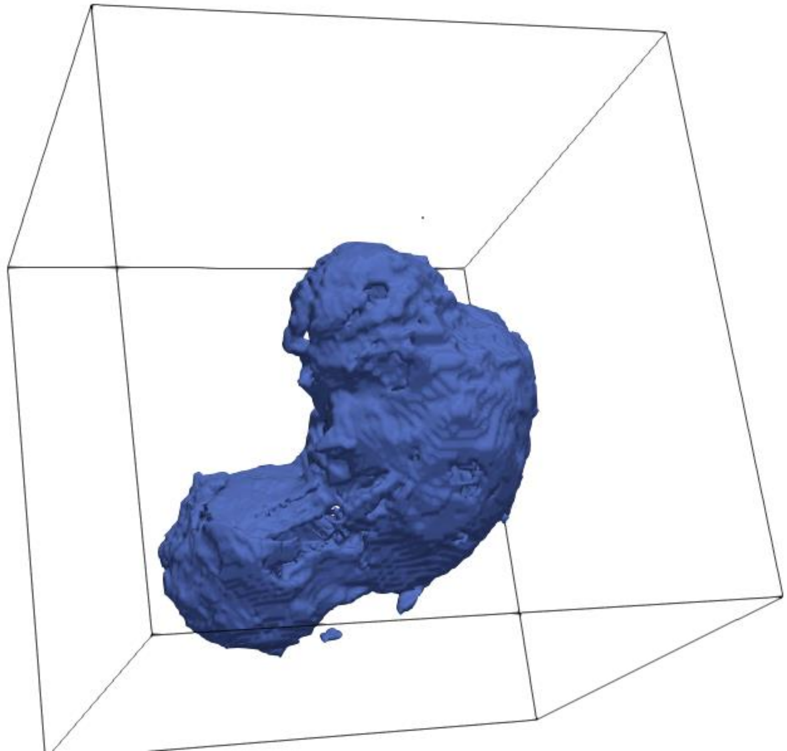



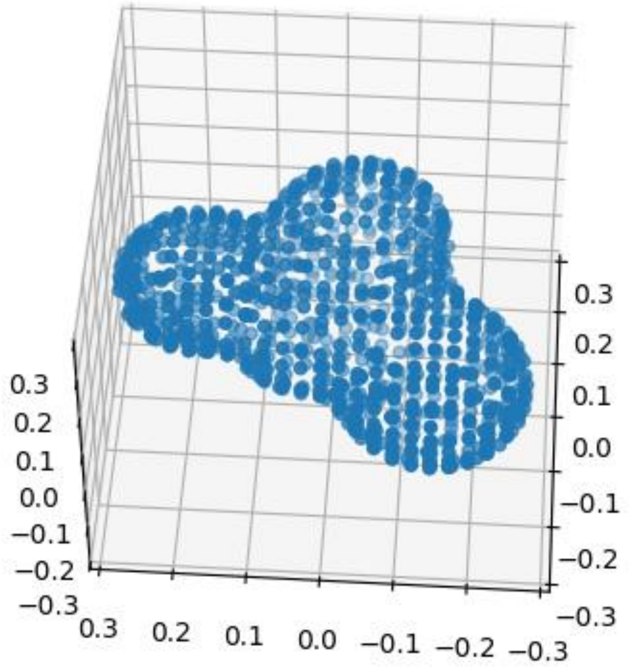

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2

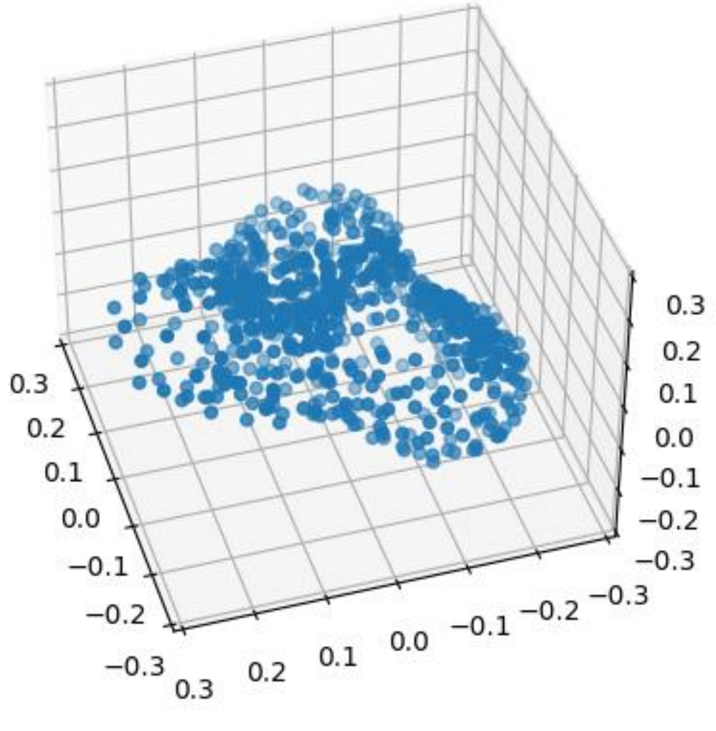




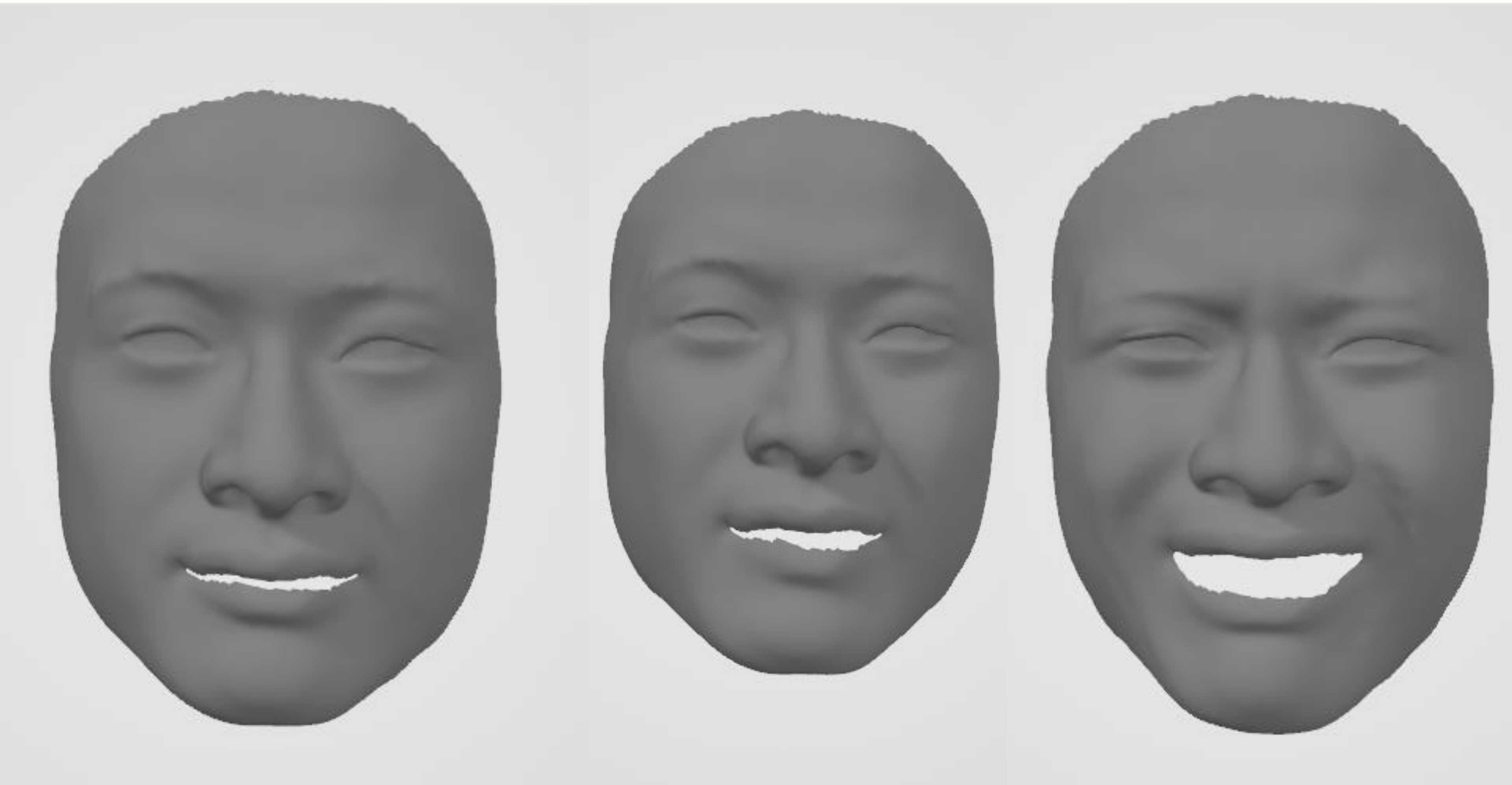


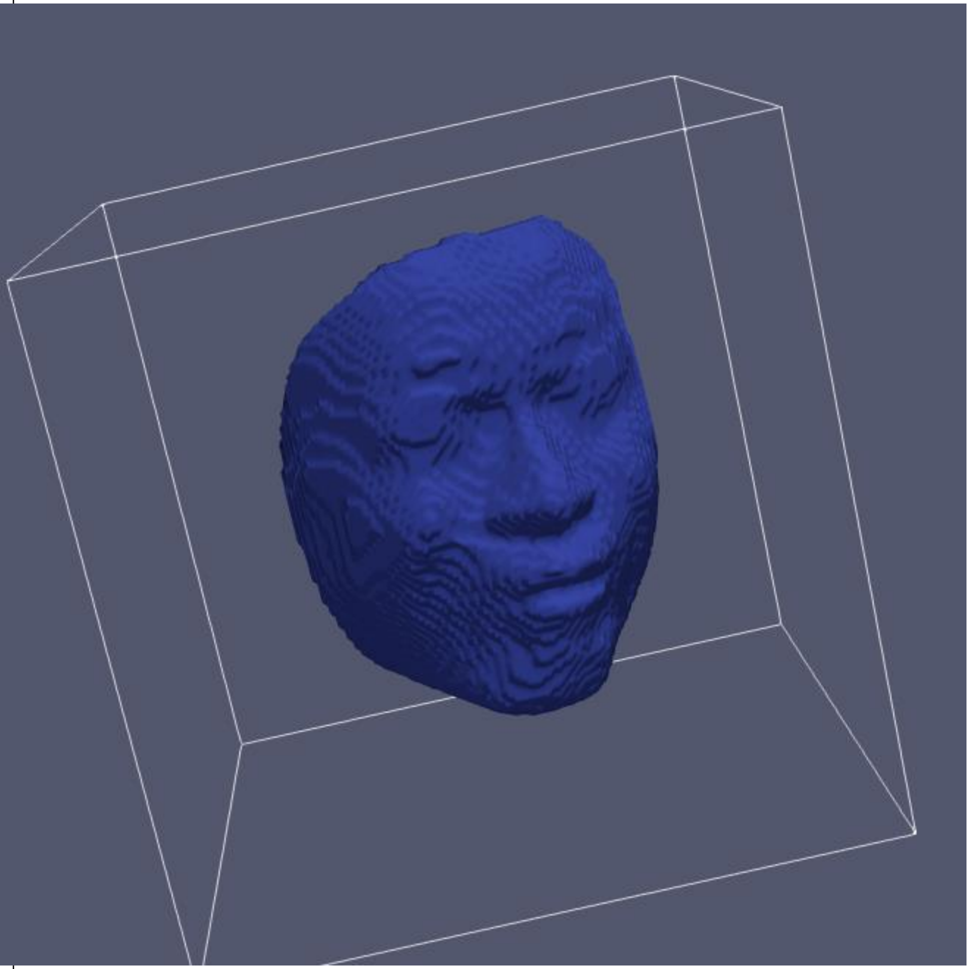
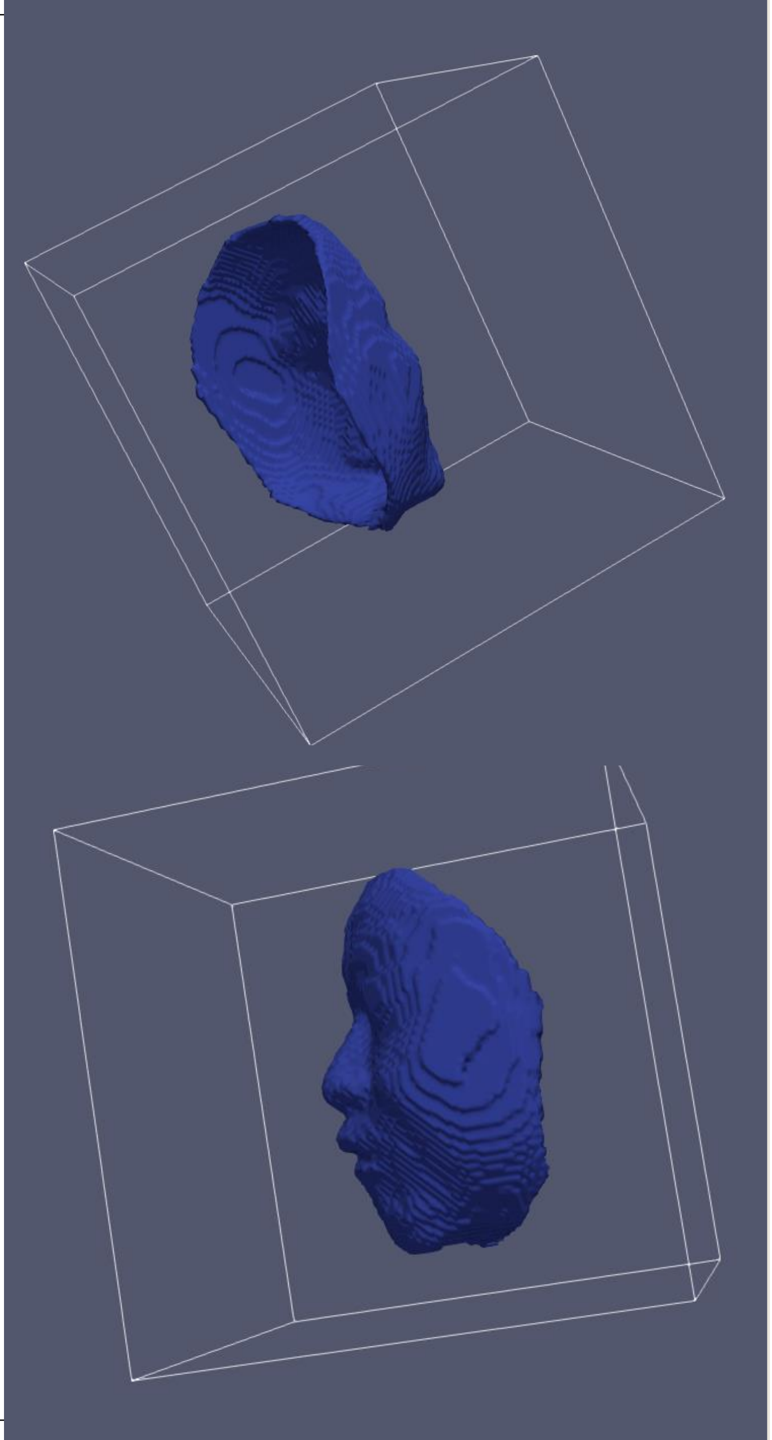

−0.3
0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3




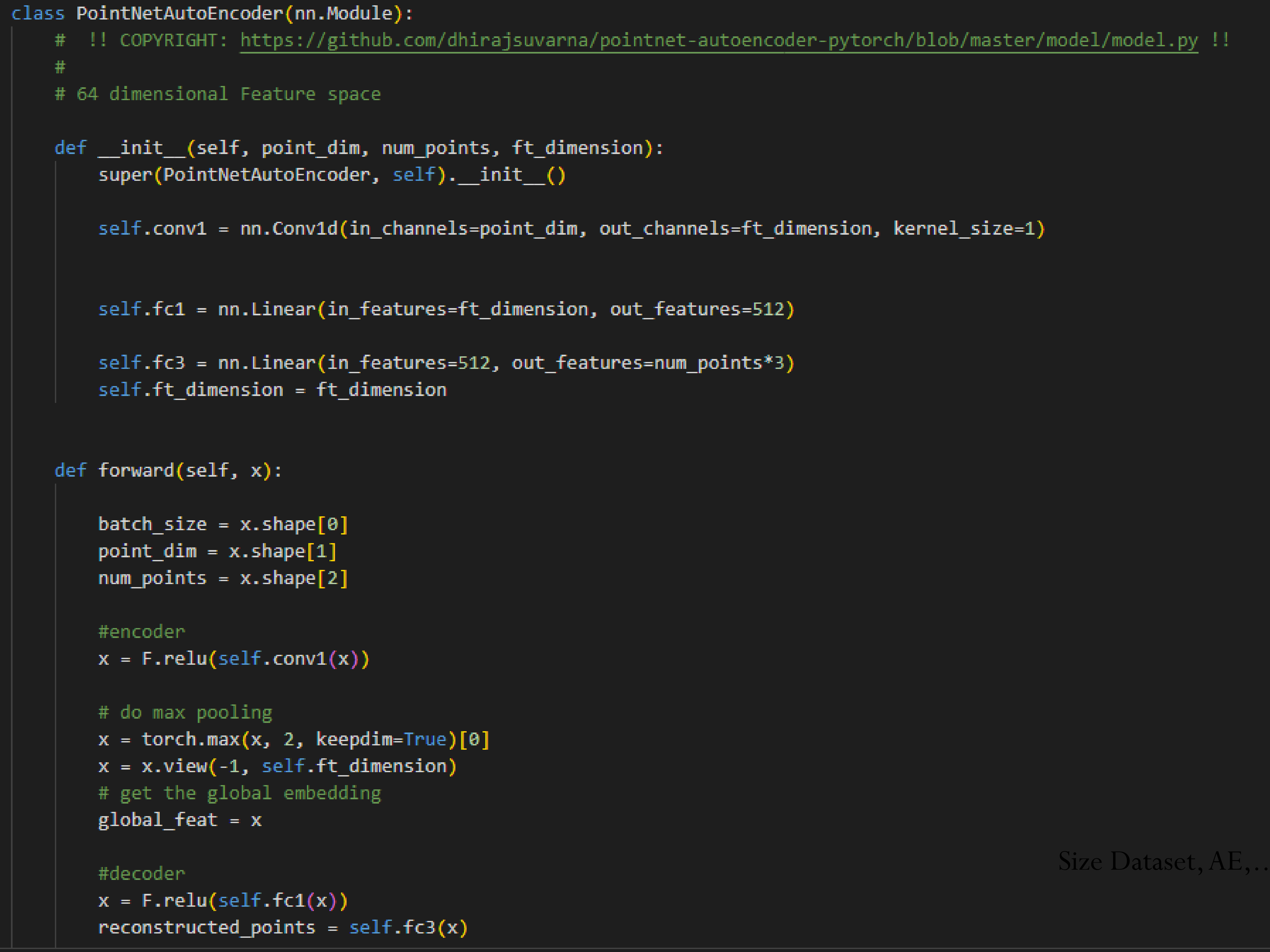

0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

Differences are marginal



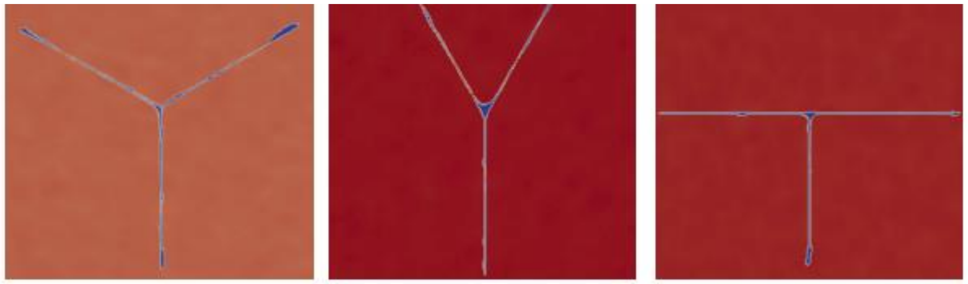





```python
class PointNetAutoEncoder(nn.Module):
    #  !! COPYRIGHT: https://github.com/dhirajsuvarna/pointnet-autoencoder-pytorch/blob/master/model/model.py !!
    #
    # 64 dimensional Feature space

    def __init__(self, point_dim, num_points, ft_dimension):
        super(PointNetAutoEncoder, self).__init__()

        self.conv1 = nn.Conv1d(in_channels=point_dim, out_channels=ft_dimension, kernel_size=1)


        self.fc1 = nn.Linear(in_features=ft_dimension, out_features=512)

        self.fc3 = nn.Linear(in_features=512, out_features=num_points*3)
        self.ft_dimension = ft_dimension


    def forward(self, x):

        batch_size = x.shape[0]
        point_dim = x.shape[1]
        num_points = x.shape[2]

        #encoder
        x = F.relu(self.conv1(x))

        # do max pooling
        x = torch.max(x, 2, keepdim=True)[0]
        x = x.view(-1, self.ft_dimension)
        # get the global embedding
        global_feat = x

        #decoder
        x = F.relu(self.fc1(x))
        reconstructed_points = self.fc3(x)
```

Size Dataset, AE,..

Thesis..

- Images of training AE seperately
- 

(a) $\frac{2\pi}{3}$ (b) $\frac{5}{6}\pi$ (c) $\frac{\pi}{2}$

Figure 6.17: Reconstructing slices at different angles

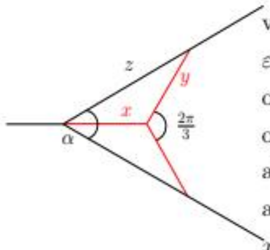


Figure 6.18: Perimeter for slices

As ε tends to zero, the Ambrosio-Tortorelli based loss does not converge against a minimal perimeter solution, but for a fixed and small ε, we can show that it behaves similarly. We can see this in the profiles of the reconstructed slices, where angles below $\frac{2\pi}{3}$ are being flattened out. This is because, in these cases, we can enter a cross inside with all angles being $\frac{2\pi}{3}$. Then the perimeter is minimized by this cross and not the original shape, i.e., referring to (??), one can show that $x + 2y < 2z$.

$$y + 2z < 2x$$

$$\sin\left(\frac{\pi}{3} - \frac{\alpha}{2}\right) + 2\sin\left(\frac{\alpha}{2}\right) < \sqrt{3}$$

$$\sin\left(\frac{\alpha}{2}\right)\left(2 - \cos\left(\frac{\pi}{3}\right)\right) + \sin\left(\frac{\pi}{3}\right)\cos\left(\frac{\alpha}{2}\right) < \sqrt{3}$$