CSE275 HW2

Problem 1: Deform a shape

In this problem, we will practice part of what we learned in the image-to-3D lecture for shape deformation.

1. Laplacian

Given a mesh $M = (V, E, F)$, we assume that the adjacency matrix is $A \in \mathbb{R}^{n \times n}$, $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix where $D[i, i]$ is the degree of the i-th vertex. The Laplacian matrix is defined as $L = D - A$.

Prove that:

(a) $\sum_{(i,j) \in E} ||x_i - x_j||^2 = x^T L x$ for $x \in \mathbb{R}^n$. [1pt]

(b) $L \in \mathbb{S}^n_+$, i.e., L is a symmetric and positive semi-definite matrix. [1pt]

(c) For the data matrix $P \in \mathbb{R}^{n \times 3}$ where each row corresponds to a point in $\mathbb{R}^3$, denote the columns of P as $P = [x, y, z]$ and rows of P as $P = [p_1^T; p_2^T; \ldots; p_n^T]$, show that $\sum_{(i,j) \in E} ||p_i - p_j||^2 = x^T L x + y^T L y + z^T L z$. (hint: Use the conclusion from 1(a)) [1pt]

(a):

$$A[i, j] = \begin{cases} 1, & (i, j) \in E \\ 0, & otherwise \end{cases}$$

Denote $D[i, i] = d_i$, we have:

$$L[i, j] = \begin{cases} d_i, & i = j \\ -1, & (i, j) \in E, i \neq j \\ 0, & otherwise \end{cases}$$

Hence L is symmetric. (1)

$$x^T L x = \sum_{i=1}^{|V|} d_i (x_i)^2 + \sum_{(i,j) \in E} (-2 x_i x_j) (2)$$

Since $\sum_{i=1}^{|V|} d_i = 2|E|$, each edge $(i, j) \in E$ can be grouped with 2 vertices i and j. So we have:

$$\begin{aligned} (2) &= \sum_{(i,j) \in E} (x_i^2 - 2 x_i x_j + x_j^2) \\ &= \sum_{(i,j) \in E} ||x_i - x_j||^2 \end{aligned}$$

So the equation is proved.

(b):

From (a) $=>$ For $\forall x \in R^n : x^T L x = \sum_{(i,j) \in E} ||x_i - x_j||^2 \geq 0$

$=>$ L is Positive Semi-Definite.

L is also symmetric, as shown in (a) at (1).

(c):

From (a) we can get:

$$
\begin{aligned}
x^TLx + y^TLy + z^TLz &= \sum_{(i,j)\in E} ||x_i - x_j||^2 + \sum_{(i,j)\in E} ||y_i - y_j||^2 + \sum_{(i,j)\in E} ||z_i - z_j||^2 \\
&= \sum_{(i,j)\in E} (||x_i - x_j||^2 + ||y_i - y_j||^2 + ||z_i - z_j||^2) \\
&= \sum_{(i,j)\in E} ||[x_i, y_i, z_i]^T - [x_j, y_j, z_j]^T||^2 \\
&= \sum_{(i,j)\in E} ||p_i - p_j||^2
\end{aligned}
$$

Hence the equation is proved.

2. Normalized Laplacian

Normalized Laplacian is defined as the normalized version of the Laplacian matrix above:

$$
L_{norm} = D^{-1}L
$$

(a) Prove that the sum of each row of L_{norm} is 0. [1pt]

$D^{-1} = diag(\frac{1}{d_1}, \frac{1}{d_2}, \ldots, \frac{1}{d_n})$. Denote $A = [\vec{a_1}, \vec{a_2}, \ldots, \vec{a_n}]$. $D^{-1}A = [\frac{1}{d_1}\vec{a_1}, \frac{1}{d_2}\vec{a_2}, \ldots, \frac{1}{d_n}\vec{a_n}]$

$$
L_{norm} = D^{-1}(D - A) = I - D^{-1}A = [\vec{I_1} - \frac{1}{d_1}\vec{a_1}, \vec{I_2} - \frac{1}{d_2}\vec{a_2}, \ldots, \vec{I_n} - \frac{1}{d_n}\vec{a_n}]
$$

Where I denotes the $n * n$ identity matrix. I_i denotes the ith row of I.

Denote $L_i = \vec{I_i} - \frac{1}{d_i}\vec{a_i}, i \in \{1, 2, \ldots, n\}$. Since by definition $deg(p_i) = d_i$, we have:

$$
\sum_{j=1}^{n} L_{ij} = 1 - \frac{1}{d_i}\sum_{k=1}^{d_i} 1 = 1 - 1 = 0
$$

So the sum of each row of L_{norm} is 0. Proved.

(b) The difference between a vertex x and the average position of its 1-ring neighborhood is a quantity that provides interesting geometric insight of the shape. It can be shown that,

$$
x - \frac{1}{|N(x)|}\sum_{y_i \in N(x)} y_i \approx H\vec{n}\Delta A
$$

for a good mesh, where $N(x)$ is the 1-ring neighorhood vertices of x by the mesh topology, $H = \frac{1}{2}(\kappa_{min} + \kappa_{\max})$ is the mean curvature at x (in the sense of the underlying continuous surface being approximated), $\vec{n}$ is the surface normal vector at x, and ΔA is a quantity proportional to the total area of the 1-ring fan (triangles formed by x and vertices along the 1-ring).

Define $\Delta p_i := p_i - \frac{1}{|N(p_i)|}\sum_{p_j \in N(p_i)} p_j$. Prove that $\Delta p_i = [L_{norm}P]_i$, where P and p_i are defined as in 1(c), and $[X]_i$ is to access the i-th row of X. [1pt]

$$
\begin{aligned}
L_{norm}P &= (I - D^{-1}A)P \\
&= P - D^{-1}AP \\
&= [p_1, p_2, \ldots, p_n] - [\frac{1}{d_1}\vec{a_1}P, \frac{1}{d_2}\vec{a_2}P, \ldots, \frac{1}{d_n}\vec{a_n}P] \\
&= [p_1 - \frac{1}{d_1}\vec{a_1}P, p_2 - \frac{1}{d_2}\vec{a_2}P, \ldots, p_n - \frac{1}{d_n}\vec{a_n}P]
\end{aligned}
$$

For the ith row of the above column vector, and from the fact that $d_i = |N(p_i)|$ by definition of vertex degree, we have:

$$
\begin{aligned}
{}_i &= p_i - \frac{1}{d_i}\vec{a}_i P \\
&= p_i - \frac{1}{|N(p_i)|}\vec{a}_i[p_1, p_2, \ldots, p_n] \quad (1)
\end{aligned}
$$

Since

$$
(\vec{a}_i)_j = \begin{cases} 1, & p_j \in N(p_i) \\ 0, & otherwise \end{cases}
$$

We can further simplify (1) to:

$$
\begin{aligned}
p_i - \frac{1}{|N(p_i)|}\vec{a}_i[p_1, p_2, \ldots, p_n] &= p_i - \frac{1}{|N(p_i)|}\sum_{p_j \in N(p_i)} p_j \\
&= \Delta p_i
\end{aligned}
$$

Hence, $[L_{norm}P]_i = \Delta p_i$ proved.

3. Shape Deformation (extra credit)

Please load the source.obj and target.obj files using the trimesh library of Python, and optimize to deform the vertices of the source.obj to match target.obj. Plot the source object, target object, and deformed object. [5pt]

(a) Chamfer-only loss

In [ ]:
```
"""Load point cloud data"""
import trimesh
# import collada
import warnings
import numpy as np
warnings.filterwarnings("ignore")

meshSrc = trimesh.load('source.obj')
vertSrc = np.asarray(meshSrc.vertices)

meshTg = trimesh.load('target.obj')
vertTg = np.asarray(meshTg.vertices)
```

In [ ]:
```
"""Chamfer distance and visualization function definition"""
import torch
import numpy as np

# helper functions for computing Chamfer distance
# params:
#   xyz1: 1 x points x 3, point cloud 1 (pc1)
#   xyz2: 1 x points x 3, point cloud 2 (pc2)
# output:
#   dist1: 1 x points in point cloud 1, the nearest neighbor distance for each point of pc1 to pc2
#   idx1: 1 x points in point cloud 1, for each point of pc1, index of the nearest neighbor in pc2
#   dist2: 1 x points in point cloud 2, the nearest neighbor distance for each point of pc2 to pc1
#   idx2: 1 x points in point cloud 2, for each point of pc2, index of the nearest neighbor in pc1
def bpdist2(feature1, feature2, data_format='NWC'):
    """This version has a high memory usage but more compatible(accurate) with optimized Chamfer Distance."""
    if data_format == 'NCW':
        diff = feature1.unsqueeze(3) - feature2.unsqueeze(2)
        distance = torch.sum(diff ** 2, dim=1)
    elif data_format == 'NWC':
        diff = feature1.unsqueeze(2) - feature2.unsqueeze(1)
        distance = torch.sum(diff ** 2, dim=3)
    else:
        raise ValueError('Unsupported data format: {}'.format(data_format))
    return distance

def Chamfer_distance_torch(xyz1, xyz2, data_format='NWC'):
    """Chamfer loss computation function"""
    # print(torch.is_tensor(xyz1), (xyz1.dim()))
    assert torch.is_tensor(xyz1) and xyz1.dim() == 3
```

```
    assert torch.is_tensor(xyz2) and xyz2.dim() == 3
    if data_format == 'NCW':
        assert xyz1.size(1) == 3 and xyz2.size(1) == 3
    elif data_format == 'NWC':
        assert xyz1.size(2) == 3 and xyz2.size(2) == 3
    distance = bpdist2(xyz1, xyz2, data_format)
    dist1, idx1 = distance.min(2)
    dist2, idx2 = distance.min(1)
    return torch.sum(dist1**2)+torch.sum(dist2**2)
```

In [ ]:

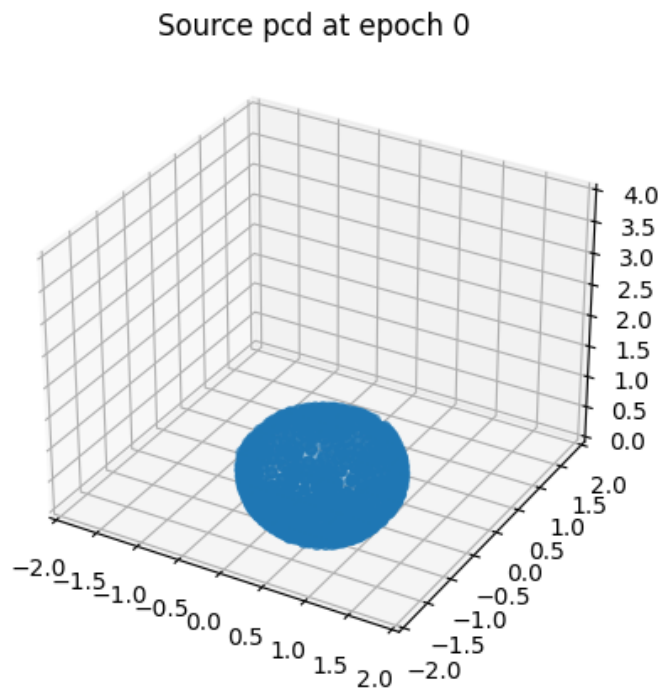
```
"""Visualization utilies."""

# You can use other visualization from previous homeworks, like Open3D
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def show_points(points, title):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_title(title)
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points[:, 0], points[:, 2], points[:, 1])


def compare_points(points1, points2):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points1[:, 0], points1[:, 2], points1[:, 1])
    ax.scatter(points2[:, 0], points2[:, 2], points2[:, 1])
```

In [ ]:

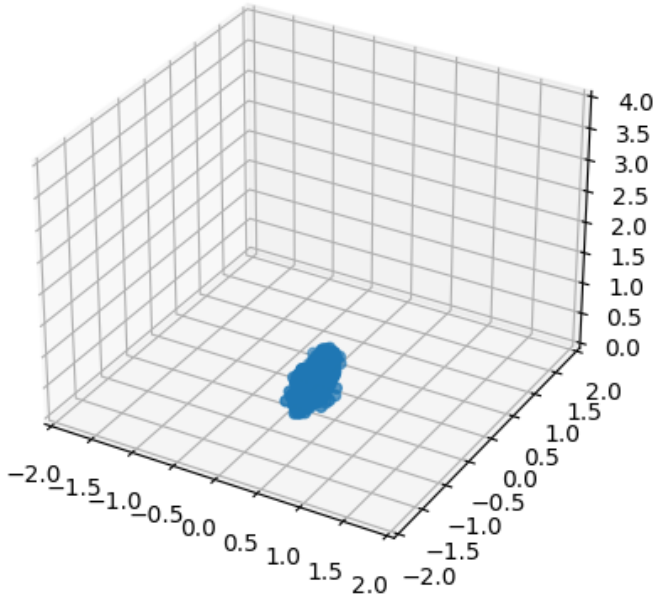
```
"""Build optimizer for registration of point clouds (Using Chamfer-only Loss)"""
import torch

LR = 0.04
EPOCH = 1000

# plot data
# plt.scatter(x,y)
# plt.show()

# Dataset loading
vertSrcTen = torch.tensor(vertSrc)
vertTgTen = torch.tensor(vertTg)
vertSrcTen.requires_grad_()

# Optimizer and Loss function definitions
optimizer_Adam = torch.optim.Adam([vertSrcTen], lr = LR, betas = (0.9,0.99))
loss_func = Chamfer_distance_torch # Using Chamfer-only loss
losses = []

vertSrcTen = vertSrcTen.unsqueeze(0)
vertTgTen = vertTgTen.unsqueeze(0)
print(vertSrcTen.shape)
print(vertTgTen.shape)
for epoch in range(EPOCH):
    optimizer_Adam.zero_grad()
    loss = loss_func(vertSrcTen, vertTgTen)
    loss.backward()
    optimizer_Adam.step()
    if ((epoch) % 10 == 0):
        print("Epoch {}. Chamfer Loss: {}".format(epoch, loss.item()))
    if ((epoch) % 100 == 0):
        show_points(vertSrcTen.squeeze(0).detach(), "Source pcd at epoch {}".format(epoch))
    losses.append(loss.item())

show_points(vertTgTen.squeeze(0), "Target pcd")
```

```
torch.Size([1, 962, 3])
torch.Size([1, 1502, 3])
Epoch 0. Chamfer Loss: 545.5872855331091
Epoch 10. Chamfer Loss: 22.241186323397308
Epoch 20. Chamfer Loss: 1.7525544605093757
Epoch 30. Chamfer Loss: 0.37065928789485564
Epoch 40. Chamfer Loss: 0.1935712052749547
Epoch 50. Chamfer Loss: 0.1310796831860575
Epoch 60. Chamfer Loss: 0.09488000572030716
Epoch 70. Chamfer Loss: 0.07122811377075337
Epoch 80. Chamfer Loss: 0.055093302427558305
Epoch 90. Chamfer Loss: 0.043670949006195406
Epoch 100. Chamfer Loss: 0.03531902965060082
Epoch 110. Chamfer Loss: 0.02902171283414586
Epoch 120. Chamfer Loss: 0.024186919150026975
Epoch 130. Chamfer Loss: 0.020392846173630852
Epoch 140. Chamfer Loss: 0.017359350986911392
Epoch 150. Chamfer Loss: 0.014900256221778634
Epoch 160. Chamfer Loss: 0.012879574456710098
Epoch 170. Chamfer Loss: 0.01119585053041943
Epoch 180. Chamfer Loss: 0.009779308571609635
Epoch 190. Chamfer Loss: 0.00857949617484338
Epoch 200. Chamfer Loss: 0.007558039615781184
Epoch 210. Chamfer Loss: 0.006683951297051941
Epoch 220. Chamfer Loss: 0.005930048038450961
Epoch 230. Chamfer Loss: 0.005276364069860536
Epoch 240. Chamfer Loss: 0.004709562205685467
Epoch 250. Chamfer Loss: 0.004215529665595679
Epoch 260. Chamfer Loss: 0.0037828981071723502
Epoch 270. Chamfer Loss: 0.003402457750050715
Epoch 280. Chamfer Loss: 0.003066369686709017
Epoch 290. Chamfer Loss: 0.0027680820947116156
Epoch 300. Chamfer Loss: 0.002503937401314632
Epoch 310. Chamfer Loss: 0.0022698195481199368
Epoch 320. Chamfer Loss: 0.0020628231217565945
Epoch 330. Chamfer Loss: 0.0018748674772026217
Epoch 340. Chamfer Loss: 0.0017091009102689434
Epoch 350. Chamfer Loss: 0.0015598866872898257
Epoch 360. Chamfer Loss: 0.001424478438562385
Epoch 370. Chamfer Loss: 0.0013016478598930196
Epoch 380. Chamfer Loss: 0.001190549122778952
Epoch 390. Chamfer Loss: 0.0010917471367226802
Epoch 400. Chamfer Loss: 0.0010044096729020966
Epoch 410. Chamfer Loss: 0.0009264202001211349
Epoch 420. Chamfer Loss: 0.0008548678490548401
Epoch 430. Chamfer Loss: 0.0007896373115746124
Epoch 440. Chamfer Loss: 0.0007314443144612048
Epoch 450. Chamfer Loss: 0.0006778944024088962
Epoch 460. Chamfer Loss: 0.0006291099225977383
Epoch 470. Chamfer Loss: 0.0005849472441197268
Epoch 480. Chamfer Loss: 0.000546521904638223
Epoch 490. Chamfer Loss: 0.0005108114775895457
Epoch 500. Chamfer Loss: 0.0004790717565593138
Epoch 510. Chamfer Loss: 0.0004461908378516722
Epoch 520. Chamfer Loss: 0.00041318680260612507
Epoch 530. Chamfer Loss: 0.00038820015745108223
Epoch 540. Chamfer Loss: 0.00036589492204140445
Epoch 550. Chamfer Loss: 0.0003458473933824268
Epoch 560. Chamfer Loss: 0.0003267913008065577
Epoch 570. Chamfer Loss: 0.00030943706154454385
Epoch 580. Chamfer Loss: 0.000294050660140487
Epoch 590. Chamfer Loss: 0.00028004529429283823
Epoch 600. Chamfer Loss: 0.00026755026138832995
Epoch 610. Chamfer Loss: 0.000256445369467673
Epoch 620. Chamfer Loss: 0.000246357705690297
Epoch 630. Chamfer Loss: 0.00023682342831370985
Epoch 640. Chamfer Loss: 0.00022774668110817265
Epoch 650. Chamfer Loss: 0.00021934789002993793
Epoch 660. Chamfer Loss: 0.00021144454266453064
Epoch 670. Chamfer Loss: 0.00020464248383296756
Epoch 680. Chamfer Loss: 0.000198215299568516
Epoch 690. Chamfer Loss: 0.00019282379651933476
Epoch 700. Chamfer Loss: 0.0001880562657764212
Epoch 710. Chamfer Loss: 0.0001837587792154799
Epoch 720. Chamfer Loss: 0.0001797667394488275
Epoch 730. Chamfer Loss: 0.00017546550146181977
```

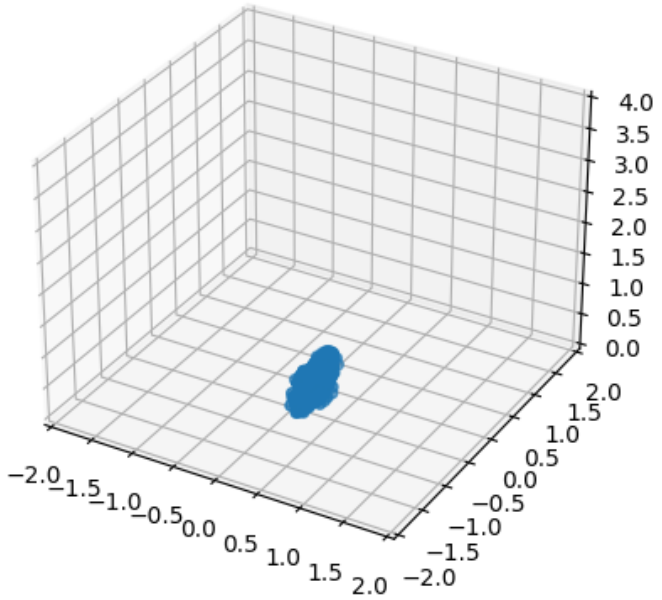
```
Epoch 740. Chamfer Loss: 0.00017198341188114297
Epoch 750. Chamfer Loss: 0.0001689575595958932
Epoch 760. Chamfer Loss: 0.0001661333936811489
Epoch 770. Chamfer Loss: 0.0001629763493083504
Epoch 780. Chamfer Loss: 0.0001593156482549083
Epoch 790. Chamfer Loss: 0.00015625713684363173
Epoch 800. Chamfer Loss: 0.00015369494629381345
Epoch 810. Chamfer Loss: 0.00015178363895261218
Epoch 820. Chamfer Loss: 0.00014979816982757196
Epoch 830. Chamfer Loss: 0.0001480177051629028
Epoch 840. Chamfer Loss: 0.00014591089495866337
Epoch 850. Chamfer Loss: 0.0001445025965225835
Epoch 860. Chamfer Loss: 0.0001432929750469018
Epoch 870. Chamfer Loss: 0.00014221684287275477
Epoch 880. Chamfer Loss: 0.0001412289473345133
Epoch 890. Chamfer Loss: 0.0001402889333198886
Epoch 900. Chamfer Loss: 0.0001394892564399651
Epoch 910. Chamfer Loss: 0.00013879421668743956
Epoch 920. Chamfer Loss: 0.0001382175875963266
Epoch 930. Chamfer Loss: 0.0001375740746992411
Epoch 940. Chamfer Loss: 0.0001374169007324574
Epoch 950. Chamfer Loss: 0.00013596145548126
Epoch 960. Chamfer Loss: 0.00013508679009141964
Epoch 970. Chamfer Loss: 0.0001346011434481508
Epoch 980. Chamfer Loss: 0.00013418793383471094
Epoch 990. Chamfer Loss: 0.0001340624098705785
```



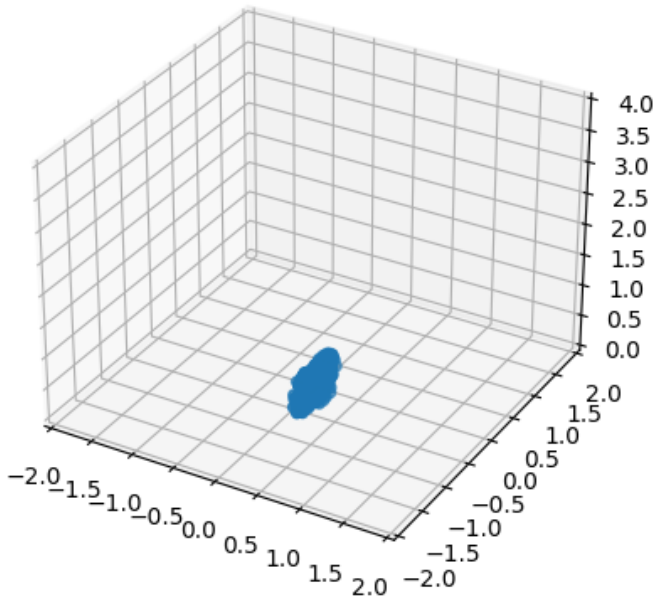
Source pcd at epoch 100



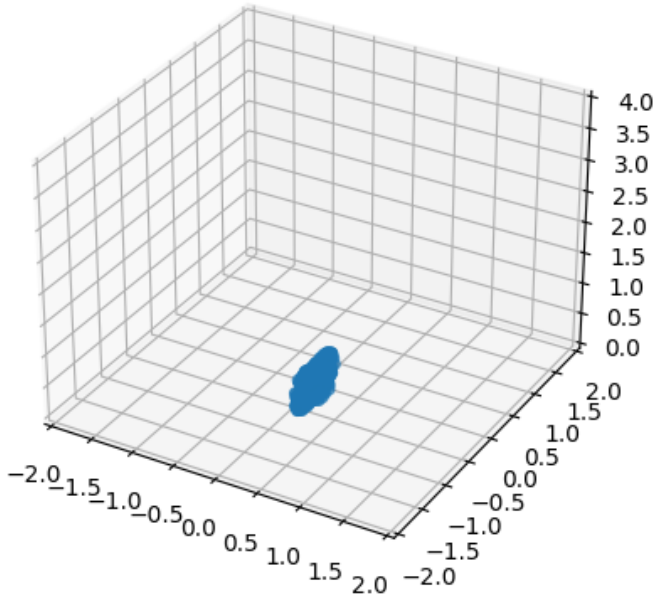
Source pcd at epoch 200



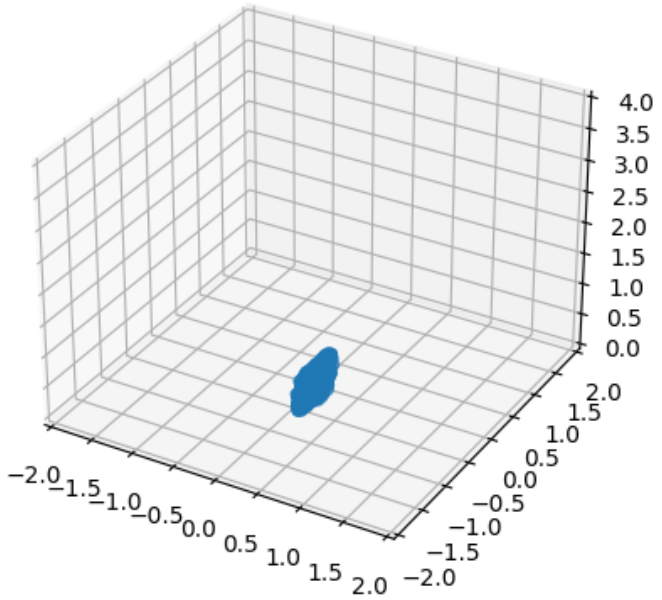
Source pcd at epoch 300



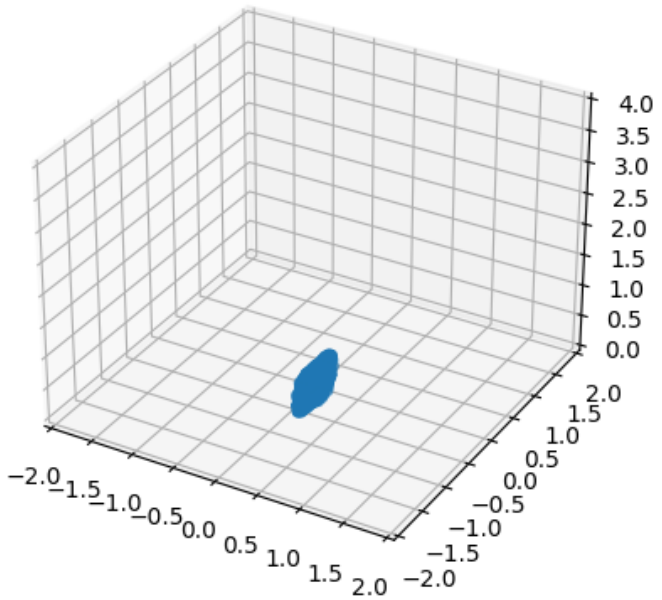
Source pcd at epoch 400



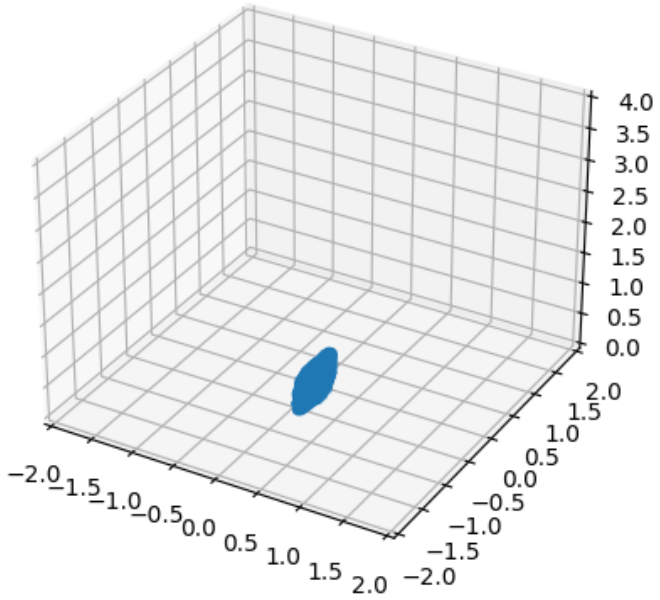
Source pcd at epoch 500



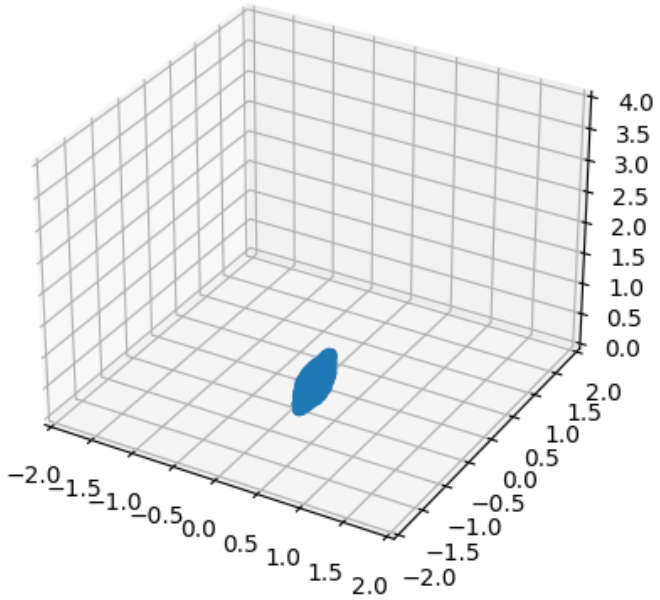
Source pcd at epoch 600



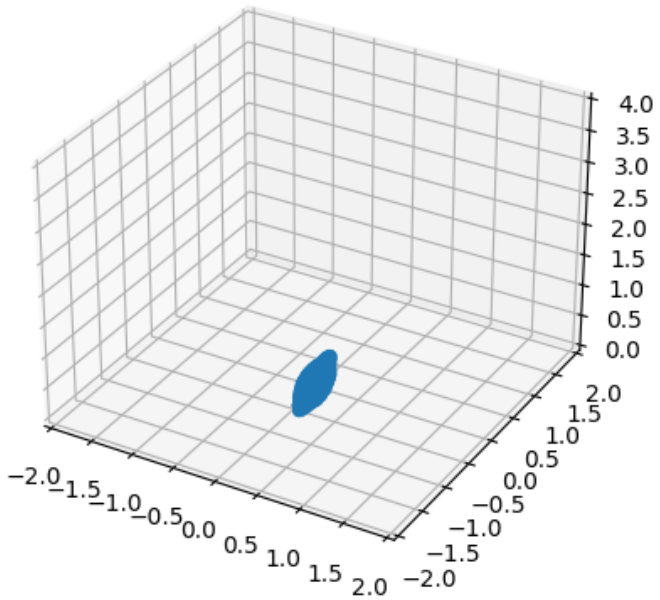
Source pcd at epoch 700

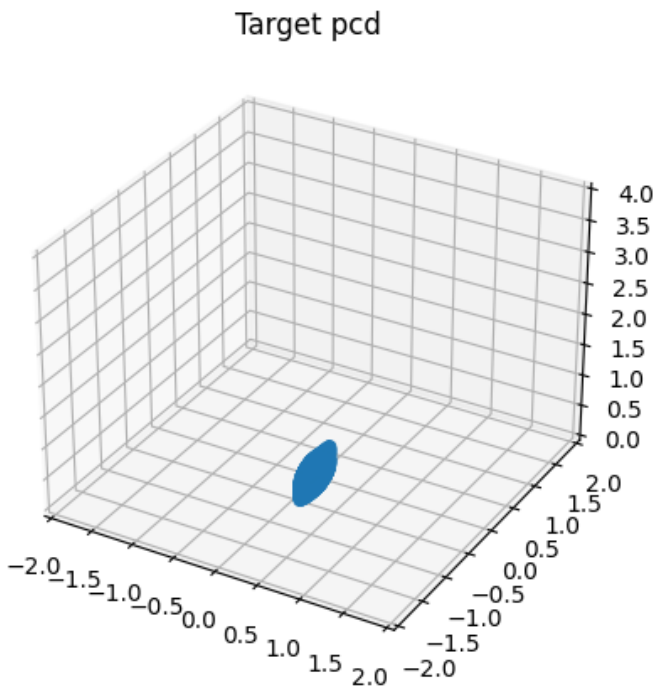


Source pcd at epoch 800



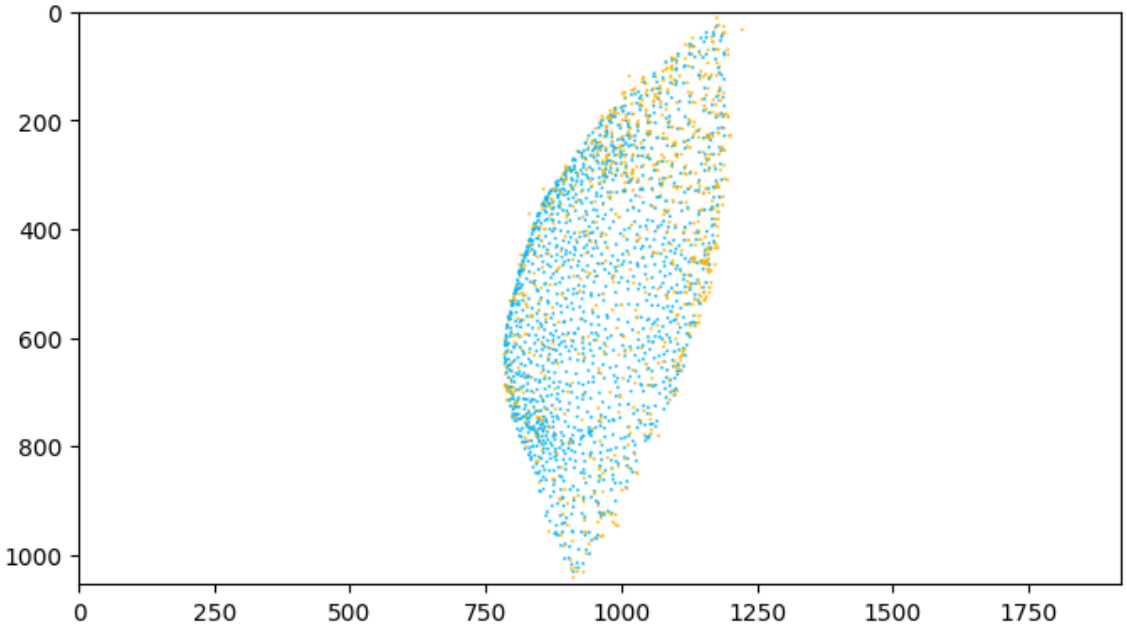
Source pcd at epoch 900



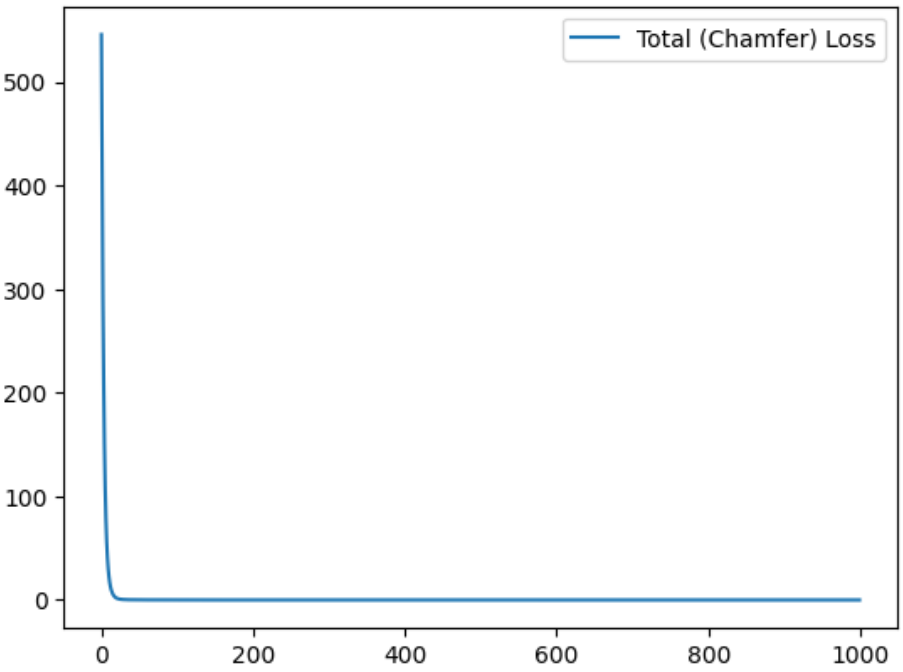


In [ ]:
```python
import open3d

vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1e-4, 1))
    view_ctl.set_front((0, 0.5, 2))
    view_ctl.set_lookat((0, 0, 0))
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(8,6))
    plt.imshow(np.asarray(img)[::-1, ::-1])
    for g in geoms:
        vis.remove_geometry(g)


pcd = open3d.geometry.PointCloud()
pcd1 = open3d.geometry.PointCloud()
src = vertSrcTen.squeeze(0).detach().numpy()
tg = vertTgTen.squeeze(0).detach().numpy()
print(src.shape)
pcd.points = open3d.utility.Vector3dVector(src)
pcd.paint_uniform_color([1, 0.706, 0])
# print(np.asarray(pcd.colors))
pcd1.points = open3d.utility.Vector3dVector(tg)
pcd1.paint_uniform_color([0, 0.706, 1])

draw_geometries([pcd, pcd1])
```

```
(962, 3)
```



In [ ]:
```
# plot Loss lines
x = list(range(len(losses)))
plt.plot(x, losses, label = "Total (Chamfer) Loss")
plt.legend()
plt.show()
```



(b) Curvature and normal-based loss

In [ ]:
```
"""Loss computation functions"""
import trimesh
import numpy as np
import networkx as nx


def Compute_Delta_pi(pcd, edges):
    """
    Compute delta pi for every point of a given mesh (differentiable inputs to differentiable mediums)
    pcd: point cloud input
    edges: points' connections
    return -> 1*n*3 delta pi matrix
    """
    g = nx.from_edgelist(edges)
    one_ring = np.array([list(g[i].keys()) for i in range(len(g))])

    delta_p = torch.zeros((len(one_ring), 3))
    ind = 0
```

```
    for i, x in zip(one_ring, pcd[0]):
        delta_p[ind] = x - torch.mean(pcd[0][i], axis=0)
        ind += 1
    return delta_p.unsqueeze(0)


def Normal_curvature_loss_torch(srcDelpi, tgDelpi, idx1, idx2):
    """
    Compute delta pi loss for src and tg point clouds.
    idx1: 1 x points in point cloud 1, for each point of pc1, index of the nearest neighbor in pc2.
    idx2: 1 x points in point cloud 2, for each point of pc2, index of the nearest neighbor in pc1.
    srcDelpi, tgDelpi: delta pis for source and target point clouds.
    """
    return torch.sum((srcDelpi[0] - tgDelpi[0][idx1, :])**2) + torch.sum((tgDelpi[0] - srcDelpi[0][idx2, :])**2)

# helper functions for computing Chamfer distance
def bpdist2(feature1, feature2, data_format='NWC'):
    """This version has a high memory usage but more compatible(accurate) with optimized Chamfer Distance."""
    if data_format == 'NCW':
        diff = feature1.unsqueeze(3) - feature2.unsqueeze(2)
        distance = torch.sum(diff ** 2, dim=1)
    elif data_format == 'NWC':
        diff = feature1.unsqueeze(2) - feature2.unsqueeze(1)
        distance = torch.sum(diff ** 2, dim=3)
    else:
        raise ValueError('Unsupported data format: {}'.format(data_format))
    return distance

# params:
#   xyz1: 1 x points x 3, point cloud 1 (pc1)
#   xyz2: 1 x points x 3, point cloud 2 (pc2)
# output:
#   dist1: 1 x points in point cloud 1, the nearest neighbor distance for each point of pc1 to pc2
#   idx1: 1 x points in point cloud 1, for each point of pc1, index of the nearest neighbor in pc2
#   dist2: 1 x points in point cloud 2, the nearest neighbor distance for each point of pc2 to pc1
#   idx2: 1 x points in point cloud 2, for each point of pc2, index of the nearest neighbor in pc1
def Chamfer_distance_torch_with_correspondances(xyz1, xyz2, data_format='NWC'):
    """
    Compute the Chamfer distance loss, together with correspondance information
    """
    # print(torch.is_tensor(xyz1), (xyz1.dim()))
    assert torch.is_tensor(xyz1) and xyz1.dim() == 3
    assert torch.is_tensor(xyz2) and xyz2.dim() == 3
    if data_format == 'NCW':
        assert xyz1.size(1) == 3 and xyz2.size(1) == 3
    elif data_format == 'NWC':
        assert xyz1.size(2) == 3 and xyz2.size(2) == 3
    distance = bpdist2(xyz1, xyz2, data_format)
    dist1, idx1 = distance.min(2)
    dist2, idx2 = distance.min(1)
    chamfer = torch.sum(dist1**2)+torch.sum(dist2**2) # compute the chamfer distance
    return chamfer, dist1, idx1, dist2, idx2

def Total_loss(srcPcd, tgPcd, srcDelpi, tgDelpi, epi=0.575):
    """
    Compute total pcd deformation loss (containing Chamfer loss and the curvature and normal loss)
    epi: The relative importance of normal curvature loss
    """
    ChamferLoss, dist1, idx1, dist2, idx2 = Chamfer_distance_torch_with_correspondances(srcPcd, tgPcd)
    NormalCurvatureLoss = Normal_curvature_loss_torch(srcDelpi, tgDelpi, idx1, idx2)
    return ChamferLoss, NormalCurvatureLoss
```

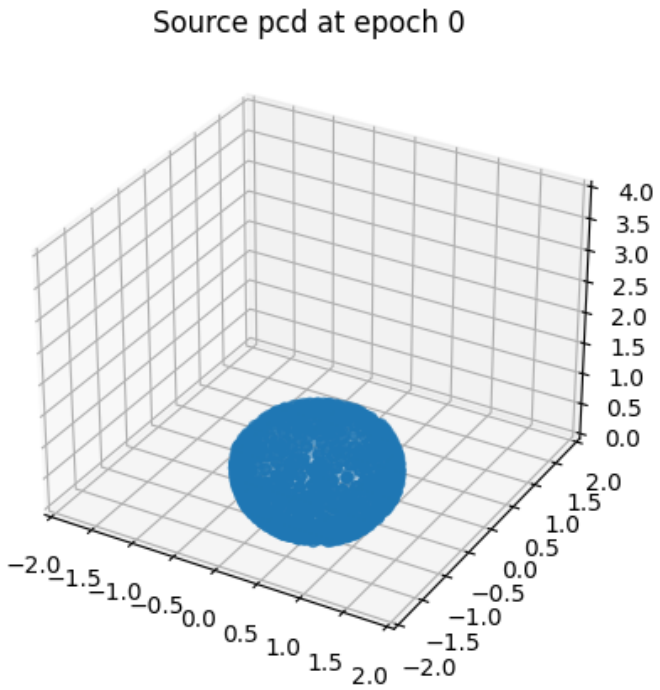
In [ ]:

```
"""Build optimizer for registration of point clouds (using Chamfer+Curvature Loss)"""
import torch
import matplotlib.pyplot as plt

LR = 0.03
EPOCH = 1000

# Dataset loading
vertSrcTen = torch.tensor(vertSrc)
vertSrcTen = vertSrcTen.unsqueeze(0)
vertTgTen = torch.tensor(vertTg)
vertTgTen = vertTgTen.unsqueeze(0)
vertSrcTen.requires_grad_()
```

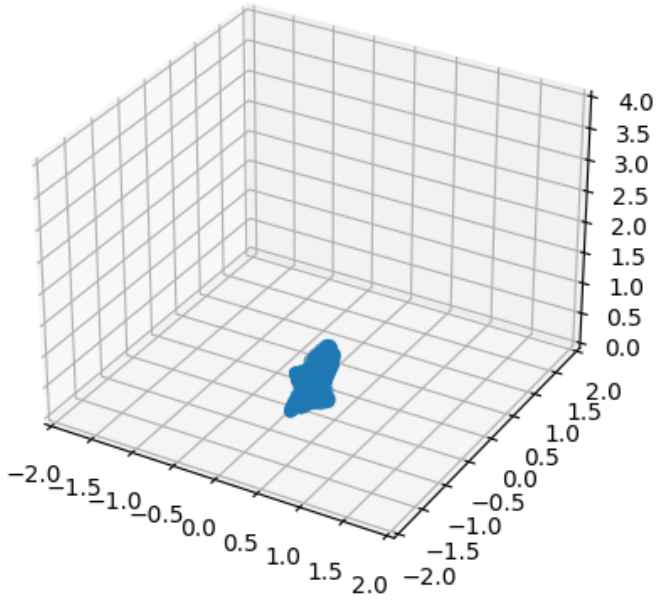
```
srcDelpiTen = Compute_Delta_pi(vertSrcTen, meshSrc.edges)
tgDelpiTen = Compute_Delta_pi(vertTgTen, meshTg.edges)
assert len(tgDelpiTen.shape) == len(srcDelpiTen.shape) == 3

# Optimizer and Loss function definitions
optimizer_Adam = torch.optim.Adam([vertSrcTen], lr = LR, betas = (0.9,0.99))
loss_func = Total_loss
epi = 0.575 # importance of Normal and Curvature loss compared to Chamfer loss
losses = [] # Total loss
cf_losses = [] # Chamfer distance loss
curv_losses = [] # Normal and Curvature loss

print(vertSrcTen.shape)
print(vertTgTen.shape)
for epoch in range(EPOCH):
    # TODO: Write a function to compute src and tg Delta pi on every epoch
    optimizer_Adam.zero_grad()
    srcDelpiTen = Compute_Delta_pi(vertSrcTen, meshSrc.edges)
    cf_loss, curv_loss = loss_func(vertSrcTen, vertTgTen, srcDelpiTen, tgDelpiTen)
    loss = (1-epi) * cf_loss + epi * curv_loss
    loss.backward()
    optimizer_Adam.step()
    if ((epoch) % 10 == 0):
        print("Epoch {}. CF Loss: {}. Curvature Loss: {}. Total Loss: {}".format(epoch, cf_loss.item(), curv_loss.
    if ((epoch) % 100 == 0):
        show_points(vertSrcTen.squeeze(0).detach(), "Source pcd at epoch {}".format(epoch))

    cf_losses.append(cf_loss.item())
    curv_losses.append(curv_loss.item())
    losses.append(loss.item())



# show result
show_points(vertTgTen.squeeze(0), "Target pcd")
```

```
torch.Size([1, 962, 3])
torch.Size([1, 1502, 3])
Epoch 0. CF Loss: 545.5872855331091. Curvature Loss: 1.399462103843689. Total Loss: 232.67928707022222
Epoch 10. CF Loss: 49.048734242357476. Curvature Loss: 3.36802339553833. Total Loss: 22.78232549351554
Epoch 20. CF Loss: 6.543107606908908. Curvature Loss: 3.6454038619995117. Total Loss: 4.876927977427863
Epoch 30. CF Loss: 1.429741103624375. Curvature Loss: 1.9452439546585083. Total Loss: 1.726155222107376
Epoch 40. CF Loss: 0.4975591090922875. Curvature Loss: 1.3604191541671753. Total Loss: 0.9937036081882579
Epoch 50. CF Loss: 0.25424099658088056. Curvature Loss: 1.357884407043457. Total Loss: 0.8888359575968621
Epoch 60. CF Loss: 0.17303430352868512. Curvature Loss: 0.8381080031394958. Total Loss: 0.5554516852752497
Epoch 70. CF Loss: 0.1346005486086644. Curvature Loss: 0.6208484172821045. Total Loss: 0.414193067135428
Epoch 80. CF Loss: 0.10686501417064388. Curvature Loss: 0.4025413393974304. Total Loss: 0.2768789026661623
Epoch 90. CF Loss: 0.08852875704961377. Curvature Loss: 0.33281996846199036. Total Loss: 0.22899619243585942
Epoch 100. CF Loss: 0.07569686037772229. Curvature Loss: 0.2947402596473694. Total Loss: 0.201646810487421
Epoch 110. CF Loss: 0.06635883794432766. Curvature Loss: 0.2561812996864319. Total Loss: 0.17550674301522476
Epoch 120. CF Loss: 0.05874263980230783. Curvature Loss: 0.25047048926353455. Total Loss: 0.1689861495172229
Epoch 130. CF Loss: 0.05256859580872933. Curvature Loss: 0.23975244164466858. Total Loss: 0.16019929896875573
Epoch 140. CF Loss: 0.04632935665160978. Curvature Loss: 0.21214047074317932. Total Loss: 0.14167074799932033
Epoch 150. CF Loss: 0.040463034010783994. Curvature Loss: 0.21446651220321655. Total Loss: 0.14051502950108435
Epoch 160. CF Loss: 0.03621954434724051. Curvature Loss: 0.22109709680080414. Total Loss: 0.14252412769481385
Epoch 170. CF Loss: 0.03390138826987481. Curvature Loss: 0.31890907883644104. Total Loss: 0.19778081109070844
Epoch 180. CF Loss: 0.02905237944616907. Curvature Loss: 0.33747977018356323. Total Loss: 0.20639812464982235
Epoch 190. CF Loss: 0.02772077981372871. Curvature Loss: 0.43710532784461975. Total Loss: 0.2631168926963169
Epoch 200. CF Loss: 0.0289373836788131. Curvature Loss: 0.5433261394500732. Total Loss: 0.3247109182472877
Epoch 210. CF Loss: 0.026484793763439398. Curvature Loss: 1.0804522037506104. Total Loss: 0.6325160247037404
Epoch 220. CF Loss: 0.019210307969990976. Curvature Loss: 1.56282377243042. Total Loss: 0.9067880261928797
Epoch 230. CF Loss: 0.01873570654329553. Curvature Loss: 1.0765093564987183. Total Loss: 0.6269555403665024
Epoch 240. CF Loss: 0.020692304054951198. Curvature Loss: 0.9549455046653748. Total Loss: 0.5578878929158286
Epoch 250. CF Loss: 0.017757058147387778. Curvature Loss: 0.5415743589401245. Total Loss: 0.3189519912020502
Epoch 260. CF Loss: 0.018461223734028698. Curvature Loss: 0.35855531692504883. Total Loss: 0.21401532731886527
Epoch 270. CF Loss: 0.018581482193429116. Curvature Loss: 0.3928980231285095. Total Loss: 0.23381348578051975
Epoch 280. CF Loss: 0.01768769200270804. Curvature Loss: 0.4615916907787323. Total Loss: 0.2729324905538639
Epoch 290. CF Loss: 0.016293888390632408. Curvature Loss: 0.3590654134750366. Total Loss: 0.21338750935370035
Epoch 300. CF Loss: 0.017419781154999578. Curvature Loss: 0.2625936269760132. Total Loss: 0.1583947425020824
Epoch 310. CF Loss: 0.017408146758262744. Curvature Loss: 0.23375025391578674. Total Loss: 0.14180485017820038
Epoch 320. CF Loss: 0.016706452198985. Curvature Loss: 0.21525272727012634. Total Loss: 0.13087056110994932
Epoch 330. CF Loss: 0.01532505750264642. Curvature Loss: 0.22137850522994995. Total Loss: 0.13380578249526537
Epoch 340. CF Loss: 0.014677796197976784. Curvature Loss: 0.178310826420784. Total Loss: 0.10876678447827161
Epoch 350. CF Loss: 0.01373207099805524. Curvature Loss: 0.17577019333839417. Total Loss: 0.1069039861283437
Epoch 360. CF Loss: 0.013961103926086793. Curvature Loss: 0.1599949598312378. Total Loss: 0.09793057107154862
Epoch 370. CF Loss: 0.012483440386268852. Curvature Loss: 0.16678345203399658. Total Loss: 0.10120594559359618
Epoch 380. CF Loss: 0.012489233723779355. Curvature Loss: 0.1809435784816742. Total Loss: 0.10935047674416247
Epoch 390. CF Loss: 0.012195134291871424. Curvature Loss: 0.16785424947738647. Total Loss: 0.10169912403342646
Epoch 400. CF Loss: 0.011514326139307017. Curvature Loss: 0.15274396538734436. Total Loss: 0.09272136796187043
Epoch 410. CF Loss: 0.011180969849549343. Curvature Loss: 0.1441907435655594. Total Loss: 0.08766158563843579
Epoch 420. CF Loss: 0.010767063378400185. Curvature Loss: 0.14967679977416992. Total Loss: 0.09064016180596779
Epoch 430. CF Loss: 0.01017238573441464. Curvature Loss: 0.1445980668067932. Total Loss: 0.08746715384114843
Epoch 440. CF Loss: 0.009868204783823765. Curvature Loss: 0.13869892060756683. Total Loss: 0.083945866009947
Epoch 450. CF Loss: 0.009429625079227929. Curvature Loss: 0.13796712458133698. Total Loss: 0.0833386831951213
Epoch 460. CF Loss: 0.009076168427601904. Curvature Loss: 0.13743242621421814. Total Loss: 0.08288101590984819
Epoch 470. CF Loss: 0.008952565308686941. Curvature Loss: 0.13459832966327667. Total Loss: 0.08119887869498894
Epoch 480. CF Loss: 0.008974575638995912. Curvature Loss: 0.13236388564109802. Total Loss: 0.07992342516491432
Epoch 490. CF Loss: 0.008434070761527659. Curvature Loss: 0.12754763662815094. Total Loss: 0.07692436703701672
Epoch 500. CF Loss: 0.008143413334129778. Curvature Loss: 0.13964270055294037. Total Loss: 0.0837555001321846
Epoch 510. CF Loss: 0.008124506057668233. Curvature Loss: 0.14057281613349915. Total Loss: 0.08428228211609683
Epoch 520. CF Loss: 0.00770317076331971. Curvature Loss: 0.12925726175308228. Total Loss: 0.07759677159231707
Epoch 530. CF Loss: 0.007378534092048191. Curvature Loss: 0.14501775801181793. Total Loss: 0.08652108374809646
Epoch 540. CF Loss: 0.00719542954294253. Curvature Loss: 0.1571880728006363. Total Loss: 0.09344119978864547
Epoch 550. CF Loss: 0.0068701067881688505. Curvature Loss: 0.14577367901802063. Total Loss: 0.08673966156539169
Epoch 560. CF Loss: 0.006929827361272046. Curvature Loss: 0.14937323331832886. Total Loss: 0.08883478429646359
Epoch 570. CF Loss: 0.0068055094347367635. Curvature Loss: 0.15113413333892822. Total Loss: 0.08979446817964686
Epoch 580. CF Loss: 0.006420318368240342. Curvature Loss: 0.15879100561141968. Total Loss: 0.0940334590627201
Epoch 590. CF Loss: 0.005979485228850024. Curvature Loss: 0.15846775472164154. Total Loss: 0.09366023534432776
Epoch 600. CF Loss: 0.005369482134153109. Curvature Loss: 0.16399982571601868. Total Loss: 0.09658192745855163
Epoch 610. CF Loss: 0.00503854687951153. Curvature Loss: 0.1768456995487213. Total Loss: 0.1038276589192491
Epoch 620. CF Loss: 0.004532259144522153. Curvature Loss: 0.16867080330848694. Total Loss: 0.09891192278385996
Epoch 630. CF Loss: 0.0039591196540343186. Curvature Loss: 0.14794905483722687. Total Loss: 0.08675333275689906
Epoch 640. CF Loss: 0.0035415262141097973. Curvature Loss: 0.1498345285654068. Total Loss: 0.08766000442875073
Epoch 650. CF Loss: 0.003293314613365864. Curvature Loss: 0.1441919356584549. Total Loss: 0.08431001761647273
Epoch 660. CF Loss: 0.0031082366677163296. Curvature Loss: 0.15004751086235046. Total Loss: 0.08759831411422454
Epoch 670. CF Loss: 0.003246916563261636. Curvature Loss: 0.14955124258995056. Total Loss: 0.08737190328354971
Epoch 680. CF Loss: 0.002823264058037579. Curvature Loss: 0.1430315375328064. Total Loss: 0.0834430168356813
Epoch 690. CF Loss: 0.0026195114731316104. Curvature Loss: 0.13696902990341187. Total Loss: 0.07987048159031052
Epoch 700. CF Loss: 0.002518756664236408. Curvature Loss: 0.14237865805625916. Total Loss: 0.08293819623935919
Epoch 710. CF Loss: 0.002357129690145438. Curvature Loss: 0.13541355729103088. Total Loss: 0.07886457183536427
Epoch 720. CF Loss: 0.0021512420145905708. Curvature Loss: 0.1365634948015213. Total Loss: 0.07943828624948865
Epoch 730. CF Loss: 0.002117136413800386. Curvature Loss: 0.13203273713588715. Total Loss: 0.07681860645647125
```

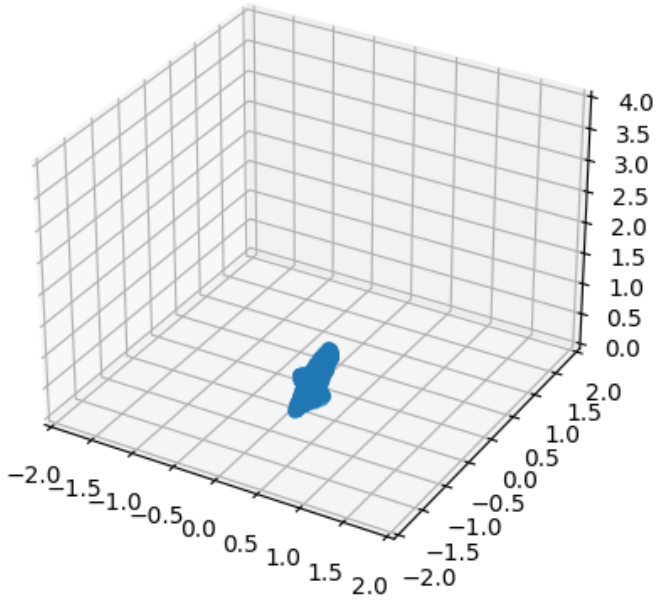
```
Epoch 740. CF Loss: 0.002095417148175134. Curvature Loss: 0.14465636014938354. Total Loss: 0.08406795490352162
Epoch 750. CF Loss: 0.0020709054385352485. Curvature Loss: 0.1504860520362854. Total Loss: 0.08740961324212547
Epoch 760. CF Loss: 0.0020308897602182804. Curvature Loss: 0.14188073575496674. Total Loss: 0.08244454934455349
Epoch 770. CF Loss: 0.002016753969893941. Curvature Loss: 0.14082717895507812. Total Loss: 0.08183274982649097
Epoch 780. CF Loss: 0.0021676341048728273. Curvature Loss: 0.14763134717941284. Total Loss: 0.08580927061284946
Epoch 790. CF Loss: 0.0020062791520892276. Curvature Loss: 0.1452294886112213. Total Loss: 0.08435962533614824
Epoch 800. CF Loss: 0.0018628893746897735. Curvature Loss: 0.14637835323810577. Total Loss: 0.08495928295879912
Epoch 810. CF Loss: 0.0018770814703819344. Curvature Loss: 0.13687793910503387. Total Loss: 0.07950257125754553
Epoch 820. CF Loss: 0.0018737378038460695. Curvature Loss: 0.13298827409744263. Total Loss: 0.07726459766278021
Epoch 830. CF Loss: 0.0018640009644070499. Curvature Loss: 0.13085158169269562. Total Loss: 0.07603185876558588
Epoch 840. CF Loss: 0.0018126548311251894. Curvature Loss: 0.1378515362739563. Total Loss: 0.08003501166075308
Epoch 850. CF Loss: 0.002000518134950594. Curvature Loss: 0.1355760246515274. Total Loss: 0.0788064347545113
Epoch 860. CF Loss: 0.001723417374977933. Curvature Loss: 0.14002954959869385. Total Loss: 0.08124943893326622
Epoch 870. CF Loss: 0.0018007493156301593. Curvature Loss: 0.1412060558795929. Total Loss: 0.08195879537450232
Epoch 880. CF Loss: 0.0017661045799050373. Curvature Loss: 0.14460116624832153. Total Loss: 0.08389626503924452
Epoch 890. CF Loss: 0.0018512305353022682. Curvature Loss: 0.1473584771156311. Total Loss: 0.08551789731899134
Epoch 900. CF Loss: 0.0018307614596763063. Curvature Loss: 0.15475061535835266. Total Loss: 0.0897596722360088
Epoch 910. CF Loss: 0.001932680399225185. Curvature Loss: 0.1479892134666443. Total Loss: 0.08591518542287505
Epoch 920. CF Loss: 0.0019205092400095278. Curvature Loss: 0.14348506927490234. Total Loss: 0.0833201312600729
Epoch 930. CF Loss: 0.0019070567619941334. Curvature Loss: 0.1513931155204773. Total Loss: 0.08786154054812195
Epoch 940. CF Loss: 0.0018829573325762658. Curvature Loss: 0.1584930419921875. Total Loss: 0.0919337545217366
Epoch 950. CF Loss: 0.0018365079197858209. Curvature Loss: 0.14528928697109222. Total Loss: 0.0843218569918741
Epoch 960. CF Loss: 0.0017667581497204678. Curvature Loss: 0.15088842809200287. Total Loss: 0.08751171352365546
Epoch 970. CF Loss: 0.0017765325880762153. Curvature Loss: 0.15230295062065125. Total Loss: 0.08832921774140044
Epoch 980. CF Loss: 0.001704552783306351. Curvature Loss: 0.156911700963974. Total Loss: 0.0909486637322483
Epoch 990. CF Loss: 0.0018862342333153797. Curvature Loss: 0.14623665809631348. Total Loss: 0.08488772497430705
```



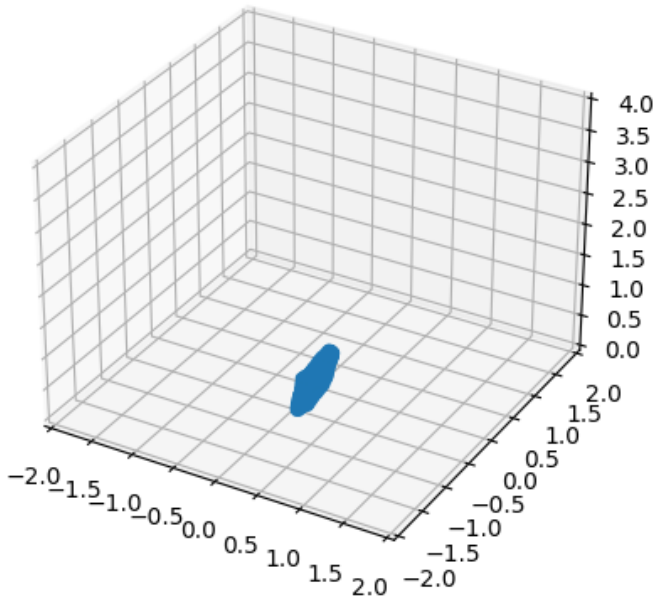
Source pcd at epoch 100



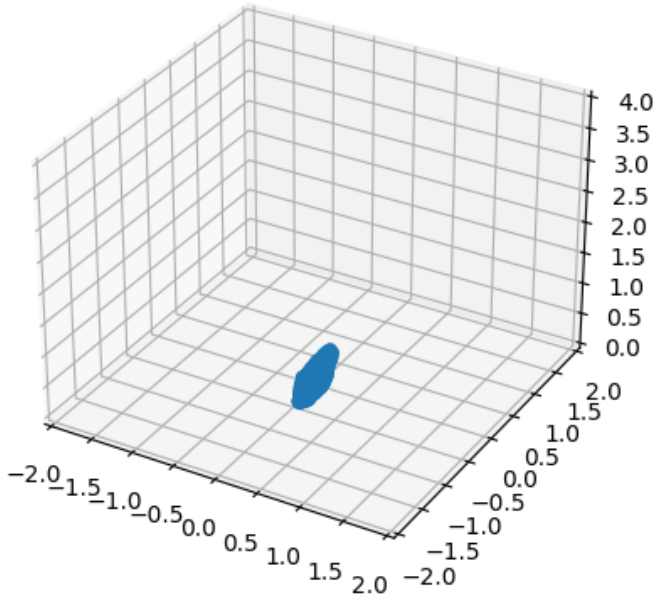
Source pcd at epoch 200



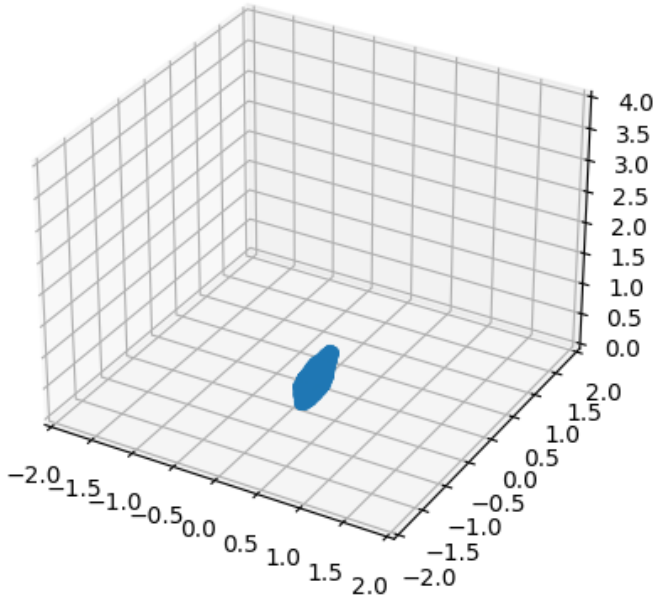
Source pcd at epoch 300



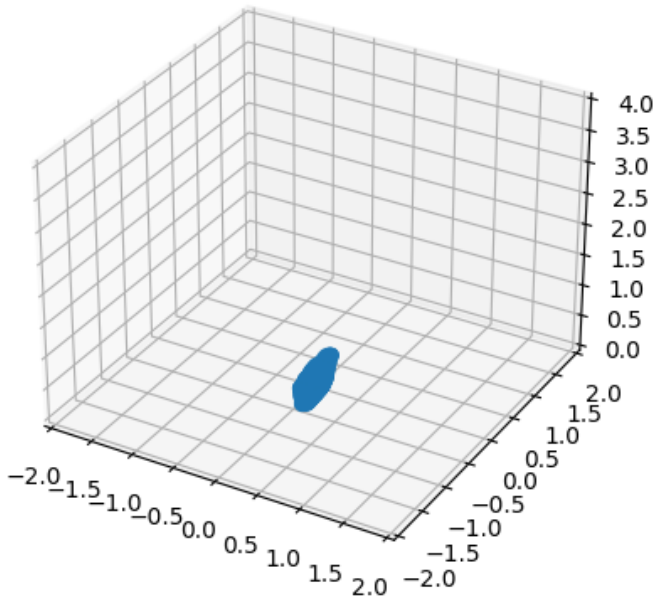
Source pcd at epoch 400



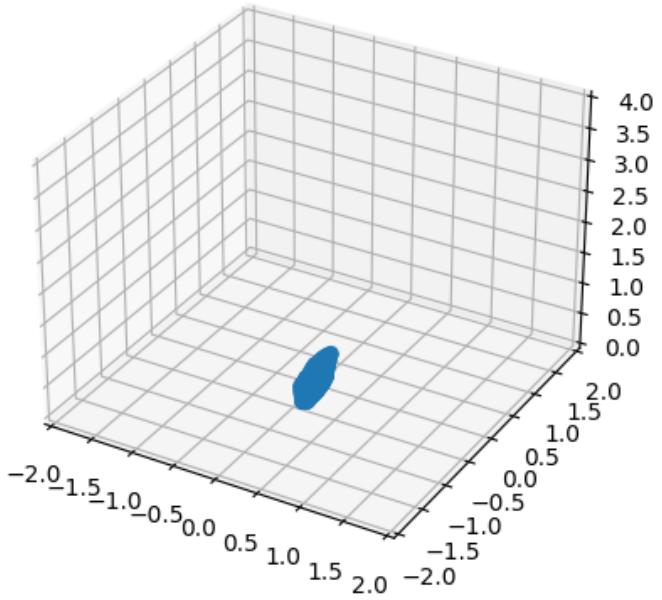
Source pcd at epoch 500



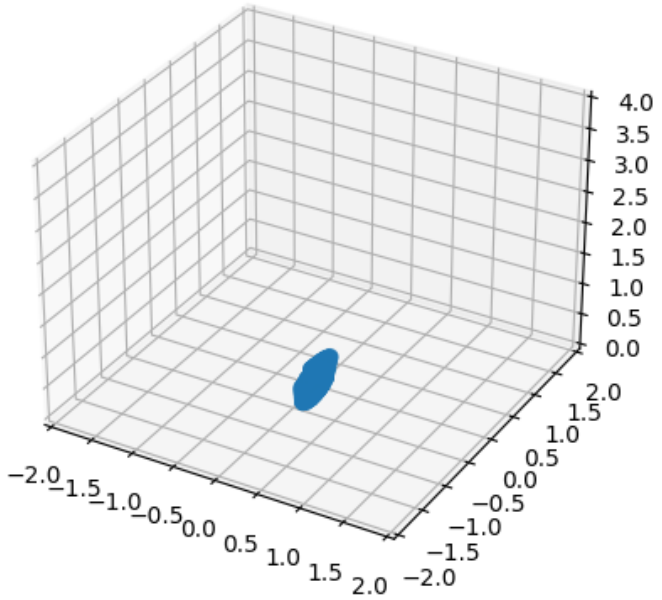
Source pcd at epoch 600



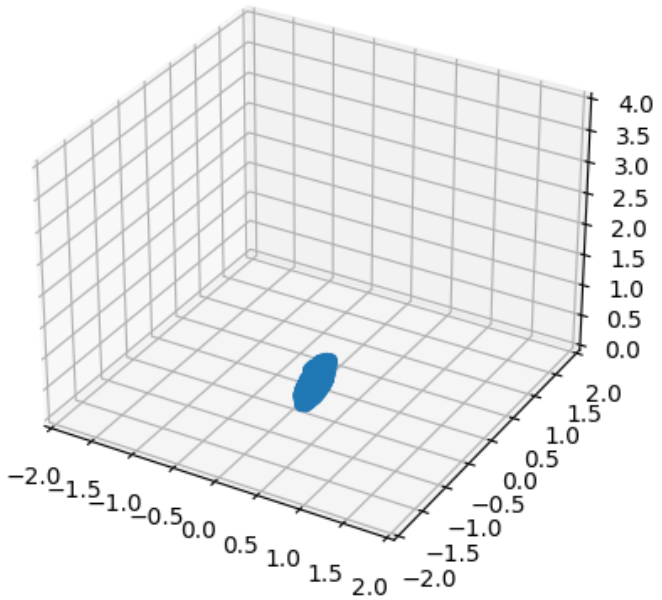
Source pcd at epoch 700

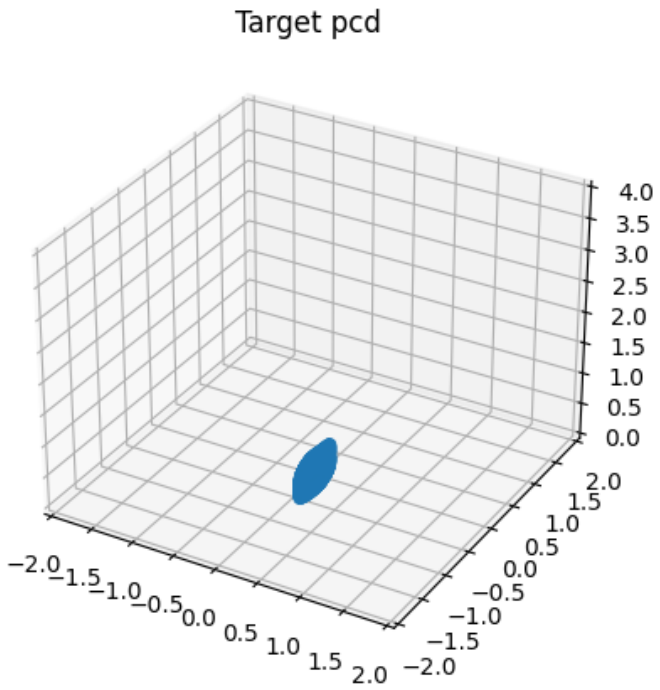


Source pcd at epoch 800

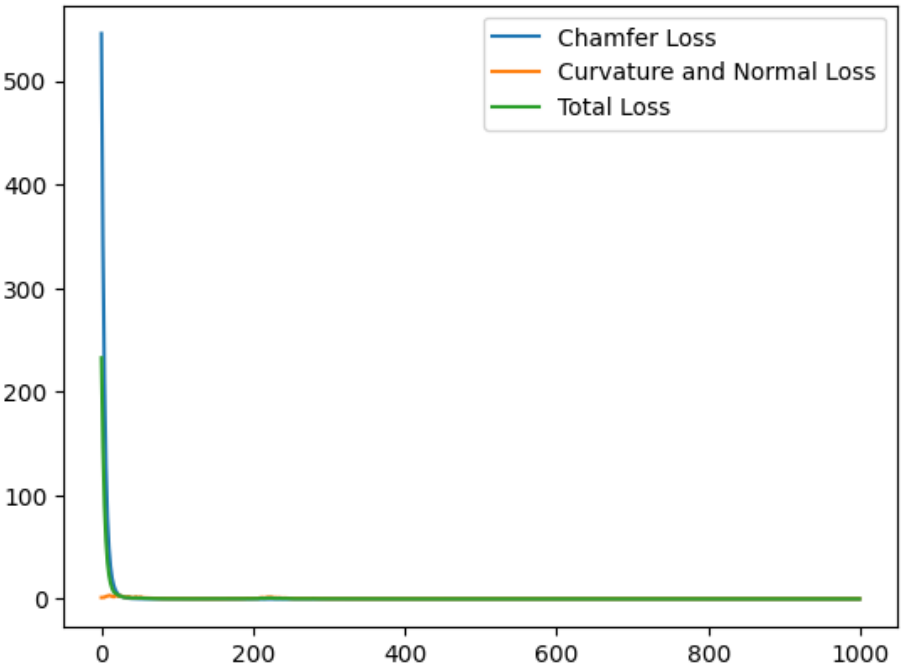


Source pcd at epoch 900





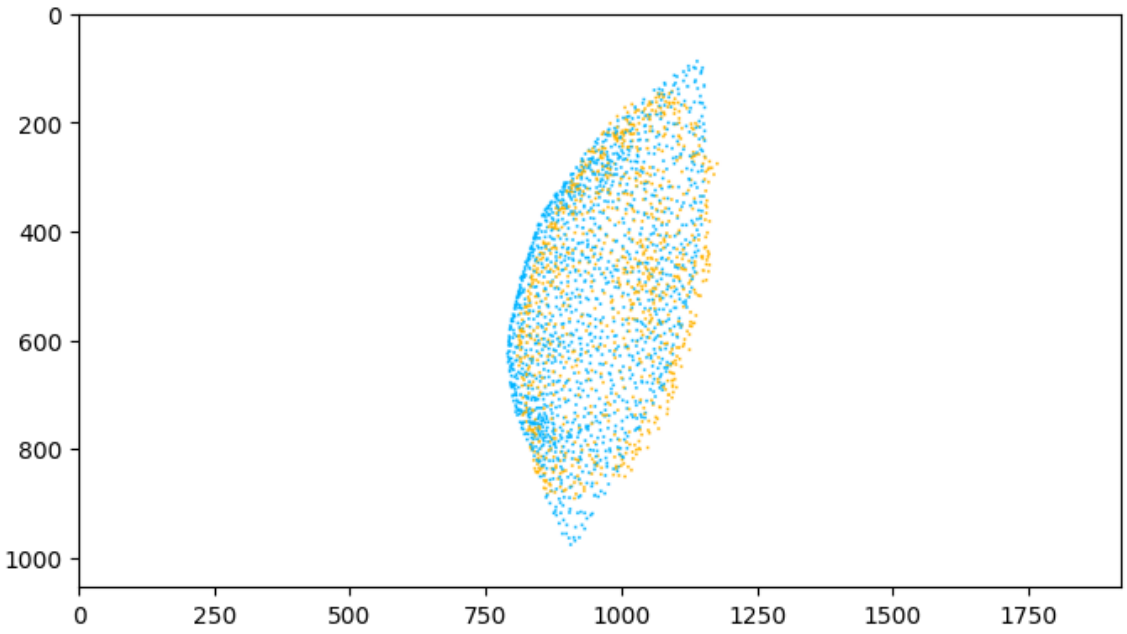
In [ ]:
```
# plot Loss lines
x = list(range(len(curv_losses)))
plt.plot(x, cf_losses, label = "Chamfer Loss")
plt.plot(x, curv_losses, label = "Curvature and Normal Loss")
plt.plot(x, losses, label = "Total Loss")
plt.legend()
plt.show()
```



In [ ]:
```
import open3d

vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1e-4, 1))
    view_ctl.set_front((0, 0.5, 2))
    view_ctl.set_lookat((0, 0, 0))
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(8,6))
```

```
    plt.imshow(np.asarray(img)[::-1, ::-1])
    for g in geoms:
        vis.remove_geometry(g)


pcd = open3d.geometry.PointCloud()
pcd1 = open3d.geometry.PointCloud()
src = vertSrcTen.squeeze(0).detach().numpy()
tg = vertTgTen.squeeze(0).detach().numpy()
print(src.shape)
pcd.points = open3d.utility.Vector3dVector(src)
pcd.paint_uniform_color([1, 0.706, 0])
# print(np.asarray(pcd.colors))
pcd1.points = open3d.utility.Vector3dVector(tg)
pcd1.paint_uniform_color([0, 0.706, 1])

draw_geometries([pcd, pcd1])
```

```
(962, 3)
```



Problem 2: ICP

In [ ]:
```
"""Visualization utilies."""

# You can use other visualization from previous homeworks, like Open3D
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def show_points(points, title):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_title(title)
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points[:, 0], points[:, 2], points[:, 1])

def compare_points(points1, points2):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points1[:, 0], points1[:, 2], points1[:, 1])
    ax.scatter(points2[:, 0], points2[:, 2], points2[:, 1])
```

In [ ]:
```
"""Load data."""

import trimesh
```

```
import numpy as np

source_pcd = trimesh.load("banana.source.ply").vertices
target_pcd = trimesh.load("banana.target.ply").vertices
gt_T = np.loadtxt("banana.pose.txt")
```

In [ ]:

```
"""Implement your own ICP."""
from tqdm import tqdm
import numpy as np
from sklearn.neighbors import NearestNeighbors

def oneNN(source_pcd, target_pcd):
    '''
    Map source_pcd to nearest neighbour in target_pcd
    Outputs:
        dists: Euclidean distances of the nearest neighbor
        inds: target_pcd indices of the nearest neighbor
    '''
    neigh = NearestNeighbors(n_neighbors=1)
    neigh.fit(target_pcd)
    dists, inds = neigh.kneighbors(source_pcd, return_distance=True)
    return dists.ravel(), inds.ravel()



def getRT(source_pcd, target_pcd):
    """
    Get R, t, given source pcd and aligned target pcd
    """
    # Get R, t (by Umeyama's minimization of distance)
    N = source_pcd.shape[0]
    qbar = np.mean(target_pcd, axis=0)
    pbar = np.mean(source_pcd, axis=0)
    M = (target_pcd-qbar).T @ (source_pcd-pbar)
    U, D, VT = np.linalg.svd(M)

    R = U @ VT
    if (np.linalg.det(R)<0):
        VT[-1,:] *= -1
        R = U @ VT

    t = qbar - (R @ pbar)
    return R, t

def icp(source_pcd, target_pcd):
    """Iterative closest point.

    Args:
        source_pcd (np.ndarray): [N1, 3]
        target_pcd (np.ndarray): [N2, 3]

    Returns:
        np.ndarray: [4, 4] rigid transformation to align source to target.
    """
    T = np.eye(4)
    R, t = np.eye(3), np.zeros(3)
    N = source_pcd.shape[0]
    source_pcd_cp = source_pcd.copy()

    # Implement your own algorithm here.
    nsteps = 300
    for i in tqdm(range(nsteps)):
        # Update correspondance (by re-aligning target)
        distances, indices = oneNN(source_pcd, target_pcd)
        # Umayama's algo for getting R, t
        R, t = getRT(source_pcd, target_pcd[indices, :])
        source_pcd = source_pcd @ R.T + t

    R, t = getRT(source_pcd_cp, source_pcd)
    T[:3, :3], T[:3, -1] = R, t
    # print("Final T:")
    # print(T)
    return T
```

In [ ]:

```python
"""Metric and visualization."""

def compute_rre(R_est: np.ndarray, R_gt: np.ndarray):
    """Compute the relative rotation error (geodesic distance of rotation)."""
    assert R_est.shape == (3, 3), 'R_est: expected shape (3, 3), received shape {}.'.format(R_est.shape)
    assert R_gt.shape == (3, 3), 'R_gt: expected shape (3, 3), received shape {}.'.format(R_gt.shape)
    # relative rotation error (RRE)
    # Rotational degree loss (not objective of optimization)
    rre = np.arccos(np.clip(0.5 * (np.trace(R_est.T @ R_gt) - 1), -1.0, 1.0))
    return rre


def compute_rte(t_est: np.ndarray, t_gt: np.ndarray):
    assert t_est.shape == (3,), 't_est: expected shape (3,), received shape {}.'.format(t_est.shape)
    assert t_gt.shape == (3,), 't_gt: expected shape (3,), received shape {}.'.format(t_gt.shape)
    # relative translation error (RTE)
    rte = np.linalg.norm(t_est - t_gt) # Resembling MSE loss
    return rte


# Visualization
T = icp(source_pcd, target_pcd)
# T = np.eye(4)
print("GT T:")
print(gt_T)
print("pred T:")
print(T)
rre = np.rad2deg(compute_rre(T[:3, :3], gt_T[:3, :3]))
rte = compute_rte(T[:3, 3], gt_T[:3, 3])
print(f"rre={rre}, rte={rte}")
compare_points(source_pcd @ T[:3, :3].T + T[:3, 3], target_pcd)
```

```
100%|██████████| 300/300 [00:16<00:00, 18.02it/s]
GT T:
[[ 0.04139069 -0.12505186  0.99128646  1.14856815]
 [-0.15543338  0.9792519   0.13002374  1.55152014]
 [-0.98697886 -0.15946078  0.02109468  0.44714717]
 [ 0.          0.          0.          1.        ]]
pred T:
[[-0.03226957 -0.50271669 -0.86384872  1.09361539]
 [ 0.13146246  0.854655   -0.50227726  1.53642465]
 [ 0.99079579 -0.12977195  0.03850902  0.48842529]
 [ 0.          0.          0.          1.        ]]
rre=179.9503708905986, rte=0.07036734069765385
```