hw2_Q1

November 17, 2022

1 Problem 1.3 (Chamfer, Curvature & Normal Loss)

[1]:
```
import sys
import time
import numpy as np
from tqdm.notebook import tqdm

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

import torch
from torch import nn
from torch.functional import F

import trimesh
print("Trimesh version:", trimesh.__version__)
```

```
Trimesh version: 3.16.2
```

[2]:
```
"""Visualization utilies."""
def show_points(points):
    fig = plt.figure(figsize=(14,8))
    ax = fig.gca(projection='3d')
    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(figsize=(14,8))
    ax = fig.gca(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])
```

[3]:
```
# define device type - cuda:0 or cpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

1.0.1 Load data

[4]:
```
"""Load data."""
source_pcd = trimesh.load("../data/source.obj")
target_pcd = trimesh.load("../data/target.obj")
print("Source PCD:", source_pcd)
print("Target PCD:", target_pcd)

source_vertices = source_pcd.vertices
source_faces = source_pcd.faces
target_vertices = target_pcd.vertices
target_faces = target_pcd.faces

num_src_pts = source_vertices.shape[0]
num_tgt_pts = target_vertices.shape[0]
num_src_faces = source_faces.shape[0]
num_tgt_faces = target_faces.shape[0]
```

```
Source PCD: <trimesh.Trimesh(vertices.shape=(962, 3), faces.shape=(1920, 3))>
Target PCD: <trimesh.Trimesh(vertices.shape=(1502, 3), faces.shape=(3000, 3))>
```

1.0.2 Convert numpy data to torch tensor

[5]:
```
src_vtx_ten = torch.clone(torch.tensor(source_vertices)).to(device)
src_faces_ten = torch.clone(torch.tensor(source_faces)).to(device)
tgt_vtx_ten = torch.clone(torch.tensor(target_vertices)).to(device)
tgt_faces_ten = torch.clone(torch.tensor(target_faces)).to(device)
```

[6]:
```
def plot_mesh_3d(vertices, faces, plt_title):
    # Creating figure
    fig = plt.figure(figsize=(14, 9))
    ax = plt.axes(projection ='3d')

    # Creating color map
    my_cmap = cm.get_cmap('rainbow')

    surf_plot = ax.plot_trisurf(vertices[:,0], vertices[:,1], vertices[:,2],
                                triangles=faces,
                                cmap=my_cmap,
                                linewidth = 0.1,
                                antialiased = True)
```

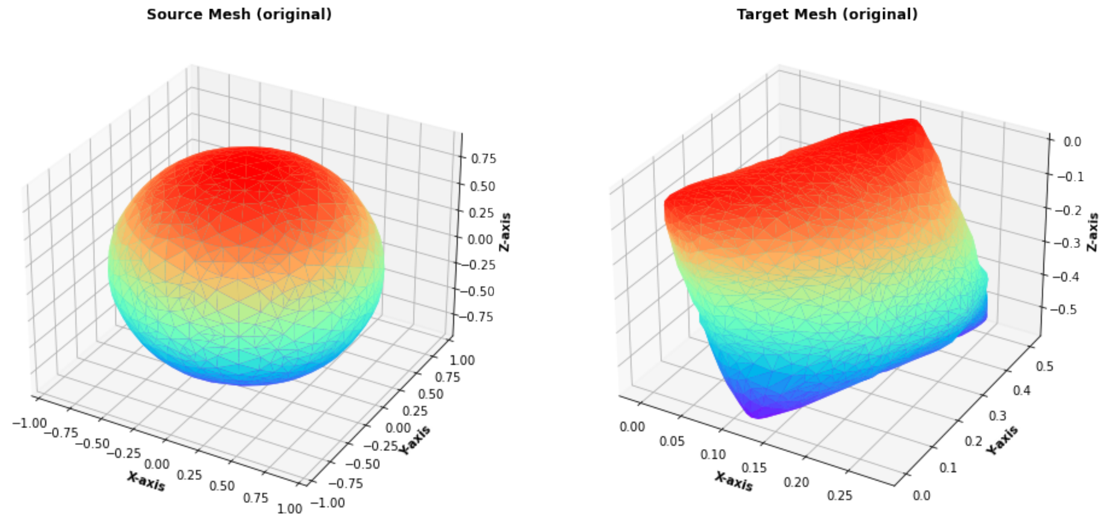
```python
#     fig.colorbar(surf_plot, shrink=0.5, aspect=10)
    ax.set_title(plt_title, fontweight ='bold')
    ax.set_xlabel('X-axis', fontweight ='bold')
    ax.set_ylabel('Y-axis', fontweight ='bold')
    ax.set_zlabel('Z-axis', fontweight ='bold')
    plt.show()
```

[7]:
```python
def plot_meshes_3d(vertices1, faces1, vertices2, faces2, plt_title1,␣
 ↪plt_title2):
    # Creating figure
    fig = plt.figure(figsize=(16, 10))
    # Creating color map
    my_cmap = cm.get_cmap('rainbow')

    ax = fig.add_subplot(1, 2, 1, projection='3d')
    surf_plot = ax.plot_trisurf(vertices1[:,0], vertices1[:,1], vertices1[:,2],
                                triangles=faces1,
                                cmap=my_cmap,
                                linewidth = 0.1,
                                antialiased = True)
    ax.set_title(plt_title1, fontweight ='bold')
    ax.set_xlabel('X-axis', fontweight ='bold')
    ax.set_ylabel('Y-axis', fontweight ='bold')
    ax.set_zlabel('Z-axis', fontweight ='bold')

    ax = fig.add_subplot(1, 2, 2, projection='3d')
    surf_plot = ax.plot_trisurf(vertices2[:,0], vertices2[:,1], vertices2[:,2],
                                triangles=faces2,
                                cmap=my_cmap,
                                linewidth = 0.1,
                                antialiased = True)
    ax.set_title(plt_title2, fontweight ='bold')
    ax.set_xlabel('X-axis', fontweight ='bold')
    ax.set_ylabel('Y-axis', fontweight ='bold')
    ax.set_zlabel('Z-axis', fontweight ='bold')
#     fig.colorbar(surf_plot, shrink=0.5, aspect=10)

    plt.show()
```

1.0.3 Display source and target point clouds

[8]:
```python
plot_meshes_3d(source_vertices,
               source_faces,
               target_vertices,
               target_faces,
               "Source Mesh (original)",
```

```
                "Target Mesh (original)")
```



1.0.4 Function to compute adjacency matrix for any mesh

[9]:
```python
def get_adjacency_mat(faces, num_pts):
    num_faces = faces.shape[0]
    adj_mat = torch.zeros((num_pts, num_pts))

    for i in range(num_faces):
        vertex_lst = faces[i]
        adj_mat[vertex_lst[0], vertex_lst[1]] = 1
        adj_mat[vertex_lst[1], vertex_lst[0]] = 1

        adj_mat[vertex_lst[0], vertex_lst[2]] = 1
        adj_mat[vertex_lst[2], vertex_lst[0]] = 1

        adj_mat[vertex_lst[1], vertex_lst[2]] = 1
        adj_mat[vertex_lst[2], vertex_lst[1]] = 1

    return adj_mat
```

[10]:
```python
def get_Lnorm_mat(A_mat):
    num_pts = A_mat.shape[0]
    D_mat = torch.diag(torch.sum(A_mat, axis=1))
    L_mat = D_mat - A_mat
    D_inv = torch.linalg.inv(D_mat)
    Lnorm_mat = torch.eye(num_pts) - torch.matmul(D_inv, A_mat)

    return Lnorm_mat
```

[11]:
```python
# adjacency matrix not changing for src and tgt pcd. so just one time␣
↪computation
src_adj_mat = get_adjacency_mat(src_faces_ten, num_src_pts)
tgt_adj_mat = get_adjacency_mat(tgt_faces_ten, num_tgt_pts)
print(src_adj_mat.shape, tgt_adj_mat.shape)

# Lnorm matrix not changing for src and tgt pcd. so just one time computation
src_Lnorm_mat = get_Lnorm_mat(src_adj_mat)
tgt_Lnorm_mat = get_Lnorm_mat(tgt_adj_mat)
print(src_Lnorm_mat.shape, tgt_Lnorm_mat.shape)

src_Lnorm_mat = src_Lnorm_mat.type(torch.DoubleTensor).to(device)
tgt_Lnorm_mat = tgt_Lnorm_mat.type(torch.DoubleTensor).to(device)

# deltaP not changing for tgt pcd. so just one time computation
src_deltaP = torch.matmul(src_Lnorm_mat, src_vtx_ten)
tgt_deltaP = torch.matmul(tgt_Lnorm_mat, tgt_vtx_ten)
```

```
torch.Size([962, 962]) torch.Size([1502, 1502])
torch.Size([962, 962]) torch.Size([1502, 1502])
```

1.0.5 Function to compute chamfer loss

[12]:
```python
# 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
```

[13]:
```python
# 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(xyz1, xyz2, data_format='NWC'):
    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 dist1, idx1, dist2, idx2
```

[14]:
```
def compute_chamfer_loss(src_vertices, tgt_vertices):
    dist1, idx1, dist2, idx2 = Chamfer_distance_torch(src_vertices.
↪unsqueeze(0), tgt_vertices.unsqueeze(0))
#     print(dist1.shape, idx1.shape, dist2.shape, idx2.shape)
    chamfer_loss = torch.sum(torch.square(dist1)) + torch.sum(torch.
↪square(dist2))

    return chamfer_loss
```

[15]:
```
# original chamfer loss written by me

# def compute_chamfer_loss(src_vertices, tgt_vertices):
#     num_src_pts = src_vertices.shape[0]
#     num_tgt_pts = tgt_vertices.shape[0]

#     chamfer_loss = 0

#     for i in range(num_src_pts):
#         dists = torch.square(torch.linalg.norm(tgt_vertices -␣
↪src_vertices[i], axis=1))
#         chamfer_loss += torch.min(dists)

#     for j in range(num_tgt_pts):
#         dists = torch.square(torch.linalg.norm(src_vertices -␣
↪tgt_vertices[j], axis=1))
#         chamfer_loss += torch.min(dists)

#     return chamfer_loss
```

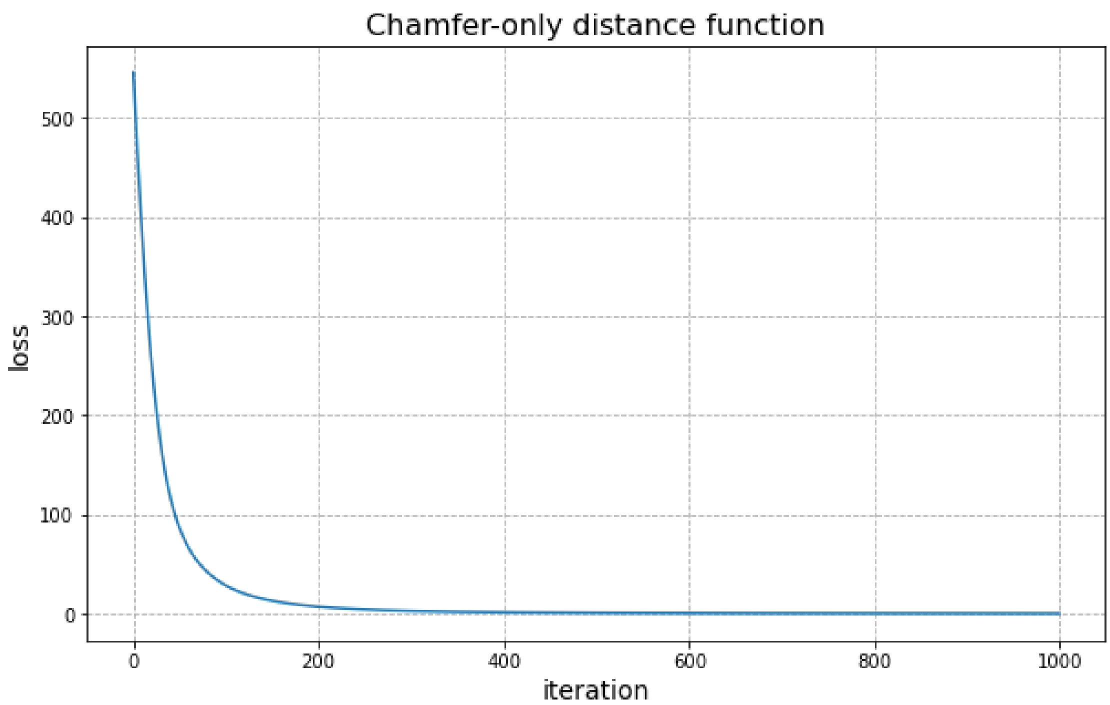
1.0.6 Below code iterates and deforms the source vertices towards target mesh using Chamfer Loss

[16]:
```
chamfer_losses = []
num_itr = 1000

weights_chamfer = nn.Parameter(torch.clone(src_vtx_ten)).to(device)

# Instantiate optimizer
optimizer = torch.optim.Adam([weights_chamfer], lr=0.005)

for _ in tqdm(range(num_itr)):
    loss = compute_chamfer_loss(weights_chamfer, tgt_vtx_ten)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    chamfer_losses.append(loss.item())

deformed_chamfer_vertices = weights_chamfer.cpu().detach().numpy()
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

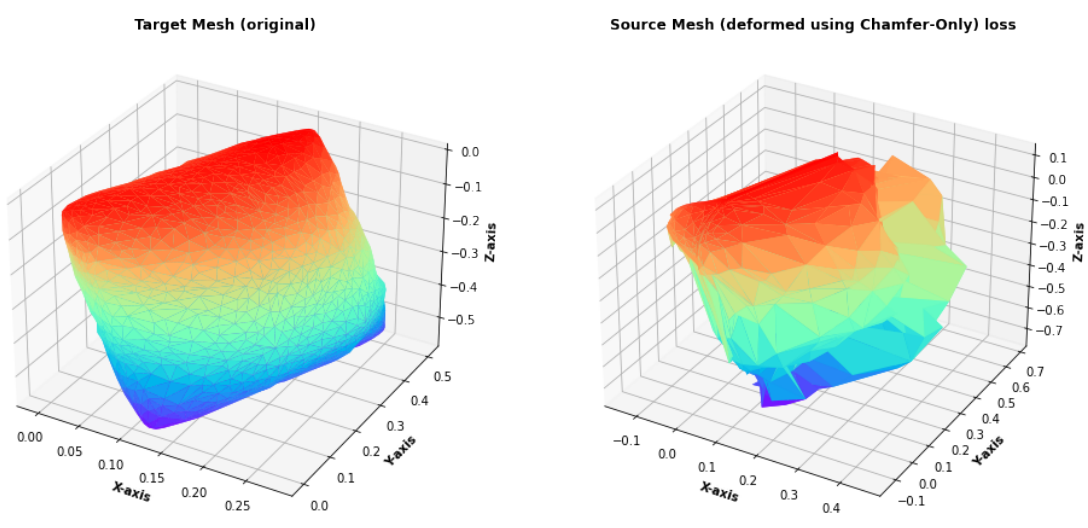
1.0.7 Plot the variation of chamfer distance function with optimization iteration

[17]:
```
fig = plt.figure(figsize=(10, 6))
plt.plot(chamfer_losses)
plt.grid(linestyle='--')
plt.title("Chamfer-only distance function", fontsize=16)
plt.xlabel("iteration", fontsize=14)
plt.ylabel("loss", fontsize=14)
plt.show()
```



1.0.8 Result: Comparison between Target Mesh and Deformed Mesh using Chamfer Loss

[18]:
```
plot_meshes_3d(target_vertices,
               target_faces,
               deformed_chamfer_vertices,
               source_faces,
               "Target Mesh (original)",
               "Source Mesh (deformed using Chamfer-Only) loss")
```



1.0.9 Describe what has happened in the deformation process by language

- The source mesh is spherical (slightly elliptical) in shape, whereas the target mesh is rectangular at the boundaries but slighly spherical at the middle.
- The chamfer loss measures the sum of the squared distances of each point in the source set to its nearest neighbour in the target set and vice-versa. The objective function in the part (a) optimization problem is the chamfer loss itself whereas the vertices of the source mesh are the optimization variables.
- The adam optimizer optimizes the vertices to minimize the chamfer distance loss. Therefore, for each point in source mesh, the distance to nearest point in the target mesh is minimized which results in source vertices moving more closer to their corresponding nearest neighbors in target mesh. The loss value saturates near 0 after which we do not see any significant improvement.
- The result using only chamfer loss is far from perfect. But, we can see that the mesh is being deformed in the direction of target mesh shape.

1.0.10 Function to compute Curvature + Chamfer loss

[19]:
```python
def compute_curvature_loss(src_vertices, tgt_vertices):
    dist1, idx1, dist2, idx2 = Chamfer_distance_torch(src_vertices.
 ↪unsqueeze(0), tgt_vertices.unsqueeze(0))
    chamfer_loss = torch.sum(torch.square(dist1)) + torch.sum(torch.
 ↪square(dist2))

    # compute deltaP for src vertices / weight matrix
    src_deltaP = torch.matmul(src_Lnorm_mat, src_vertices)

    # curvature + normal loss
    deltaP_src = torch.sum(torch.square(torch.linalg.norm(src_deltaP -␣
 ↪tgt_deltaP[idx1], axis=1)))
    deltaP_tgt = torch.sum(torch.square(torch.linalg.norm(tgt_deltaP -␣
 ↪src_deltaP[idx2], axis=1)))

    return chamfer_loss + deltaP_src + deltaP_tgt
```

[20]:
```python
# code written by me

# def compute_curvature_loss(src_vertices, tgt_vertices):
#     num_src_pts = src_vertices.shape[0]
#     num_tgt_pts = tgt_vertices.shape[0]

#     # compute deltaP for src vertices / weight matrix
#     src_deltaP = torch.matmul(src_Lnorm_mat, src_vertices)
```

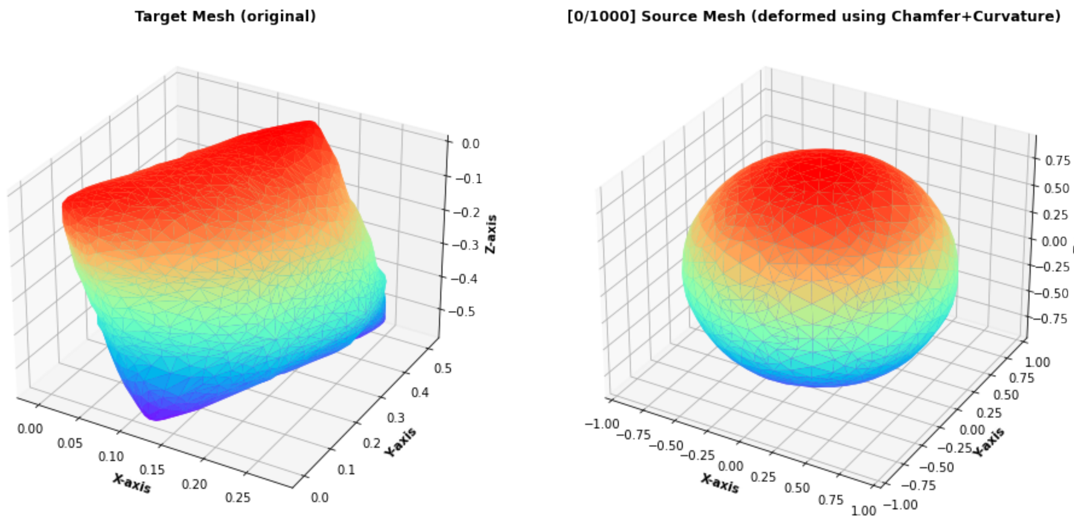
```python
#     chamfer_loss = 0
#     min_idx_lst = []

#     for i in range(num_src_pts):
#         dists = torch.square(torch.linalg.norm(tgt_vertices -
 ↪src_vertices[i], axis=1))
#         min_idx_lst.append(torch.argmin(dists))

#         chamfer_loss += torch.min(dists)

#     deltaP_src = torch.sum(torch.square(torch.linalg.norm(src_deltaP -
 ↪tgt_deltaP[min_idx_lst], axis=1)))

#     min_idx_lst = []

#     for j in range(num_tgt_pts):
#         dists = torch.square(torch.linalg.norm(src_vertices -
 ↪tgt_vertices[j], axis=1))
#         min_idx_lst.append(torch.argmin(dists))

#         chamfer_loss += torch.min(dists)

#     deltaP_tgt = torch.sum(torch.square(torch.linalg.norm(tgt_deltaP -
 ↪src_deltaP[min_idx_lst], axis=1)))

#     return chamfer_loss + deltaP_src + deltaP_tgt
```

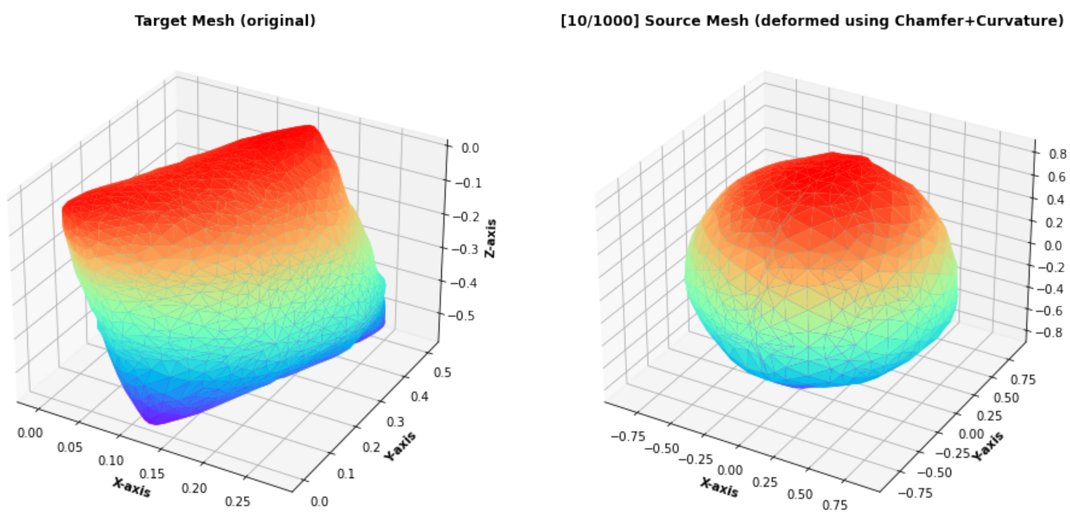
1.0.11 Below code iterates and deforms the source vertices towards target mesh using Curvature+Normal+Chamfer Loss

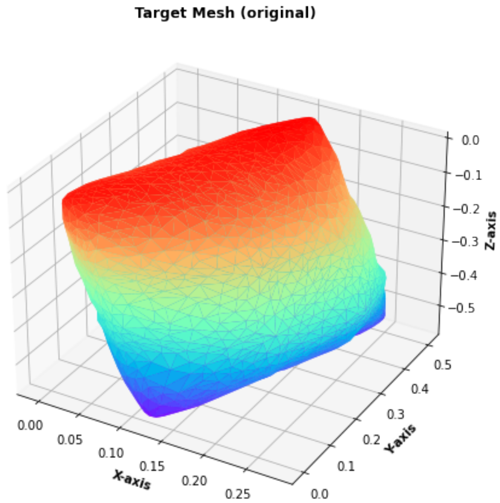
[21]:
```python
curv_losses = []
num_itr = 1000

weights_curv = nn.Parameter(torch.clone(src_vtx_ten)).to(device)

# Instantiate optimizer
optimizer = torch.optim.Adam([weights_curv], lr=0.005)

disp_itr = [0, 10, 20, 40, 50, 80]
for itr in tqdm(range(num_itr)):
    loss = compute_curvature_loss(weights_curv, tgt_vtx_ten)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    curv_losses.append(loss.item())
```

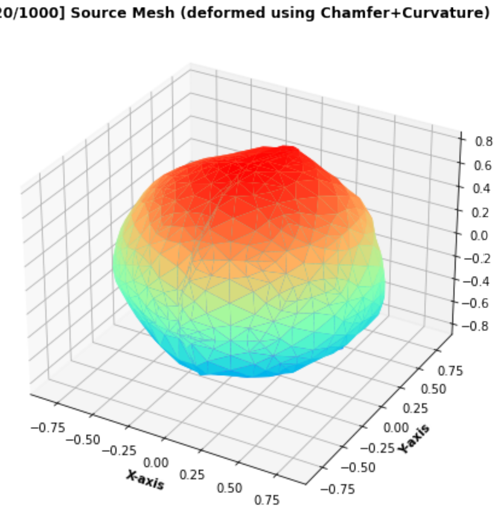
```
    if (itr % 100 == 0) or (itr in disp_itr):
        deformed_curv_vertices = weights_curv.cpu().detach().numpy()
        plot_meshes_3d(target_vertices,
                       target_faces,
                       deformed_curv_vertices,
                       source_faces,
                       "Target Mesh (original)",
                       "[{}/{}] Source Mesh (deformed using Chamfer+Curvature)".
↪format(itr, num_itr))

deformed_curv_vertices = weights_curv.cpu().detach().numpy()
```

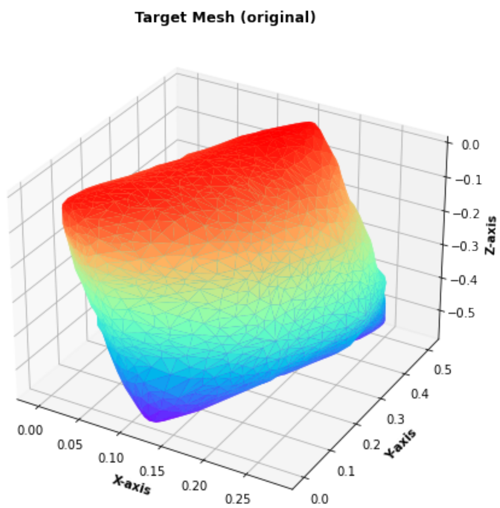
```
0%|          | 0/1000 [00:00<?, ?it/s]
```

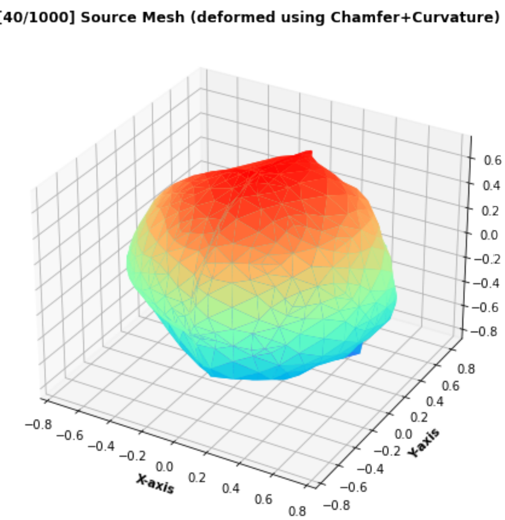


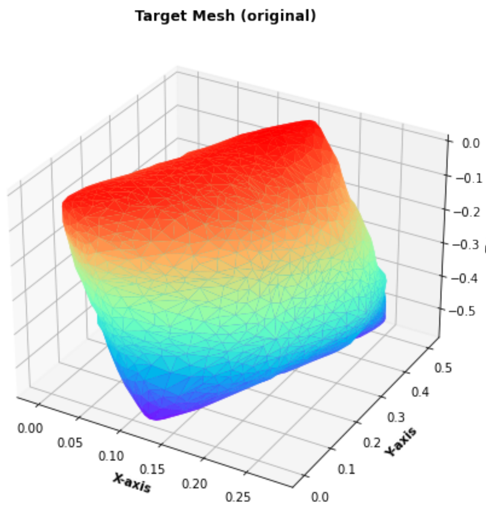


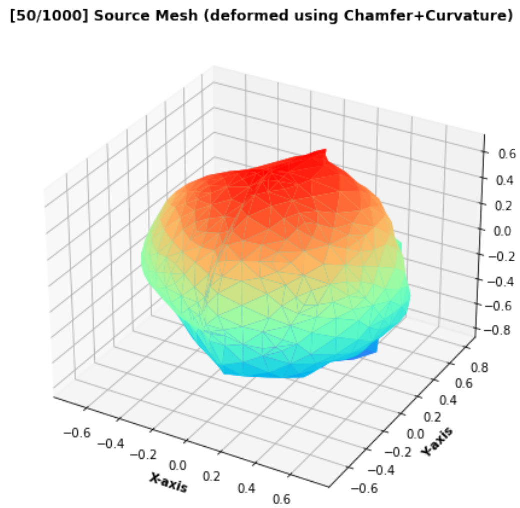

Target Mesh (original)
X-axis
Y-axis
Z-axis

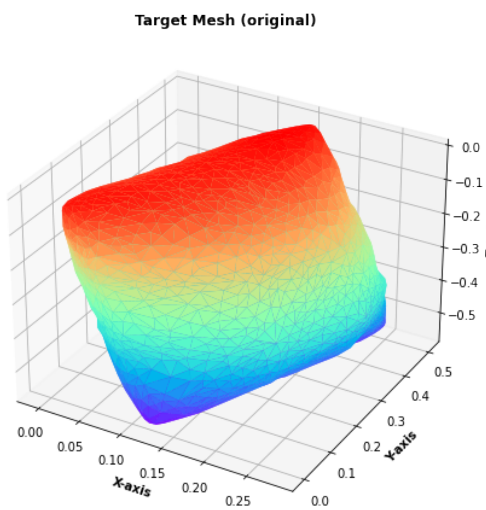

[20/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

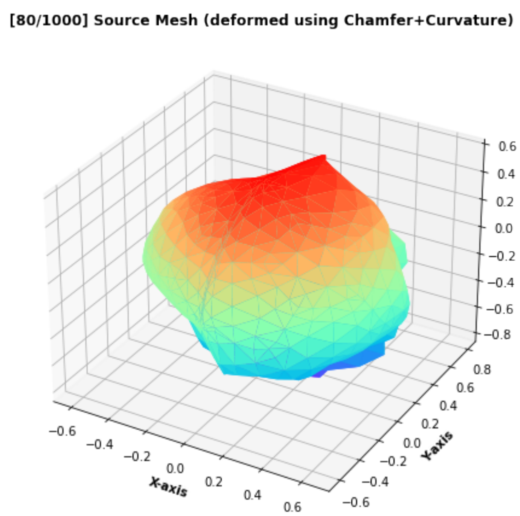

Target Mesh (original)
X-axis
Y-axis
Z-axis

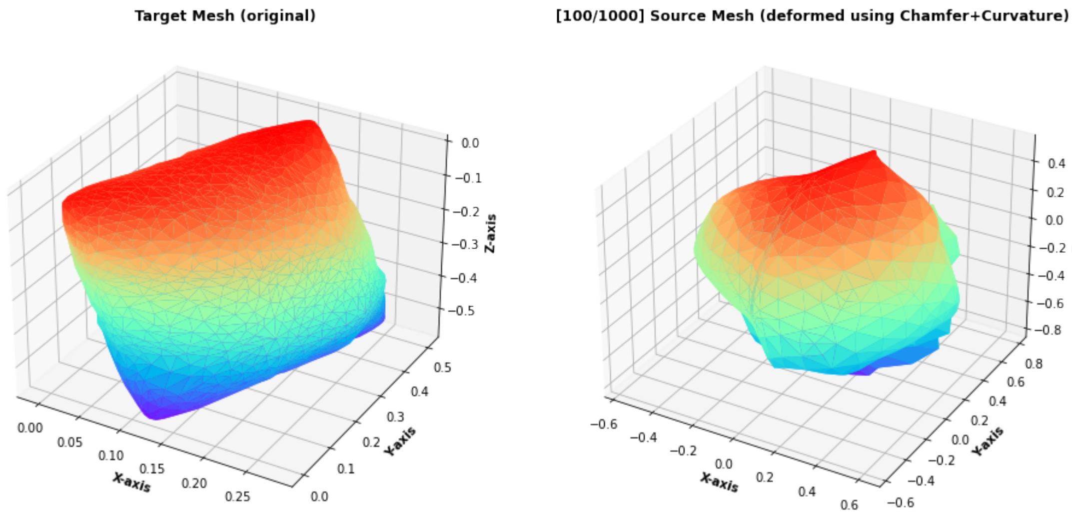

[40/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

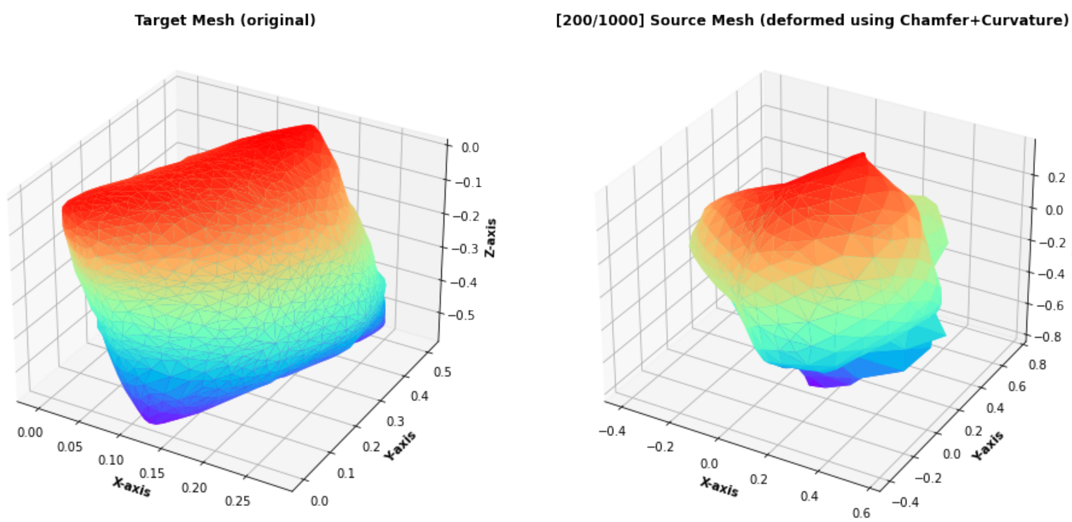

Target Mesh (original)
X-axis
Y-axis
Z-axis

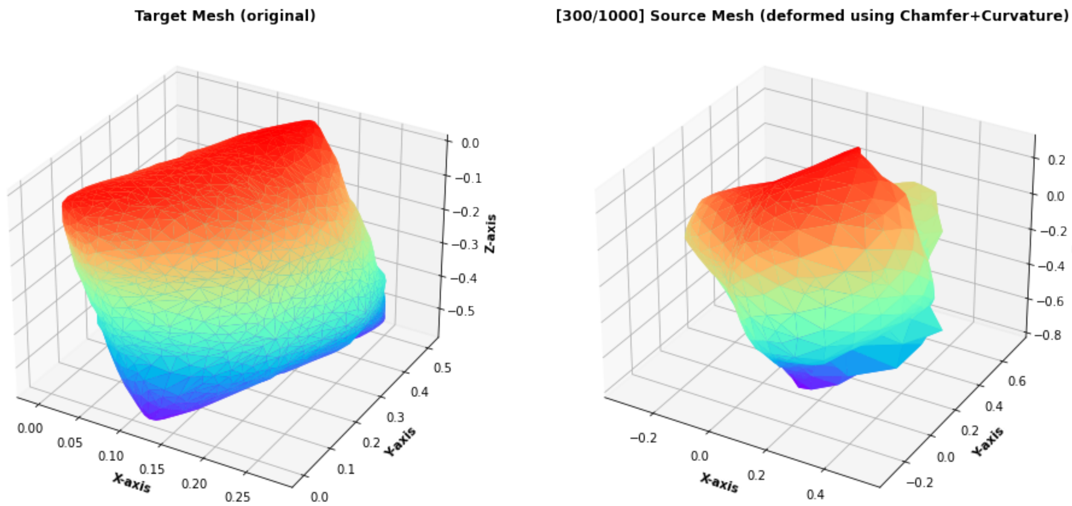

[50/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

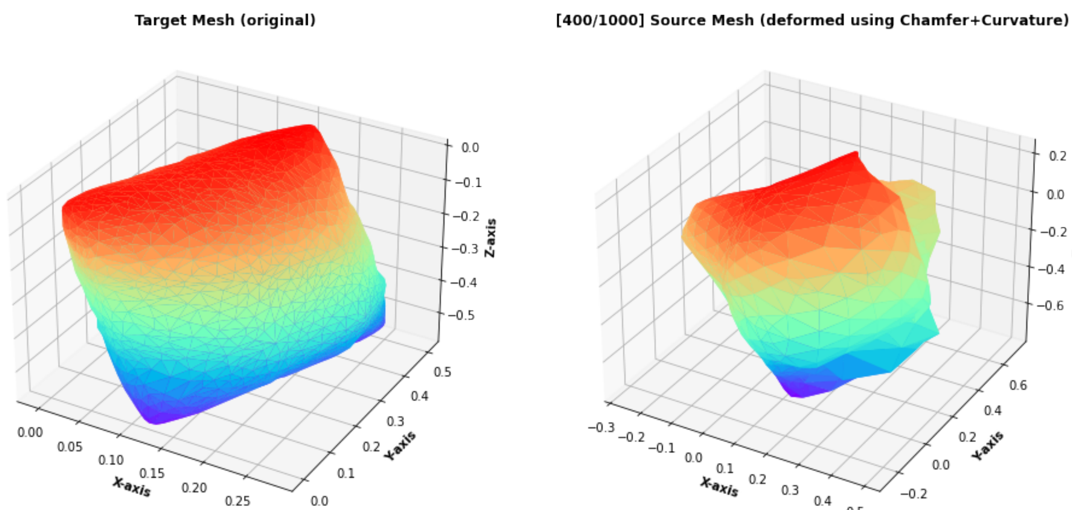

Target Mesh (original)
X-axis
Y-axis
Z-axis

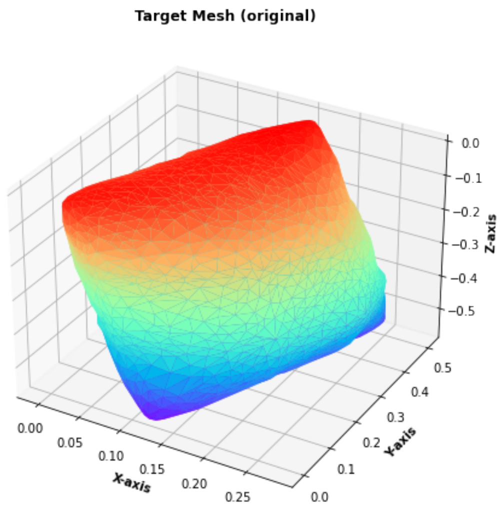

[80/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

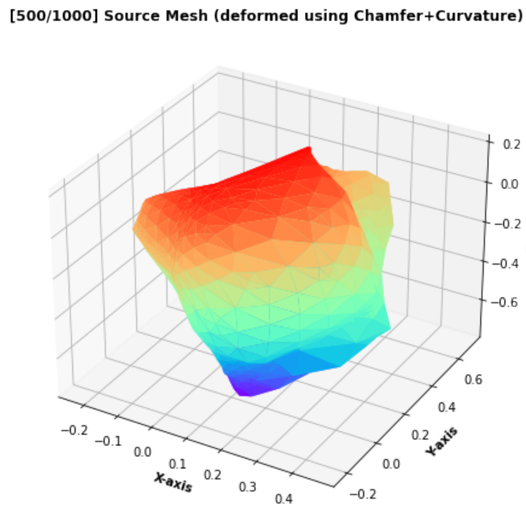

Target Mesh (original)
[100/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

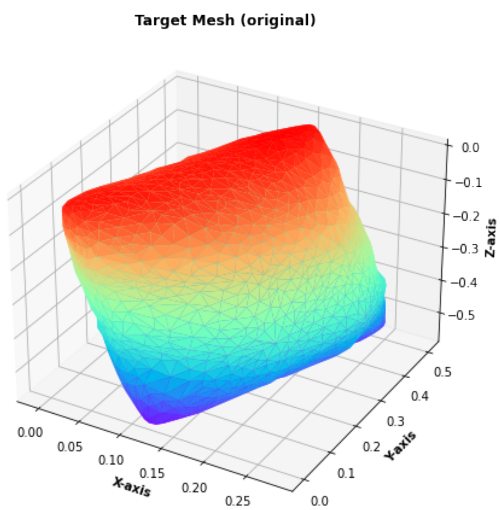

Target Mesh (original)
[200/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

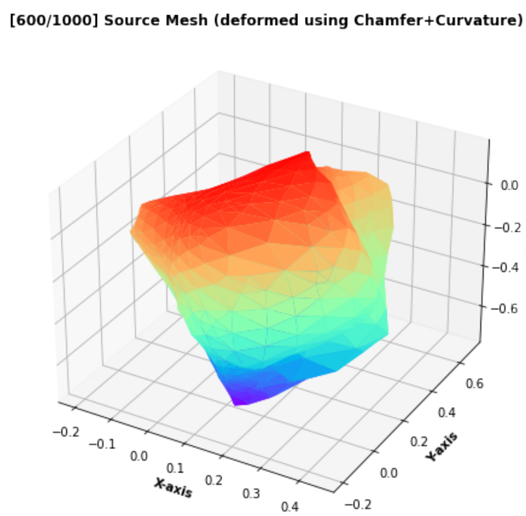

Target Mesh (original)
[300/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

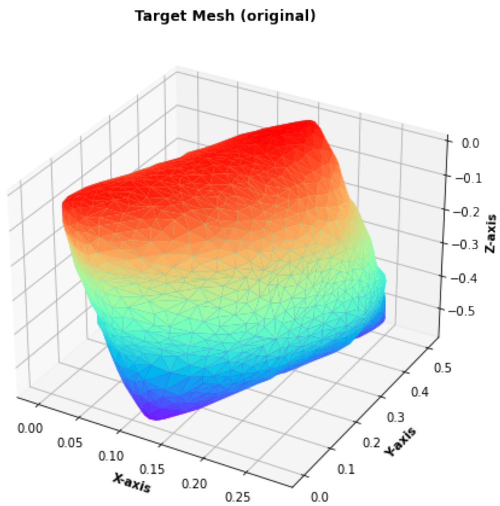

Target Mesh (original)
[400/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis

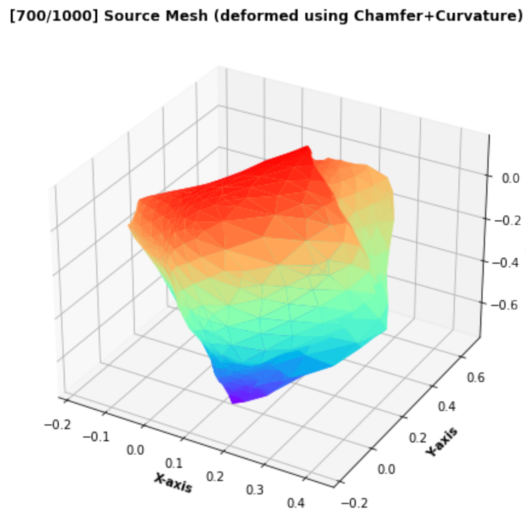

Target Mesh (original)
X-axis
0.00
0.05
0.10
0.15
0.20
0.25
Y-axis
0.0
0.1
0.2
0.3
0.4
0.5
Z-axis
0.0
−0.1
−0.2
−0.3
−0.4
−0.5

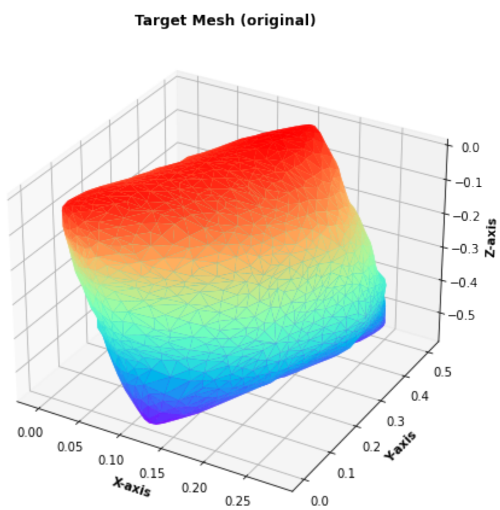

[500/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
Y-axis
−0.2
0.0
0.2
0.4
0.6
Z-axis
0.2
0.0
−0.2
−0.4
−0.6

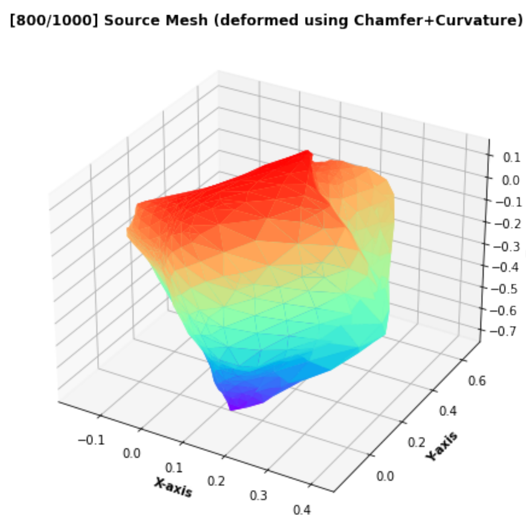

Target Mesh (original)
X-axis
0.00
0.05
0.10
0.15
0.20
0.25
Y-axis
0.0
0.1
0.2
0.3
0.4
0.5
Z-axis
0.0
−0.1
−0.2
−0.3
−0.4
−0.5

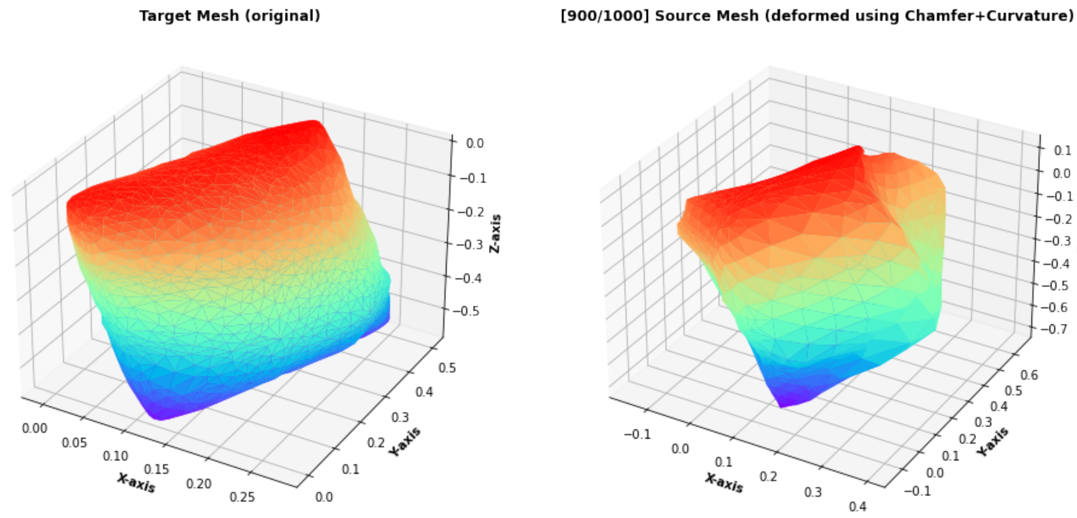

[600/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
Y-axis
−0.2
0.0
0.2
0.4
0.6
Z-axis
0.0
−0.2
−0.4
−0.6


Target Mesh (original)
X-axis
Y-axis
Z-axis

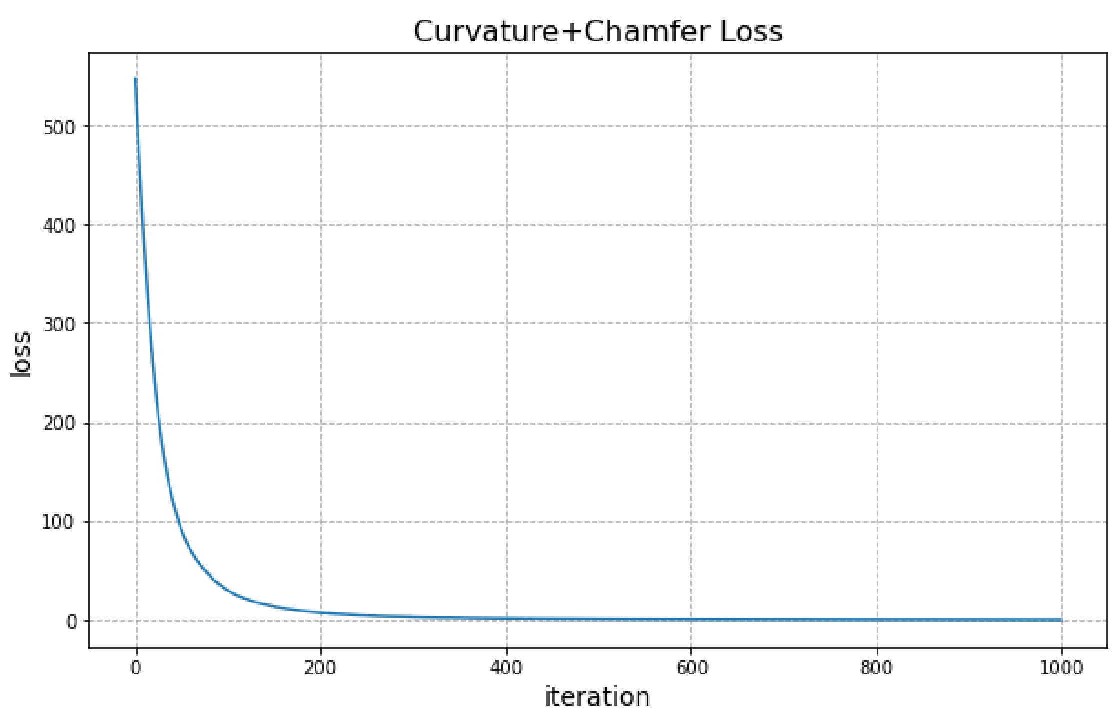

[700/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis


Target Mesh (original)
X-axis
Y-axis
Z-axis


[800/1000] Source Mesh (deformed using Chamfer+Curvature)
X-axis
Y-axis
Z-axis



[70]:
```
fig = plt.figure(figsize=(10, 6))
plt.plot(curv_losses)
plt.grid(linestyle='--')
plt.title("Curvature+Chamfer Loss", fontsize=16)
plt.xlabel("iteration", fontsize=14)
plt.ylabel("loss", fontsize=14)
plt.show()
```



1.0.12 Final Result: Comparison between Target Mesh and Deformed Mesh using Curvature+Chamfer Loss

[71]:
```
plot_meshes_3d(target_vertices,
               target_faces,
               deformed_curv_vertices,
               source_faces,
               "Target Mesh (original)",
               "Source Mesh (deformed) using Chamfer+Curvature Loss")
```