Lab 2-2

Young Min Kim

2022.07.14.

Table of Contents

1. Approximated point normals
2. Point Cloud to Mesh

1. Approximated Point Normals

- We start by computing approximate normals for the points in a point cloud, based on local tangent-plane fitting around each point.

- Note that this procedure produces unoriented normals - consistently orienting those normals typically needed to reconstruct a surface but will not be required in this exercise.

- After implementation, computed normals will be compared with ground truth normals.

1. Approximated Point Normals

- Open ‘`lab2-2/python/estimate_normals.py`’
- First, points and ground truth normals are loaded from csv files.
- Then, point cloud object is constructed using `open3d.geometry.PointCloud()`.
- Visualize ground truth point cloud using `draw_geometry()` method.

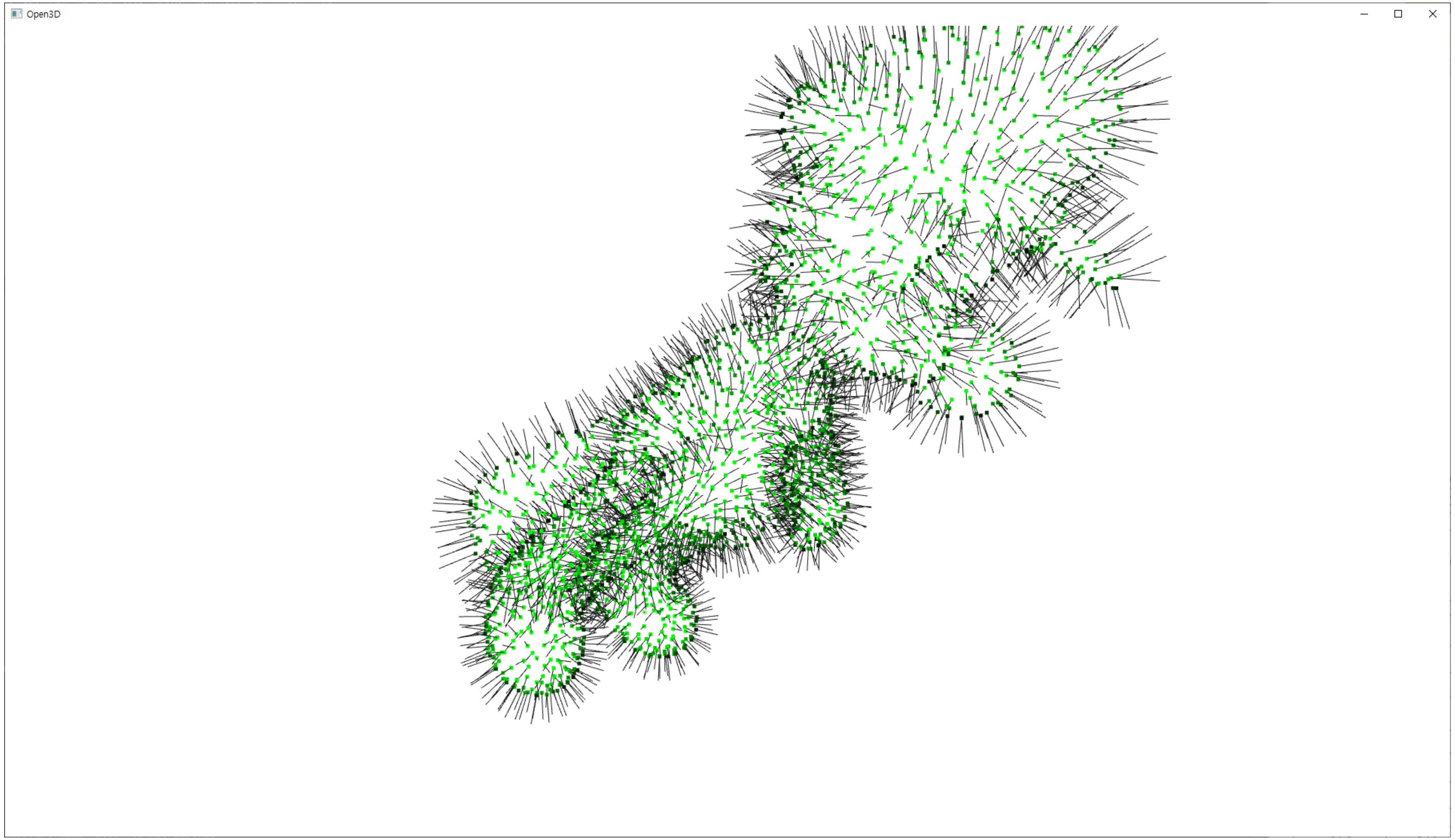
```
model = 'luigi'

# load shape information
vertices = np.loadtxt('../lab3-1_data/%s/%s_vertices.csv' % (model, model), delimiter=',', dtype=np.float32)
vertex_normals = np.loadtxt('../lab3-1_data/%s/%s_vertex_normals.csv'% (model, model), delimiter=',',
                            type=np.float32)

# construct point cloud object
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(vertices)
pcd.normals = o3d.utility.Vector3dVector(vertex_normals)
pcd.colors = o3d.utility.Vector3dVector(np.array([0, 1, 0])[None].repeat(vertices.shape[0], axis=0))

# visualization
# press n to visualize normal vectors
# press + or - to increase or decrease point size
o3d.visualization.draw_geometries([pcd])
```

1. Approximated Point Normals



Expected Output

1. Approximated Point Normals

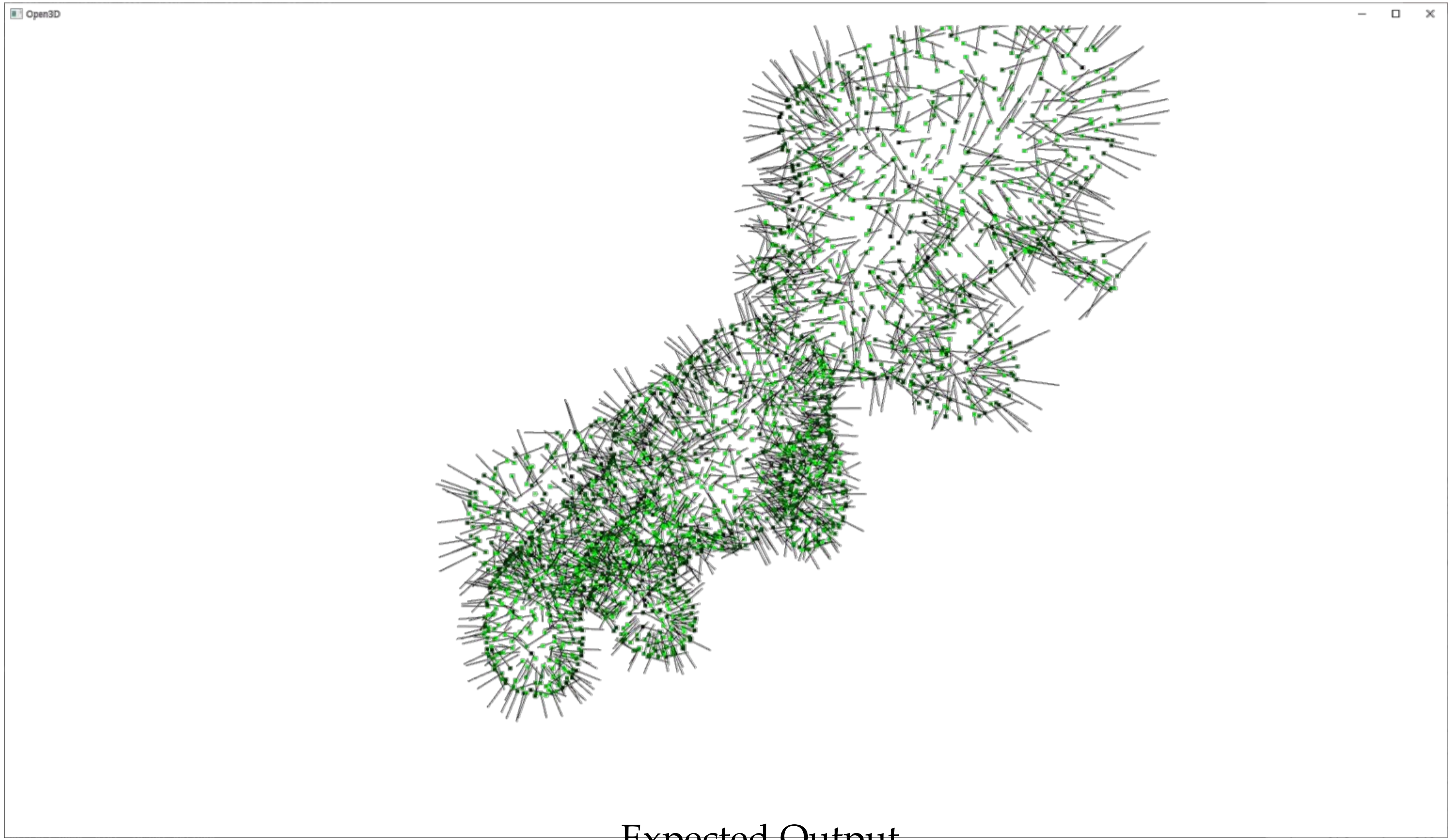
- Now, code for approximation of point normals will be implemented.
- For each point in the cloud, find its K nearest neighbors in the point cloud.
- Fit a plane to this neighborhood to compute the normal direction at the point.
- Calculated point normals will be stored in `vertex_normals_hat`.
- Use `np.linalg.eig()`.

```
# calculate vertex_normals from vertices and compare to ground truth normals

# K for K-nearest neighbor
K = 10
N, _ = vertices.shape

############################################################################################################
## TODO : calculate vertex normals from vertices
## TODO : You need to create array named vertex_normals_hat  whose shape is (N, 3) (same as 'vertex_normals')
```

1. Approximated Point Normals



Expected Output

1. Approximated Point Normals

- Answer

```
####################################################################################################
## TODO : calculate vertex normals from vertices
## TODO : You need to create array named vertex_normals_hat  whose shape is (N, 3) (same as 'vertex_normals')
vertices_self = vertices[:, None]  # (N, 1, 3)
vertices_others = vertices[None]  # (1, N, 3)
diff = vertices_self - vertices_others  # (N, N, 3)
dist = (diff ** 2).sum(axis=-1)  # (N, N)
vertex_normals_hat = []
for n in range(N):
    dist_n = dist[n].copy()
    indices = dist_n.argsort()[:K]  # ascending order, (k, )
    cov = np.matmul(diff[n, indices].T, diff[n, indices])  # (3, 3) where diff[indices, n] = (K, 3)
    eigenvalues, eigenvectors = np.linalg.eig(cov)
    vertex_normals_hat.append(eigenvectors[:, 0])
vertex_normals_hat = np.stack(vertex_normals_hat, axis=0)
####################################################################################################
```

Example Code

2. Point Cloud to Mesh

- Now, you will compute signed distance function (SDF) $f(\vec{p}), \vec{p} = (x, y, z)$ defined on all of 3D space, such that the input point cloud lies at the zero level set of this function, i.e. for any point $\overrightarrow{p_i}$ in the input point cloud, we have $f(\overrightarrow{p_i}) = 0$.

- We use ground truth normals for SDF estiation.

- Mesh will be reconstructed from calculated SDF using Marching Cubes algorithm

- Open '`lab2-2/python/point_to_mesh.py`'

2. Point Cloud to Mesh (1)

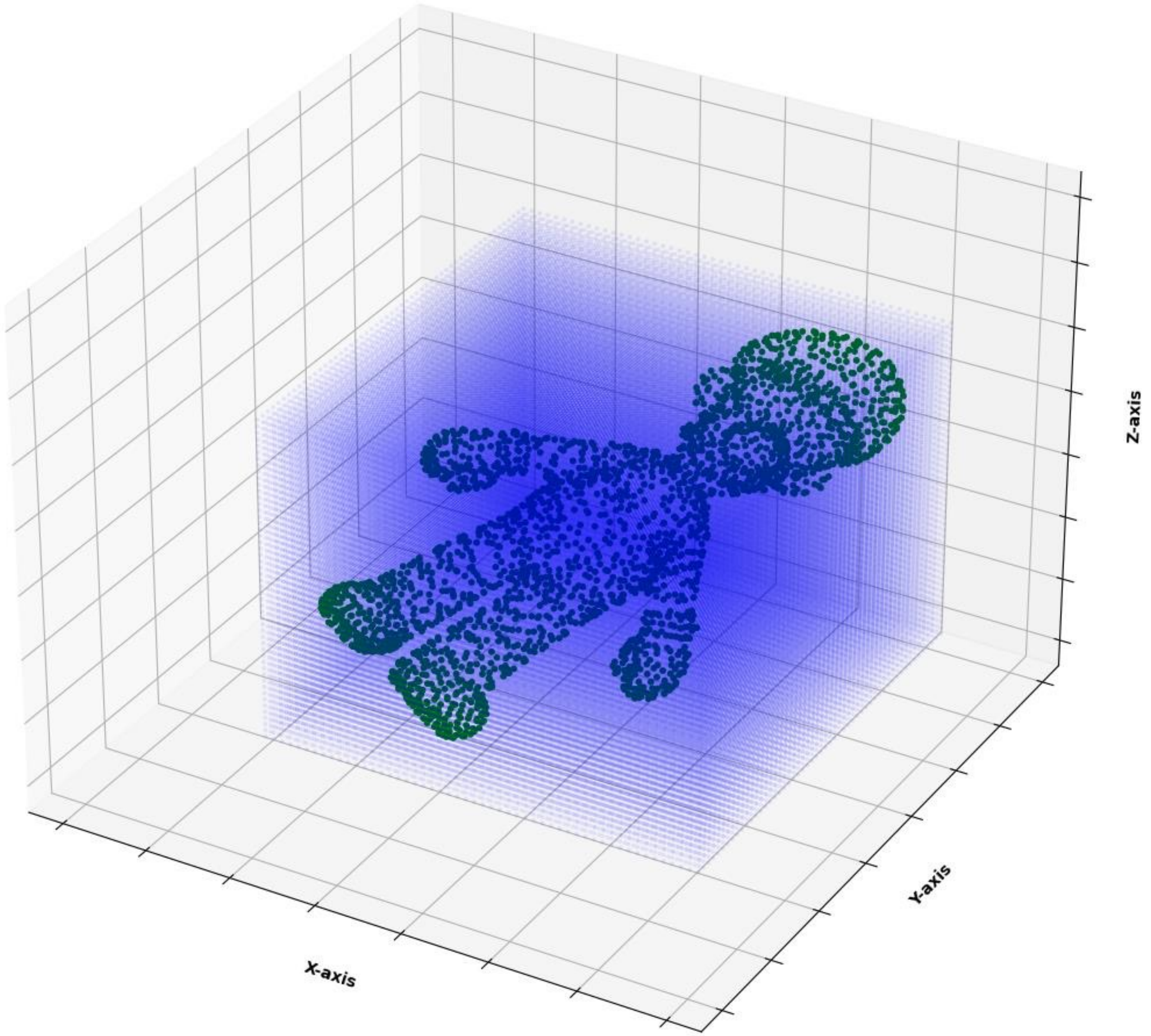
- Create a grid sampling of the 3D space.
- Create a regular volumetric grid around your point cloud
  - Compute the axis-aligned bounding box of the point cloud, enlarge it slightly and divide it into uniform cells (cubes).
- Verify your grids using visualization code.
- Calculated grids will be stored in `grids`.
- Use `np.meshgrid()`.

```
model = 'luigi'

# load shape information
vertices = np.loadtxt('../lab3-1_data/%s/%s_vertices.csv' % (model, model), delimiter=',', dtype=np.float32)
vertex_normals = np.loadtxt('../lab3-1_data/%s/%s_vertex_normals.csv'% (model, model), delimiter=',',
                              dtype=np.float32)

# first, let's create a grid
bmax = vertices.max(axis=0)  # (3, )
bmin = vertices.min(axis=0)  # (3, )
bmax += 5  # add margin
bmin -= 5  # add margin
step = 2
##############################################################################################################
## TODO - (1) : create array named grids # (Gx, Gy, Gz, 3)
```

2. Point Cloud to Mesh (1)



Expected Output

2. Point Cloud to Mesh (1)

- Answer

```python
#################################################################################################################
## TODO - (1) : create array named grids # (Gx, Gy, Gz, 3)
x_linspace = np.arange(int(bmin[0]), int(bmax[0]), step)
y_linspace = np.arange(int(bmin[1]), int(bmax[1]), step)
z_linspace = np.arange(int(bmin[2]), int(bmax[2]), step)
x_grid, y_grid, z_grid = np.meshgrid(x_linspace, y_linspace, z_linspace)
grids = np.stack([x_grid, y_grid, z_grid], axis=-1)
#################################################################################################################
```

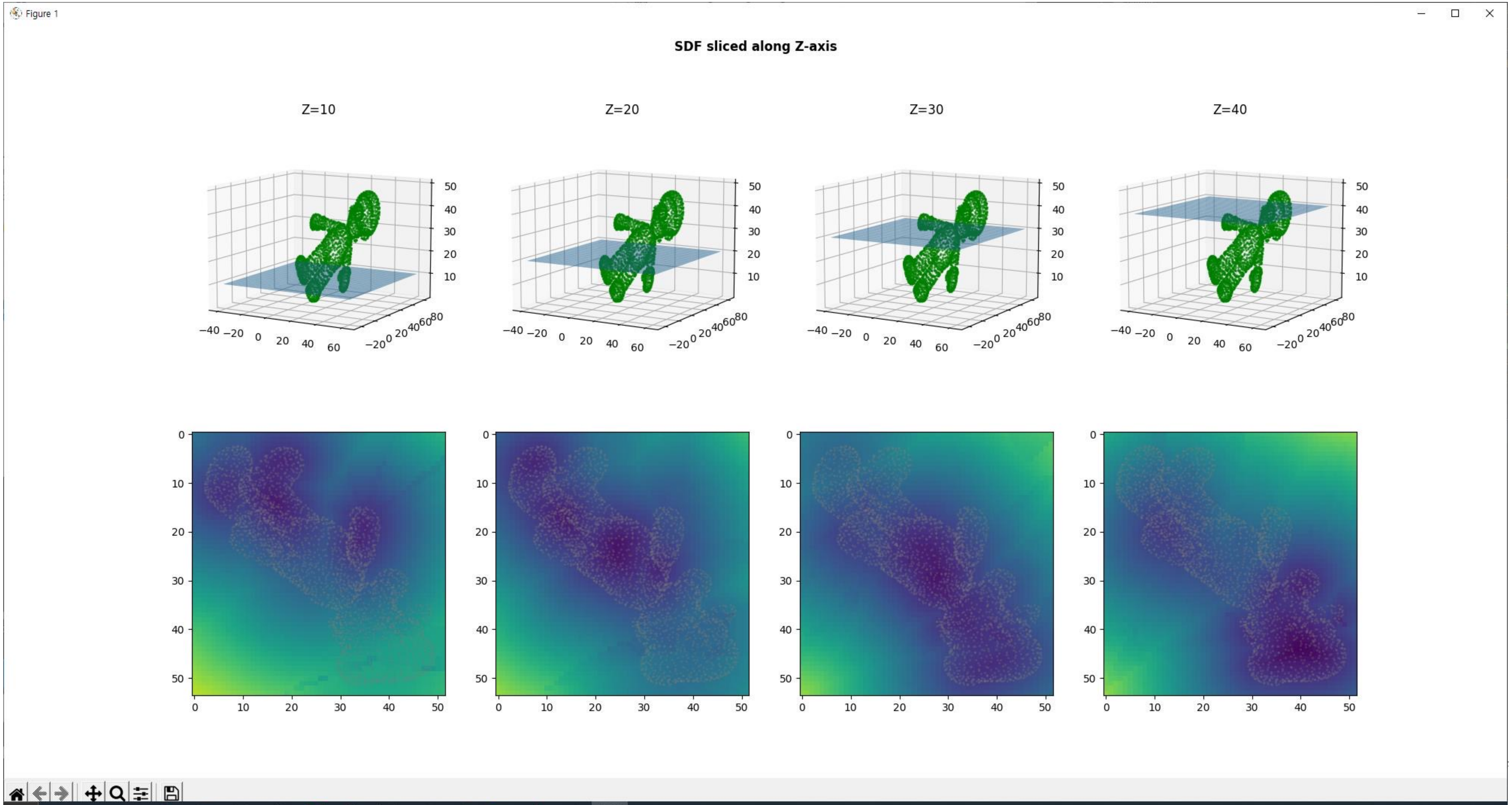
Example Code

2. Point Cloud to Mesh (2)

- Evaluate the signed distance function on the grid points.
  - For each node of your regular grid, find the closest sample from the point cloud and associated normal.
  - Use the normal to compute a signed distance from the grid point to the tangent plane associated with the point cloud sample.
- Note that `grids` here has shape of (Gx * Gy * Gz, 3).
- Verify implementation using visualization code.
- Calculated grids will be stored in `sdf`.
- Use `np.argmin()`.

```
##########################################################################################################
## TODO - (2) : create array named sdf that corresponds grids # (Gx * Gy * Gz, 3)
# SDF calculation
```

2. Point Cloud to Mesh (2)



2. Point Cloud to Mesh (2)

3DLAB VISION

- Answer

```
##############################################################################################################
## TODO - (2) : create array named sdf that corresponds grids # (Gx * Gy * Gz, 3)
# SDF calculation
vertices_self = vertices[:, None]  # (N, 1, 3)
grid_points = np.asarray(grids)[None]  # (1, G*G*G, 3)
diff = vertices_self - grid_points # (N, G*G*G, 3)
dist = (diff ** 2).sum(axis=-1)  # (N, G*G*G)
min_indices = dist.argmin(axis=0) # (G * G * G, )

_, GGG, _ = grid_points.shape
sdf = []
for i in range(GGG):
    sdf.append(np.dot(-diff[min_indices[i], i], vertex_normals[min_indices[i]]))
sdf = np.stack(sdf, axis=0)  # (G * G * G,)
##############################################################################################################
```

Example Code

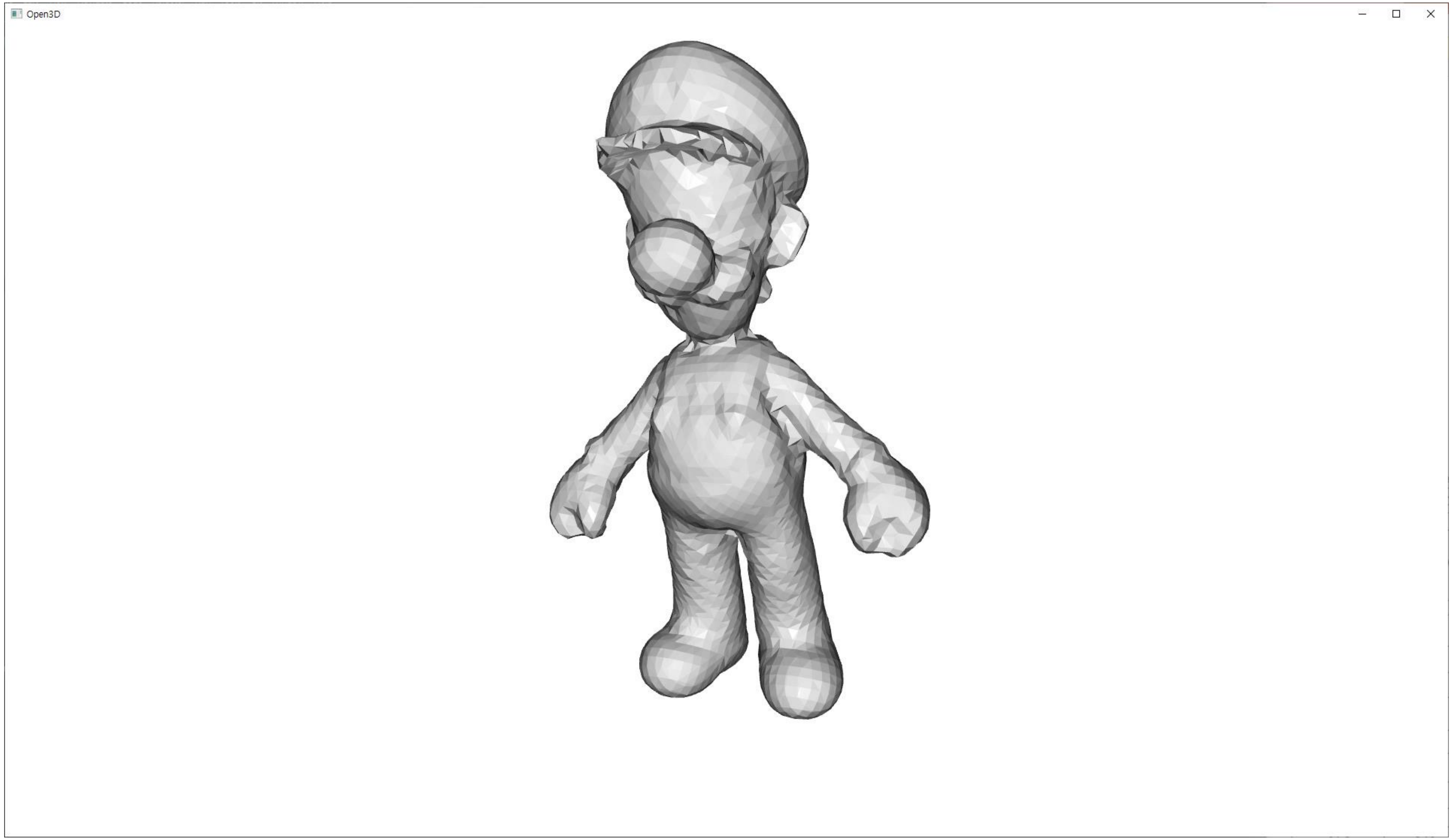
2. Point Cloud to Mesh (3)

- Mesh is reconstructed from estimated signed distance field using marching cubes algorithm.

```
## TODO - (3) : now, verify calculated sdf
# sdf to mesh
sdf = sdf.reshape(*x_grid.shape)
verts, faces, normals, _ = measure.marching_cubes(sdf, 0)
mesh = o3d.geometry.TriangleMesh()
mesh.vertices = o3d.utility.Vector3dVector(verts)
mesh.triangles = o3d.utility.Vector3iVector(faces)
mesh.compute_vertex_normals()

o3d.visualization.draw_geometries([mesh])
```

2. Point Cloud to Mesh (3)



Expected Output