Homework 3

Team Members:

- Atharv Subhekar (CWID: 20015840)
- Prasad Naik (CWID: 20016345)

NOTE: We install the open3d library to generate the point cloud. Please make sure the library is installed before running the code.

Code

```
"""
Authors:
   - Atharv Subhekar (CWID: 20015840)
   - Prasad Naik (CWID: 20016345)
"""
#%% ========== importing libraries ==========
import numpy as np
import xml.etree.ElementTree as ET
import cv2
import open3d as od3

def readData(totalCameras = 8):
    images = []
    silhouettes = []
    projectionMatrices = []

    for i in range(8):
        # storing images
        filename = filename = "cam0%d_00023_0000008550.png"%(i)
        img = cv2.imread(filename)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        images.append(img)

        # storing silhouettes
        silName = "silh_cam0%d_00023_0000008550.pbm"%(i)
        silImg = cv2.imread(silName, cv2.IMREAD_GRAYSCALE)
```

```
        silhouettes.append(silImg)

        # storing projection matrices
        camera = "calibration/cam0%d.xml"%(i)
        tree = ET.parse(camera)
        root = tree.getroot()
        cameraMatrix = np.array((root.text.split()))
        cameraMatrix = [eval(i) for i in cameraMatrix]
        cameraMatrix = np.array(cameraMatrix).reshape(3, 4)
        projectionMatrices.append(cameraMatrix)

    return images, silhouettes, projectionMatrices

#%% ========== generating voxel ==========
def generateVoxelGrid(silhouettes, projectionMatrices, voxelSize = 0.03):
    x_range = np.arange(-2.5, 2.5 + voxelSize, voxelSize)
    y_range = np.arange(-3.0, 3.0 + voxelSize, voxelSize)
    z_range = np.arange(0, 2.51 + voxelSize, voxelSize)

    voxelGrid = np.zeros((len(x_range), len(y_range), len(z_range)))

    fullOutput3D = []
    for i in range(voxelGrid.shape[0] - 1):
        for j in range(voxelGrid.shape[1] - 1):
            for k in range(voxelGrid.shape[2] - 1):

                # center of the current voxel
                voxelCenter = np.array([x_range[i] + voxelSize, y_range[j] + voxelSize,
z_range[k] + voxelSize, 1])

                # flags for checking projection of voxel grid in each silhouette
                voxelFlag = [0] * len(silhouettes)

                # checking voxel reflection in all images
                for l in range(len(silhouettes)):
```

```
                projectedPoint = np.dot(projectionMatrices[l], voxelCenter)
                projectedPoint = projectedPoint / projectedPoint[2]

                if  int(projectedPoint[0]) >= 0 and int(projectedPoint[0]) < 780 and
int(projectedPoint[1]) >=0 and int(projectedPoint[1]) < 582 and
silhouettes[l][int(projectedPoint[1]),int(projectedPoint[0])] == 255:
                    voxelFlag[l] = 1

            if np.sum(voxelFlag) == 8:
                voxelGrid[i, j, k] = 1
                fullOutput3D.append([x_range[i], y_range[j], z_range[k]])

    fullOutput3D = np.array(fullOutput3D)
    totalVoxels = np.prod(voxelGrid.shape)
    totalOccupiedVoxels = np.count_nonzero(voxelGrid)
    ratioVoxels = (totalOccupiedVoxels / totalVoxels)

    print(f"Total Voxels: {totalVoxels}")
    print(f"Total Occupied Voxels: {totalOccupiedVoxels}")
    print(f"Ratio of Occupied Voxels: {ratioVoxels * 100}%")

    return voxelGrid, fullOutput3D

#%% ========== finding surface voxels and generating ply file ==========
def surfaceVoxelGrid(images, projectionMatrices, voxelGrid, voxelSize = 0.03):
    x_range = np.arange(-2.5, 2.5 + voxelSize, voxelSize)
    y_range = np.arange(-3.0, 3.0 + voxelSize, voxelSize)
    z_range = np.arange(0, 2.51 + voxelSize, voxelSize)

    surfaceVoxels = np.zeros((len(x_range), len(y_range), len(z_range)))
    surfaceOutput3D = []
    surfaceSpatialCoordinates = []

    for i in range(1, surfaceVoxels.shape[0] - 1):
```

```
        for j in range(1, surfaceVoxels.shape[1] - 1):
            for k in range(1, surfaceVoxels.shape[2] - 1):
                if voxelGrid[i, j, k]:
                    patch = voxelGrid[i - 1: i + 2, j - 1: j + 2, k - 1: k + 2]
                    if np.sum(patch) < 27:
                        surfaceVoxels[i, j, k] = 1
                        surfaceOutput3D.append([x_range[i] + voxelSize, y_range[j] +
voxelSize, z_range[k] + voxelSize])
                        surfaceSpatialCoordinates.append([i, j, k])

    surfaceOutput3D = np.array(surfaceOutput3D)
    surfaceSpatialCoordinates = np.array(surfaceSpatialCoordinates)
    totalVoxels = np.prod(voxelGrid.shape)
    totalSurfaceVoxels = np.count_nonzero(surfaceVoxels)
    ratioSurfaceVoxels = (totalSurfaceVoxels / totalVoxels)

    print(f"Total Voxels: {totalVoxels}")
    print(f"Total Surface Voxels: {totalSurfaceVoxels}")
    print(f"Ratio of Occupied Voxels: {ratioSurfaceVoxels * 100}%")
    print(f"Output 3D Points: {len(surfaceOutput3D)}")

    xCoords = surfaceSpatialCoordinates[:, 0]
    yCoords = surfaceSpatialCoordinates[:, 1]
    zCoords = surfaceSpatialCoordinates[:, 2]

    xMin, xMax = np.min(xCoords), np.max(xCoords)
    yMin, yMax = np.min(yCoords), np.max(yCoords)
    zMin, zMax = np.min(zCoords), np.max(zCoords)

    colorVals = []
    for point in surfaceSpatialCoordinates:
        red = int(255 * (point[0] - xMin) / (xMax - xMin))
        green = int(255 * (point[1] - yMin) / (yMax - yMin))
        blue = int(255 * (point[2] - zMin) / (zMax - zMin))
        colorVals.append([red, green, blue])
```

```
    colorVals = np.array(colorVals)

    pcd = od3.geometry.PointCloud()
    pcd.colors = od3.utility.Vector3dVector(colorVals.astype(float) / 255.0)
    pcd.points = od3.utility.Vector3dVector(surfaceOutput3D)
    od3.io.write_point_cloud("falseColorRGB.ply", pcd, write_ascii = True)

    candidatePoints = {str(point): np.zeros((8, 3)) for point in surfaceOutput3D}

    for i in range(len(images)):
        occlusionMap = np.matrix(np.ones((images[i].shape[0], images[i].shape[1])) *
np.inf)
        for p in range(len(surfaceOutput3D)):
            point2D = np.dot(projectionMatrices[i], np.append(surfaceOutput3D[p], 1))
            point2D = point2D / point2D[2]
            point2D = point2D.astype(int)

            K = cv2.decomposeProjectionMatrix(projectionMatrices[i])[0]
            Z = (K[0,0] * surfaceSpatialCoordinates[p][1]) / point2D[1]

            if Z < occlusionMap[point2D[1], point2D[0]]:
                occlusionMap[point2D[1], point2D[0]] = Z
                candidatePoints[str(surfaceOutput3D[p])][i][0] = images[i][point2D[1],
point2D[0], 0]
                candidatePoints[str(surfaceOutput3D[p])][i][1] = images[i][point2D[1],
point2D[0], 1]
                candidatePoints[str(surfaceOutput3D[p])][i][2] = images[i][point2D[1],
point2D[0], 2]

    trueRGBColorVals = []
    for point in list(candidatePoints.keys()):
        medianVals = np.median(candidatePoints[point], axis = 0)
        trueRed = medianVals[0]
        trueGreen = medianVals[1]
        trueBlue = medianVals[2]
```

```
        trueRGBColorVals.append([trueRed, trueGreen, trueBlue])
    trueRGBColorVals = np.array(trueRGBColorVals)

    pcd = od3.geometry.PointCloud()
    pcd.colors = od3.utility.Vector3dVector(trueRGBColorVals.astype(float) / 255.0)
    pcd.points = od3.utility.Vector3dVector(surfaceOutput3D)
    od3.io.write_point_cloud("trueColorRGB.ply", pcd, write_ascii = True)

#%% ========== main function ==========
def main():
    # reading data
    print("Preparing Data...")
    images, silhouettes, projectionMatrices = readData(totalCameras = 8)
    print("Data Preparation...Complete")

    # generating and populating voxel grid
    print("\nGenerating and Populating Voxel Grid...Please wait, this may take some
time")
    voxelGrid, allVoxels = generateVoxelGrid(silhouettes, projectionMatrices, voxelSize =
0.03)
    print("Voxel Grid Generating...Complete")

    # finding surface voxels and generating ply file
    print("\nFinding Surface Voxels and Writing False-Color and True-Color .ply
Files...")
    surfaceVoxelGrid(images, projectionMatrices, voxelGrid, voxelSize = 0.03)
    print('ply Files...Successfully Generated')

if __name__ == "__main__":
    main()
```

Explanation

The entire assignment is divided into 4 functions. Explanation of each functions is given below:

- readData(): This functions reads the images, silhouettes and the projection matrices from the given data. Since the projection matrices were provided as XML files, we used Python's XML library to extract the matrices.
- generateVoxelGrid(): We generate the voxel grid with the size of each voxel to be 0.03 in each dimension. After generating the voxel, we populate the center of each voxel by checking if the voxel center is being projected in all the silhouettes.
- surfaceVoxelGrid(): This functions only considers voxels which have at least one empty neighbor voxel. For false RGB coloring, we use the formula given in the homework pdf. For true RGB coloring, we check if there are multiple surface vertices incident on the same point in a camera image plane. If yes, we only consider the surface vertex which is closer to the camera since it implies that vertex to be the one without any occlusion. We take the per channel median of all the RGB values for a surface vertex. We generate the ply files using open3d library.

Outputs

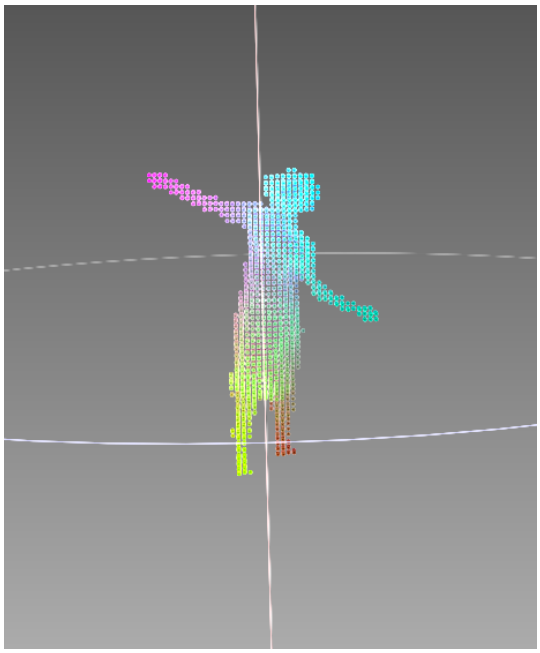

Figure 1 False Color RGB

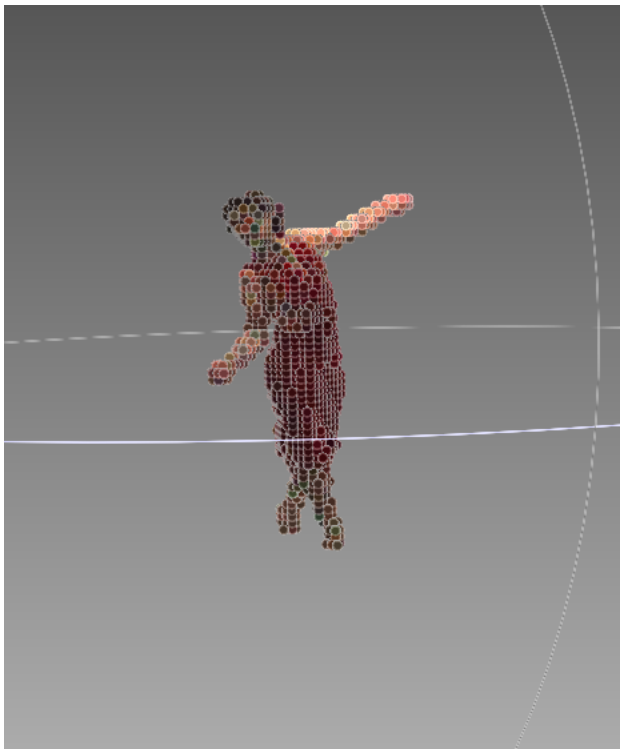

Figure 2 True Color RGB