HW3 report

Jerry Chen

Qiming Ren

import numpy as np  
from PIL import Image  
import open3d as o3d  
  
P = np.array([[776.649963, -298.408539, -32.048386, 993.1581875, 132.852554, 120.885834, -759.210876, 1982.174000,  
 0.744869, 0.662592, -0.078377, 4.629312012],  
 [431.503540, 586.251892, -137.094040, 1982.053375, 23.799522, 1.964373, -657.832764, 1725.253500,  
 -0.321776, 0.869462, -0.374826, 5.538025391],  
 [-153.607925, 722.067139, -127.204468, 2182.4950, 141.564346, 74.195686, -637.070984, 1551.185125,  
 -0.769772, 0.354474, -0.530847, 4.737782227],  
 [-823.909119, 55.557896, -82.577644, 2498.20825, -31.429972, 42.725830, -777.534546, 2083.363250,  
 -0.484634, -0.807611, -0.335998, 4.934550781],  
 [-715.434998, -351.073730, -147.460815, 1978.534875, 29.429260, -2.156084, -779.121704, 2028.892750,  
 0.030776, -0.941587, -0.335361, 4.141203125],  
 [-417.221649, -700.318726, -27.361042, 1599.565000, 111.925537, -169.101776, -752.020142, 1982.983750,  
 0.542421, -0.837170, -0.070180, 3.929336426],  
 [94.934860, -668.213623, -331.895508, 769.8633125, -549.403137, -58.174614, -342.555359, 1286.971000,  
 0.196630, -0.136065, -0.970991, 3.574729736],  
 [452.159027, -658.943909, -279.703522, 883.495000, -262.442566, 1.231108, -751.532349, 1884.149625,  
 0.776201, 0.215114, -0.592653, 4.235517090]])  
  
c\_dict = {}  
s\_dict = {}  
p\_mat = np.zeros((3, 4, 8))  
for i in range(8):  
 c\_dict[i] = np.asarray(Image.open("cam0" + str(i) + "\_00023\_0000008550.png"))  
 s\_dict[i] = np.asarray(Image.open("silh\_cam0" + str(i) + "\_00023\_0000008550.pbm"))  
 p\_mat[:, :, i] = np.reshape(P[i], (3, 4))  
  
x\_range = 5  
y\_range = 6  
z\_range = 2.5  
volume = x\_range \* y\_range \* z\_range  
vox\_num = 10000000  
vox\_size = np.power((volume / vox\_num), 1 / 3)  
vox\_grid = []  
# rvox\_grid = []  
surf\_grid = []  
d\_grid = []  
  
for x in np.arange(-2.5, 2.5, vox\_size):  
 for y in np.arange(-3, 3, vox\_size):  
 minz = 3  
 maxz = -3  
 for z in np.arange(0, 2.5, vox\_size):  
 pass\_mat = np.zeros(8)  
 coord = [x, y, z, 1]  
 for i in range(8):  
 # point = np.dot(p\_mat[:,:,i],np.transpose(coord))  
 point = np.dot(coord, np.transpose(p\_mat[:, :, i]))  
 point = point / point[2]  
 # check if point is within bounds  
 if ((0 <= point[1] < 582) and (0 <= point[0] < 780)):  
 pass\_mat[i] = s\_dict[i][int(point[1]), int(point[0])]  
 # if point is in the silhouette for all 8 views, mark as occupied  
 if (np.sum(pass\_mat) == 8):  
 vox\_grid.append([x, y, z])  
 # rvox\_grid.append([float(str(x)[0:10]),float(str(y)[0:10]),float(str(z)[0:10])])  
 if (z < minz):  
 minz = z  
 if (z > maxz):  
 maxz = z  
 if (minz != 3 and maxz != -3):  
 surf\_grid.append([x, y, minz])  
 surf\_grid.append([x, y, maxz])  
 d\_grid.append('zbot')  
 d\_grid.append('ztop')  
  
  
def clear\_grid(vox, surf):  
 for s in surf:  
 if (s in vox):  
 vox.remove(s)  
  
  
clear\_grid(vox\_grid, surf\_grid)  
for z in np.arange(0, 2.5, vox\_size):  
 for y in np.arange(-3, 3, vox\_size):  
 minx = 3  
 maxx = -3  
 for x in np.arange(-2.5, 2.5, vox\_size):  
 if ([x, y, z] in vox\_grid):  
 if (x < minx):  
 minx = x  
 if (x > maxx):  
 maxx = x  
 if (minx != 3 and maxx != -3):  
 surf\_grid.append([minx, y, z])  
 surf\_grid.append([maxx, y, z])  
 d\_grid.append('xleft')  
 d\_grid.append('xright')  
clear\_grid(vox\_grid, surf\_grid)  
for z in np.arange(0, 2.5, vox\_size):  
 for x in np.arange(-2.5, 2.5, vox\_size):  
 miny = 3  
 maxy = -3  
 for y in np.arange(-3, 3, vox\_size):  
 if ([x, y, z] in vox\_grid):  
 if (y < miny):  
 miny = y  
 if (y > maxy):  
 maxy = y  
 if (minz != 3 and maxz != -3):  
 surf\_grid.append([x, miny, z])  
 surf\_grid.append([x, maxy, z])  
 d\_grid.append('yback')  
 d\_grid.append('yfront')  
  
  
  
color\_grid = []  
d\_dict = {  
 "ztop": 6,  
 "zbot": 3,  
 "yback": 3,  
 "yfront": 0,  
 "xright": 2,  
 "xleft": 5  
}  
for i in range(0, len(d\_grid)):  
 view = d\_dict[d\_grid[i]]  
 coord = surf\_grid[i] + [1]  
 point = np.dot(coord, np.transpose(p\_mat[:, :, view]))  
 point = point / point[2]  
 rgb = c\_dict[view][int(point[1]), int(point[0])]  
 color\_grid.append([float(rgb[0]) / 255, float(rgb[1]) / 255, float(rgb[2]) / 255])  
  
  
print(vox\_size, len(color\_grid), len(surf\_grid))  
pcd = o3d.geometry.PointCloud()  
  
# pcd.points = o3d.utility.Vector3dVector(vox\_grid)  
# o3d.io.write\_point\_cloud("./data.ply", pcd)  
# o3d.visualization.draw\_geometries([pcd])  
  
pcd.points = o3d.utility.Vector3dVector(surf\_grid)  
pcd.colors = o3d.utility.Vector3dVector(color\_grid)  
o3d.io.write\_point\_cloud("./surf.ply", pcd)  
o3d.visualization.draw\_geometries([pcd])

Part 1. Define a voxel grid with x ranging from -2.5 m to 2.5 m, y from -3 to 3 m and z from 0 to 2.5 m. Set the size of each voxel so that the total number of voxels fits comfortably in the memory of your computer. The goal is to estimate which voxels are occupied and which are free space.

Part 2. The goal of this part is to identify surface points that should be included in the output  
3D model. Faces of are represent voxels by one point each. The output is in ASCII ply format. The number after element vertex is the total number of vertices and has to be correct for the model to be displayed correctly. Each vertex is represented by three floating point numbers for the coordinates and three unsigned characters for the RGB colors.

Part 3. Coloring the model by selecting RGB values from the images. First, finding two nearest neighbor points and then finding axis of smallest variance amongst neighbors and corresponding to respective image.