Group Submission

Nikhil Naik

CWID : 20011278

And

Manaswini Kolipaka

CWID : 20012610

Assignment 3

In [5]:
```
import cv2
import numpy as np
import pandas as pd
from xml.dom import minidom
import multiprocessing
from threading import Thread
from itertools import product
import tqdm as tqdm

import all_functions as af
```

In [2]:
```
points = 350
dense_voxel = af.get_visual_hull(points)
surface_voxel, nd_grid = af.calc_surface_voxel(dense_voxel, points)
af.write_ply_file(surface_voxel, [], 0, "textureless_model")
#af.visualize_3d_model("textureless_model")
```

```
Empty Grid Generated
Voxel grid generated for Image 0
Voxel grid generated for Image 1
Voxel grid generated for Image 2
Voxel grid generated for Image 3
Voxel grid generated for Image 4
Voxel grid generated for Image 5
Voxel grid generated for Image 6
Voxel grid generated for Image 7
Occupied and Empty distinguished Voxel grid generated
Calculating Surface Voxel from Voxel grid
Surface Voxel calculated
```

In [25]:
```
dense_voxel_count = dense_voxel[dense_voxel[:,4] == 1].shape[0]
print("""Total voxel points in 3D space = %s \nTotal voxel points which lie inside all sil
Total voxel points on the surface = %s \nRatio of Total silhouette Voxels and Surface Voxe
     % (dense_voxel.shape[0], dense_voxel_count, surface_voxel.shape[0], (surface_voxel.sh
```

```
Total voxel points in 3D space = 42875000
Total voxel points which lie inside all silhouette = 55813
Total voxel points on the surface = 20968
Ratio of Total silhouette Voxels and Surface Voxels = 37.568308458602836 %
```

In [26]:
```
print("As I have used the voxel centers as the 3D points in space, my surface voxel count
```

```
As I have used the voxel centers as the 3D points in space, my surface voxel count and 3D
points count is the same
```

In [3]:
```
if __name__ == '__main__':
    input_multuprocess = list(product(surface_voxel[:,:3], [surface_voxel]))
    with multiprocessing.Pool(10) as pool:
        results = pool.starmap(af.get_data_and_color, tqdm.tqdm(input_multuprocess, total=
    results = np.array(results)
    datas , colors = results[:,0], results[:,1]
    af.write_ply_file(datas, colors, 1, "textured_model")
    af.visualize_3d_model("textured_model")
```

```
100%|██████████| 20968/20968 [4:24:40<00:00,  1.32it/s]
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

In [14]:
```
af.visualize_3d_model("textureless_model")
```

Output Images

There is also a video in Output_data folder

In [15]:
```
print("Textureless Model")
from IPython import display
display.Image(".\\Output_data\\Textureless_3d_model.png")
```

```
Textureless Model
```

Out[15]:

In [16]:
```
print("Textured Model")
from IPython import display
display.Image(".\\Output_data\\Textured_3d_model.png")
```

```
Textured Model
```

Out[16]:

Explanation of code

- All functions are in a seperate file to enable multiprocessing in jupyter notebook
- Visual hull was generated by first creating empty voxels using numpy linspace, creating a array with rows consisting of each voxel and column = x,y co-ordinte on the image and value = sillouette pixel value , these arrays were concatinated in axis=1 to create a large array where columns represented different camera. If all columns had 255 value, they were considered as occupied points and were updated accordingly in the grid.
- To calculate surface voxel, a 3x3x3 kernel was used whose center point is occupied ie 1, then we sum all values of the kernel to see if it is 27, if not 27 atleast one of the positions in kernel is empty and we consider the point as surface voxel.
- All this process just take a minutre to run using numpy matrix multiplication instead of looping where ever possible, At this point we can create textureless model
- After this we use multiprocessing to generate textured model to improve time, even with multiprocessing it took more than 4 hours to render the textured model.
- To get color of each surface voxel,
    - We first create a vector from point to current camera (cam center determined from null space of p matrix)

    - We use concepts used in ray tracing (https://medium.com/swlh/ray-tracing-from-scratch-in-python-41670e6a96f9) to determine if a point intersects with the cam - point vector or not.
    - For each point we choose all other surface points to see if any of them intersect with the view of current camera
    - We choose all camera which have un-interrupted view to the point and determine median color and assign it to the point
- We generate Textured 3D model.

In [ ]: