CS117 Final Project Report

Name: Yuerong Zhang

1. Project Overview

In the final project, we are trying to solve the problem that after taking photos of an object, we lose the information such as the object's location and depth, which demonstrate how it is represented in three-dimensions in the real world, since the photos are in two-dimensions. Therefore, the goal of the project is that, for any object, we would like to generate a good 3D reconstruction of it, by using a collection of 2D scan photos with the structured illumination. For interests, in this project, we choose to reconstruct a 3D ceramic teapot.

2. Data Description

There are mainly two sets of data that we use for our reconstruction process. The first one is a series of chessboard photos taken from 21 different locations, which help us to calibrate our cameras for the teapot later. The second one is a collection of our teapot photos with 7 different angles, including 5 angles in front of the teapot, 1 angle from its head, and 1 angle towards its bottom. All these 7-angles scans provide us a 360-degree view of the teapot, which then will be used to generate 7 fragments that combined together to form a complete teapot. For each angle, we have background photos, teapot photos without lights, and structured light teapot scans from both the left and the right cameras. In this project, the structure light on the teapot can let us make the 3D correspondence among 6 teapot fragments easily.

3. Algorithms

We implement our algorithm of the 3D reconstruction of the teapot in the following seven steps, using the Python code and the tools in Meshlab (a 3D mesh processing software).

1) Use the collection of the chessboard images to calibrate intrinsic parameters and extrinsic parameters of the object's scanner cameras.

   Generally, we manipulate the chessboard images (calib_jpg_u directory) to calibrate our scanner cameras. First, we run calibrate.py (provided by the instructor) to obtain the intrinsic camera parameters, which are saved in calibration.pickle file. Then, we select a pair of chessboard images taken from the left and the right cameras, and we use cv2.findChessboardCorners function in OpenCV to get the extrinsic camera parameters. Therefore, we obtain the values of focal length, principle point, rotation matrix, and translation matrix for our cameras.

2) Take the object and background images from the scanner cameras to reconstruct 3D points.

   We implement a reconstruction function here. First, we compute an object mask by thresholding the difference of foreground object and

background color images and compute a decoding mask. Then, we apply the combination of these two masks, find the matching pixels from the left and right object images, and construct the 2D points of the matched pixels. Finally, we use the left and right 2D points and cameras to get the reconstructed 3D points by the triangulate function. In the end, we record the RGB color of the corresponding pixels in the foreground object images for the later use. We also visualize the resulting 3D points for the later use in the mesh generation procedure. The pseudo code of reconstruction function is shown below.

```
def reconstruct(threshold,camL,camR):

    # compute an object mask
    # threshold the difference of object and background
    object_mask = absolute(foreground - background) > threshold

    # Decode the Horizontal and Vertical coordinates for the two views
    Code,mask = decode(threshold)

    # Construct the combined 20 bit code C = H + 1024*V and mask for each view
    # Combine object mask with decoding mask
    C = H + 1024 * V
    mask = mask_horizontal * mask_vertical * object_mask

    # mask the undecodable
    C_good = C[mask]

    # Find the indices of pixels in the left and right code image that
    # have matching codes.
    match = intersection(C_good_left, C_good_right)

    # generating the corresponding pixel coordinates for the matched pixels.
    # triangulate the points
    pts3 = triangulate(pts2L,camL,pts2R,camR)

    # Record the color of the corresponding pixel in color img
    colors = foreground[pts2L[1,:],pts2L[0,:],:].T

    return pts2L,pts2R,pts3,colors
```

3) Implement mesh generation function including the cleanup.

First, we call the reconstruct function in the previous step to get the initial points and color of the object, and then triangulate 2D points to obtain the object mesh.

To clean up noisy points, we use both the bounding box pruning and the triangle pruning. In the bounding box pruning, it cuts off the points that are outside the bounding box given certain ranges by the user. In the triangle pruning, we first use Delaunay to obtain a list of triangles and compute three edges of each triangles. Later, we use the triangle edge threshold to filter edges that are longer than the threshold. After that, we make a new list of triangles based on the index map constructed from the vertices indices we have kept. We also filter the pts3, pts2L, pts2R, color with these kept indices. The pseudo code of our mesh generation function, including the cleaning procedure, is shown below.

```
def mesh_generate(scan_dir,pose,camL,camR,threshold,boxlimits,trithresh,pickle_file):

    # Reconstruct points
    pts2L,pts2R,pts3,colors = reconstruct(imprefix,imprefixL,imprefixR,threshold,camL,c
amR)

    # Bounding box pruning
    pts = np.nonzero(pts3 > boxlimits)

    pts3 = pts3[:,pts[0]]
    pts2L = pts2L[:,pts[0]]
    pts2R = pts2R[:,pts[0]]
    colors = colors[:,pts[0]]

    # Triangulate the 2D points to get the surface mesh
    T = scipy.spatial.Delaunay(pts2L.T)
    tri = T.simplices

    # Triangle pruning

    # Edges in triangles is a,b,c
    tri_thre = (a<trithresh)&(b<trithresh)&(c<trithresh)
    tri = tri[tri_thre,:]

    # Remove any points which are not refenced in any triangle
    # vertices to keep
    keep = unique(tri)

    # index of triangles
    tri = index_map[tri]

    pts3 = pts3[:,keep]
    pts2L = pts2L[:,keep]
    pts2R = pts2R[:,keep]
    colors = colors[:,keep]
```

After the cleanup, we save pts3, pts2L, pts2R, color, and triangles into a pickle file for visualization, and also use meshutil.py to create the corresponding ply file for the later use in Meshlab.

4) Implement the mesh smoothing function.

Use the slicing windows to compute and replace the average of nearest points for every 3D points, in order to smooth the mesh.

After the cleanup procedure, we implement the algorithm to smooth the surface of our generated mesh from the previous step. By using the find_neighbors function from StackOverflow (https://stackoverflow.com/questions/12374781/how-to-find-all-neighbors-of-a-given-point-in-a-delaunay-triangulation-using-sci), we can find the nearest connected points, or neighbor points, for all 3D points in the mesh triangles. Then, for every 3D point, we compute the average of its neighbors and replace it with this average value. To get a good smoothing result, we repeat this computation twice in the function. The pseudo code of the mesh smoothing function is represented below.

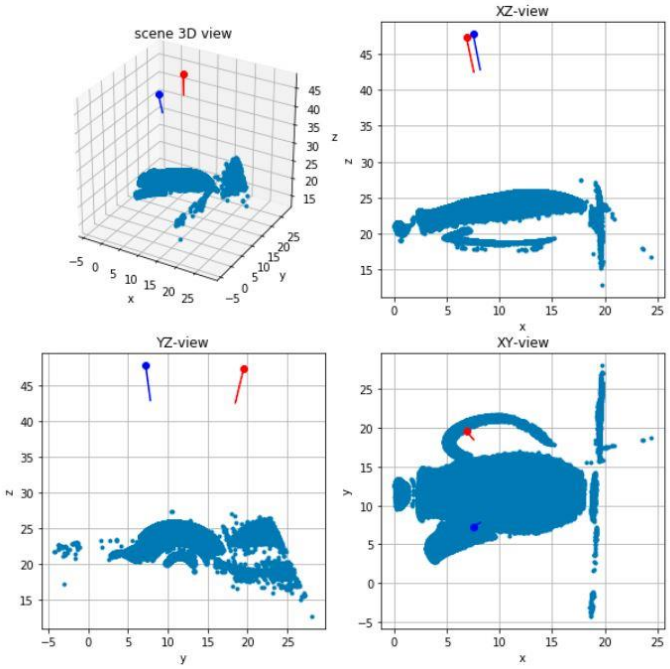
```
def mesh_smoothing(triangles,pts3,k_times):
    # get neighbors (in triangles) for every 3D points
    neighbors = find_neighbors(triangles)

    # run mesh smoothing for k times
    for k times:
        # replace each 3D point with the average of its neighbors
        for each 3D point in pts3:
            point = average(pts3[neighbors of point])

    return pts3
```

5) Generate meshes for different scans of the object.

Since there are seven different scans for the teapot, we use the mesh generation function, including the mesh smoothing, to generate one mesh for each of seven scans. To get the good resulting meshes, we experiment with different values of the mask threshold, bounding box limits, and triangle threshold parameters for each scan. For the bounding box limits parameter in mesh cleanup, we visualize the initial reconstructed 3D points (implemented in the previous reconstruction step) as the plots below, and then set the box limits for x, y, z coordinates by looking closely to these plots.



Finally, we display the resulting mesh by the Python's trimesh library for experiments.

6) Align the different scans of the object in Meshlab.

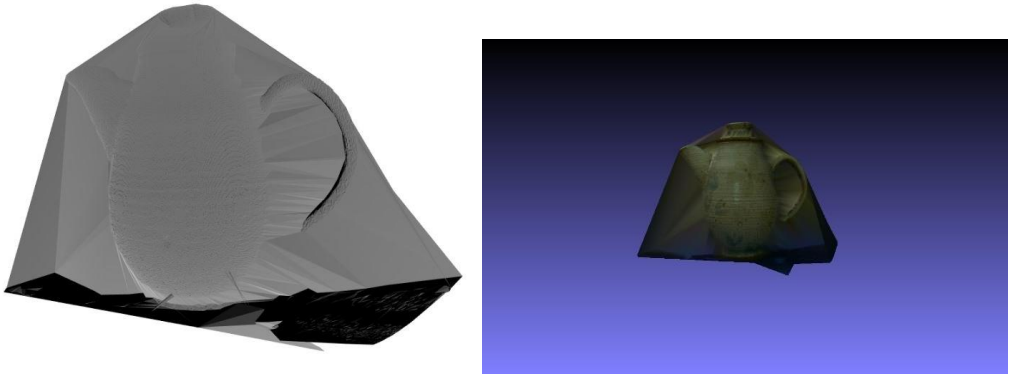
The seven meshes we generated in the previous step are fragments of the teapot, and we have to combine them together to get a complete one. We use the Align tool in Meashlab to implement this step. First, we import seven ply files saved by the mesh generation function into Meshlab, and then finding the corresponding 3D points among different scan meshes for the alignment. Finally, we get a single mesh for the complete teapot.

7) Run Poisson surface reconstruction of the aligned scan data.

After the alignment, we find that there are many holes in our single teapot mesh. We then apply the Screened Poisson surface reconstruction tool in Meshlab to the teapot mesh and set the reconstruction depth to be 12 to obtain a perfect result. In the end, we have a high quality of the 3D reconstructed teapot without holes or broken parts in the mesh.

4. Results

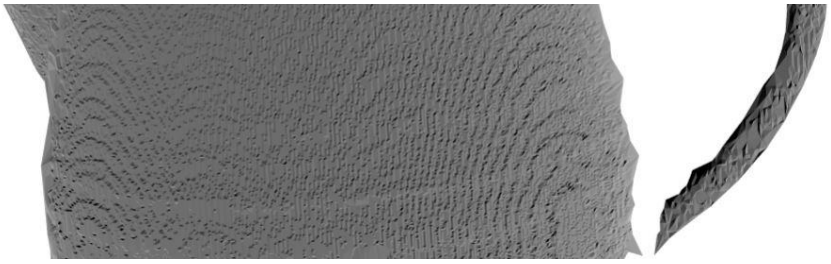
1) Mesh cleaning algorithm results:



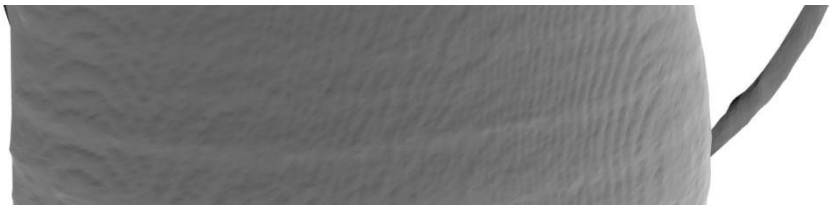
Scan_0: Before mesh cleaning

Scan_0: After mesh cleaning

2) Mesh smoothing algorithm results:



Scan_0: Before mesh smoothing



Scan_0: After mesh smoothing

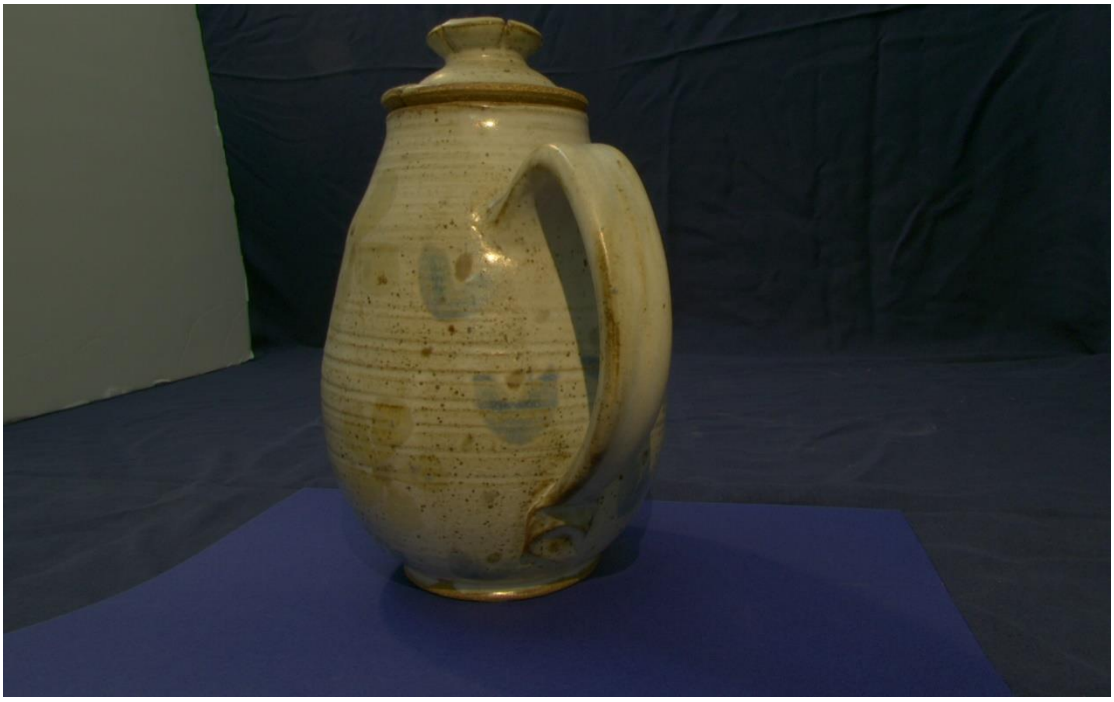
3) Final teapot model:

Scan_0

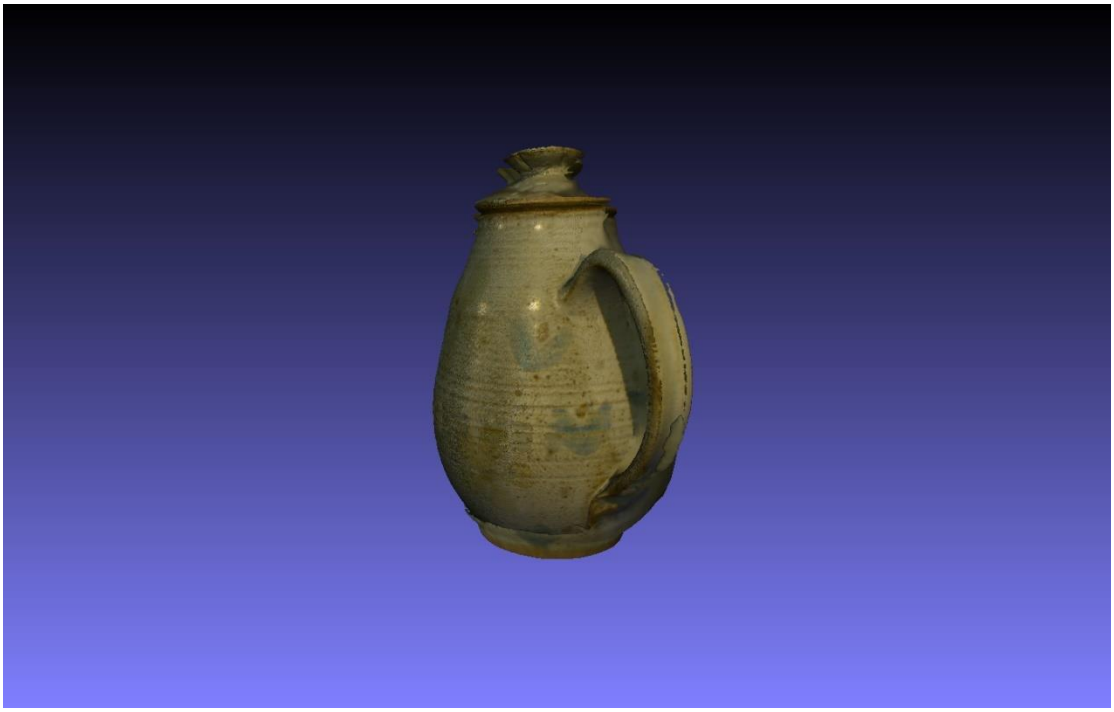
Real image

Final model

Scan_1

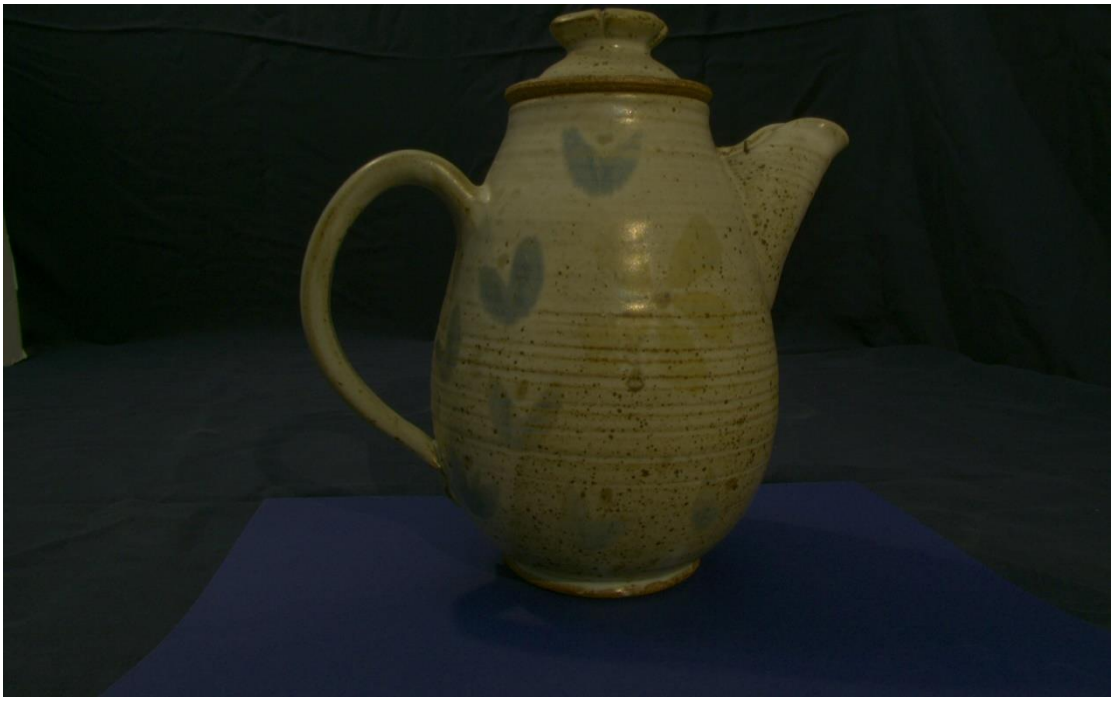


Real image



Final model

Scan_2



Real image

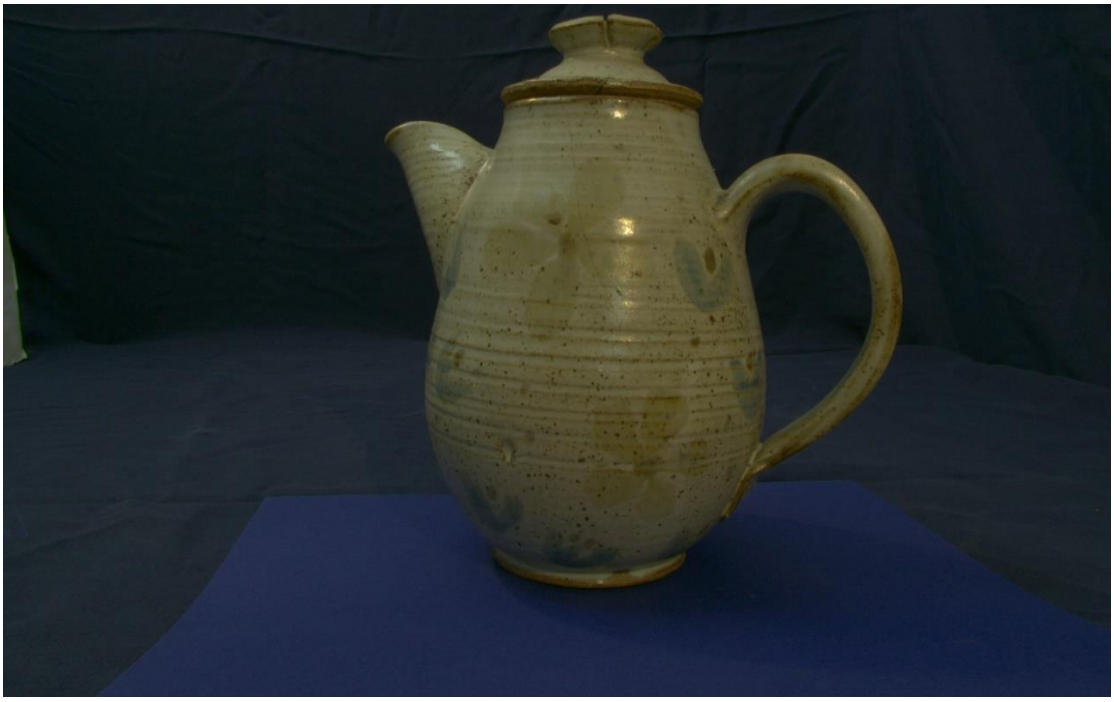
Final model

Scan_3

Real image

Final model

Scan_4

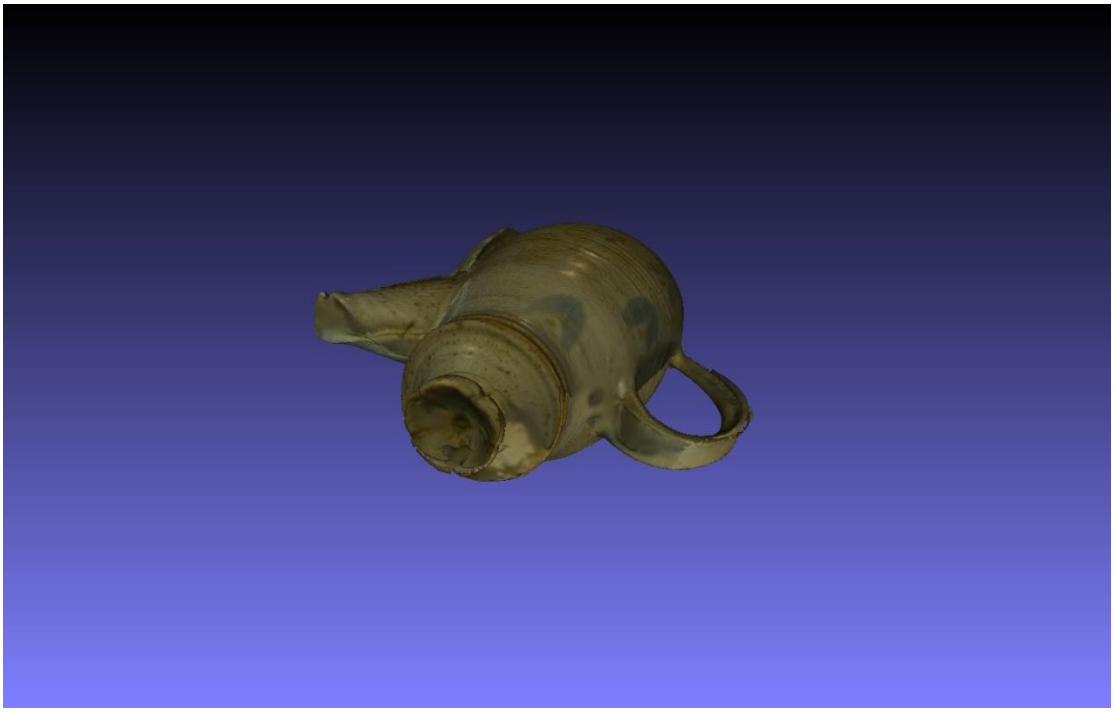


Real image

Final model

Scan_5

Real image



Final model

Scan_6

Real image

Final model

5. Assessment and Evaluation

Generally, the final teapot model is good, though the alignment of meshes needs improvement further.

After finishing this project, I think overall it is not much difficult for a computer algorithm to solve the 3D reconstruction of an object from the 2D structure illumination scan images. Although our algorithm requires much knowledge in cameras and computer vision, we can complete it with several functions in Python code by mostly the math computation. During this project, I get stuck on generate seven different teapot scans in the beginning, since I find the same parameters in mesh generation cannot suit different scans. Therefore, I had experimented many different mask threshold and cleanup parameters for different scans.

In my approach to the final teapot model, there are two major limitations. The first limitation is our algorithm for cleaning noisy points in the mesh generation. From the results, we can see that there is still much noise on the outlines of our teapot, which not be separated completely from the background. This is also the part in this project that I get stuck on. To get a better mesh, I have tried to remove the noise by lowering the triangle mesh pruning parameter, which works but this also removes some details in the teapot mesh. It is difficult to find a balance between less noise and less details when using our algorithm for the cleanup. The second limitation is that I use the Meshlab tools to align different teapot scans manually, which, in a result, the final teapot model is not aligned perfectly as some meshes are in the wrong relative locations. From the final model, we can see that the bottom of teapot is not aligned well, and the teapot is skew to some degrees. If there is more time for this project, I will implement a function to automatically find the corresponding 3D points between different meshes to accomplish the alignment process more efficiently and obtain a better result. Also, I will improve the algorithm for the mesh cleaning to get clearer outlines for the teapot meshes.

6. Appendix

1. Calibration script from the instructor's code.
2. Reconstruct function modified from the instructor's code.
3. Mesh generation function modified from the previous assignment.

4. Find_neighbors function from StackOverflow, https://stackoverflow.com/questions/12374781/how-to-find-all-neighbors-of-a-given-point-in-a-delaunay-triangulation-using-sci

5. Mesh smoothing function written by myself.

References:

6. Final Project Guidelines and Resources Page in Canvas.

7. Previous assignments.

8. **https://stackoverflow.com/questions/12374781/how-to-find-all-neighbors-of-a-given-point-in-a-delaunay-triangulation-using-sci**

9. Utilities function provided by the instructor.