Jacob Nardone
netID: nardonej
sid#: 73508576

CS117 Final Project Report
3D Mesh Reconstruction

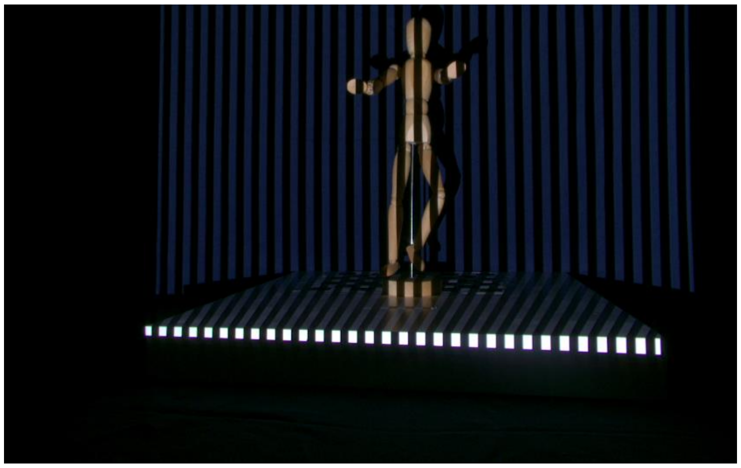
Brief Problem Overview:

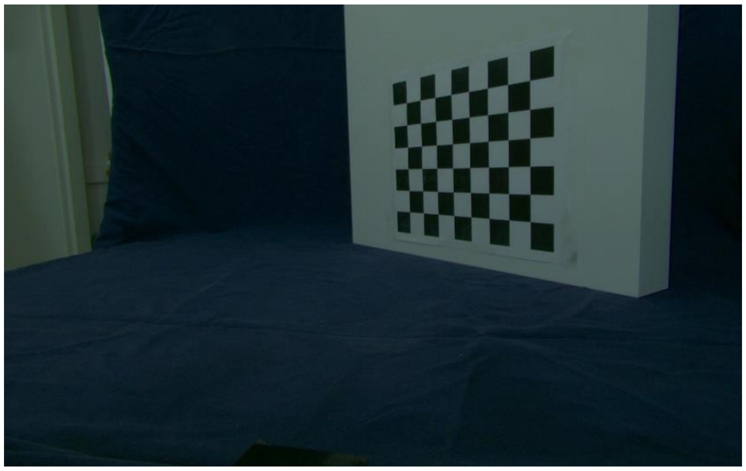
The problem I am trying to solve is creating and stringing together a pipeline that takes scan images of an object (with structured illumination) and outputs mesh data that defines that object so it can be represented as a clean 3D model reconstruction. In short, the goals of this project are to calibrate my cameras, reconstruct the 3-dimensional object points, convert my point clouds into meshes, align and tune up my meshes, and lastly render out my final object model.

The Data:

The data consisted of 5 “grab” directories where each grab contained (a) background images for both the left and right cameras, (b) object (mannequin) images for both the left and right cameras, and (c) 40 structured illumination photos (of the mannequin) for both the left and right cameras. The structurally illuminated sets of 40 were split evenly into a horizontal light pattern and a vertical light pattern. There was also a calibration directory that contained 20 photos with the chessboard (to calibrate the camera’s intrinsic parameters). Each photo had a dimension of 1920 x 1200 pixels.

Example data images:


structured illumination

calibration image


Step 1: Calibrate the cameras

First, I needed to calibrate the camera parameters (both intrinsic and extrinsic). I discovered that the camera setup was entirely the same just by looking at the images in each grab folder, so I knew I only had to calibrate the camera once. I got the intrinsics by running calibrate.py on the “calib_jpg_u” directory, which created a pickle file that stored the intrinsics. Next, I had to replicate what we did in assignment 3 to get the extrinsic camera parameters for the images in the “grab” folder. I used the OpenCV.findChessboardCorners and calibratePose functions to detect the corners in the first images for the right and left camera in the calibration folder and compute the rotation and translation of each camera.

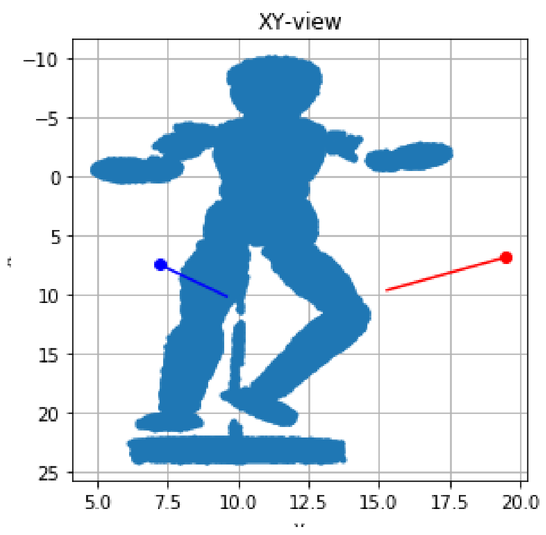
Difficulties: My original plan was to use the “color” images at the beginning of the grab folders as my extrinsic calibration photos; however, that didn’t work because OpenCV.findChessboardCorners couldn’t detect the chessboard in those photos because the board was at too odd of an angle. So, I instead used C0_01 and C1_01 from the calibration folder to calibrate the right and left cameras respectively. I was also getting odd results for the translations of both cameras (they seemed to be relatively the same), so I had to figure out to change the y-axis initial rotation parameter so that the calibration could find a better minima, and that got me more sensible and expected extrinsic values.

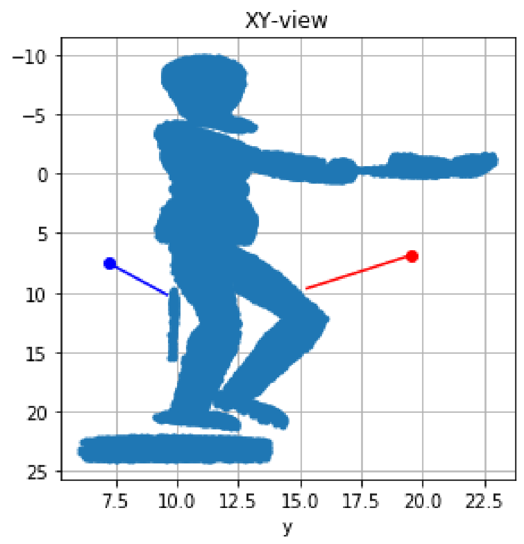
Step 2: Apply an object mask

I called camutils.reconstruct to get the reconstructed 3D point values based on the 20-bit gray codes of the two views. I had to make a few modifications first, though. To prevent triangulating points that are part of the background, I created another mask inside of the “decode” function that selected only the pixels of the object (using the “color” images at the top of the grab folders), and I combined this mask with the decoding mask using the AND operator to keep only the points of the mannequin object.

Step 3: Store and prune color values

In the reconstruct function, I also needed to keep track of the color values so that the mannequin meshes would have color like in the images. I created a 3xN array called “colors” by using the indexes in pts2R to access the object color image of whatever grab set I was currently working on and stored the color at that pixel. I then included “colors” to be returned from my reconstruct function. I used the 2D visualizations and the Trimesh 3d interactive views to determine how I needed to tune the bounding box pruning to get rid of all the pesky background/foreground noise.



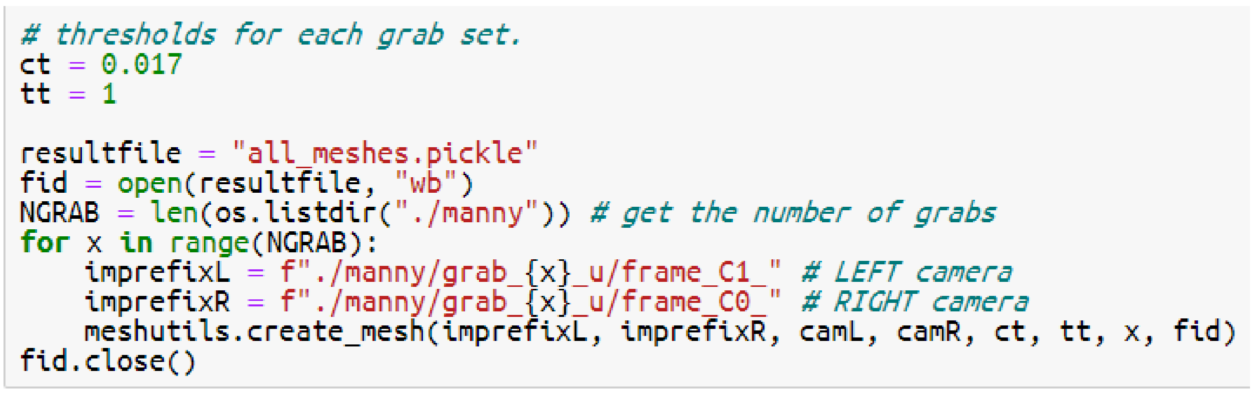


Step 4: Create meshes for each grab

Here I wrote some code to iterate through each grab in the “manny” directory. I took my pruning code from assignment 4 and put it into a function called “prune” in camutils.py. For each grab set, I reconstruct all the 2D and 3D points as well as their corresponding colors, call prune on them, and then store the pruned points, triangles, and colors into a pickle file so I can reload them easily later.

Difficulties: I had some difficulties getting pickle to store correctly. I initially tried opening and dumping the mesh data within the create_mesh function, but the problem that is was causing the all_meshes.pickle to be overwritten every time the function was being called. To fix this I created a test pickle file and found out what I needed to do was open the pickle result file first and then pass in the file ID every time I called create_mesh, this way I could store all of the meshes and not just one.

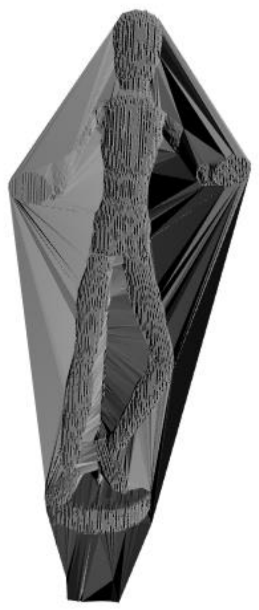
Here’s some jupyter code that I wrote to run create_mesh on each of the 5 scans and write all the mesh data to a pickle file:



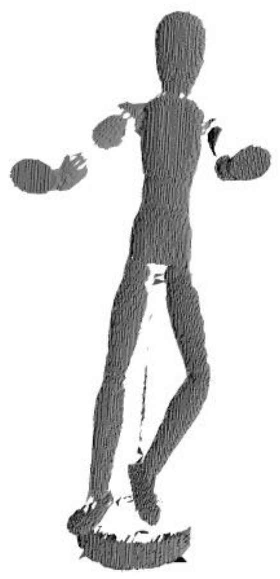
```
# thresholds for each grab set.
ct = 0.017
tt = 1

resultfile = "all_meshes.pickle"
fid = open(resultfile, "wb")
NGRAB = len(os.listdir("./manny")) # get the number of grabs
for x in range(NGRAB):
    imprefixL = f"./manny/grab_{x}_u/frame_C1_" # LEFT camera
    imprefixR = f"./manny/grab_{x}_u/frame_C0_" # RIGHT camera
    meshutils.create_mesh(imprefixL, imprefixR, camL, camR, ct, tt, x, fid)
fid.close()
```

Trimesh without triangle pruning



Trimesh with triangle pruning



Step 5: Mesh smoothing

Clearly the mesh above needs some smoothing. For mesh smoothing, I created a function inside of meshutils.py called “smooth_mesh” that takes in some mesh data and returns the same mesh but smoothed instead of being jagged. I passed in the points array, the triangles array, and an integer representing the amount of times I want the function to run the smoothing loop. In the function I loop over every point. I realized that I could use the triangles array to figure out what other points were connected by some triangle with the current point in the iteration. I then retrieve the points at those indices, compute the average location of those points, and then

Difficulties: I had an error in my function originally because I mixed up the orientations of the dimensions for the triangles array. With N being the number of triangles returned by Delaunay, I had assumed it was a (3xN) array when it was really (Nx3).

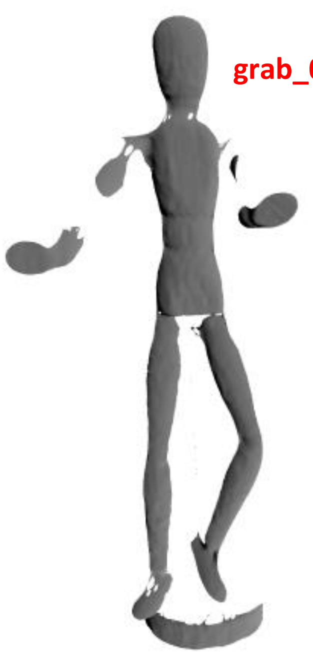
Here’s the jupyter code that ran the mesh smoothing, created the Trimesh objects, and ran the writeply function to create the 5 ply files:

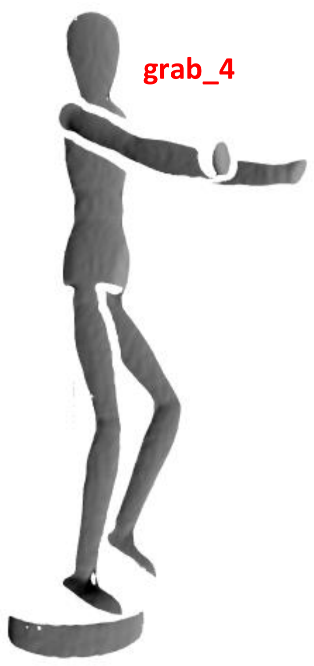
```
fid = open(resultfile, 'rb') # Open the pickle file that stores all the meshes
all_meshes = []

for x in range(NGRAB):
    mesh_data = pickle.load(fid)
    p3,tri,colors = mesh_data.values()
    p3 = meshutils.smooth_mesh(p3, tri, repeat=5)
    mesh = Trimesh(vertices=p3.T, faces=tri[:,[0,2,1]])
    all_meshes.append(mesh) # for later viewing in the notebook
    meshutils.writeply(p3, colors, tri, 'mesh' + str(x+1) + '.ply')

fid.close()
```

After smoothing (5 times)



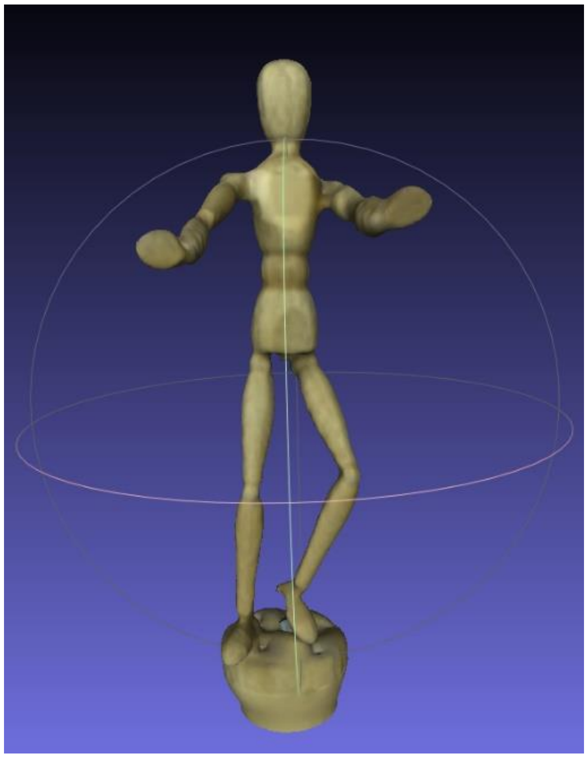
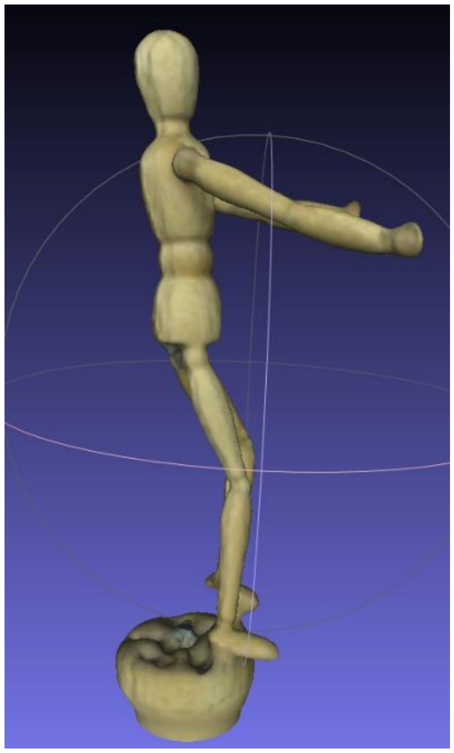
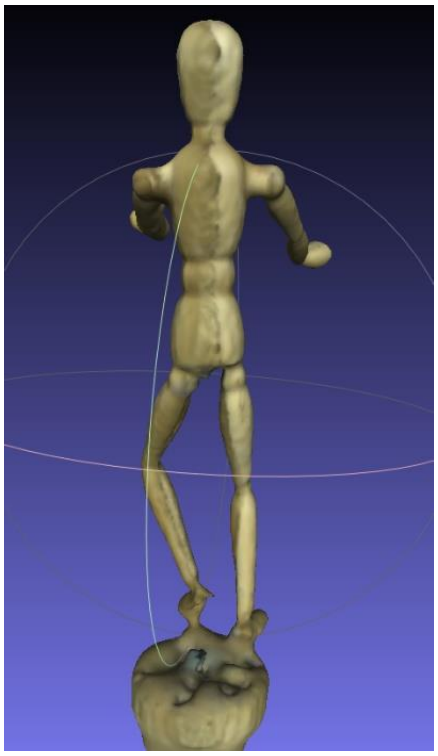


Step 6: Combine the meshes

To align the meshes into one coherent mesh object, I loaded each of the 5 ply files into mesh lab. Initially they were not aligned at all. To align them, I followed a YouTube tutorial which involved using the point-based alignment tool within MeshLab. For each mesh, I had to click on the corresponding points on the other meshes, and after doing this for all meshes, MeshLab did its magic and automatically aligned the entire set near perfectly.

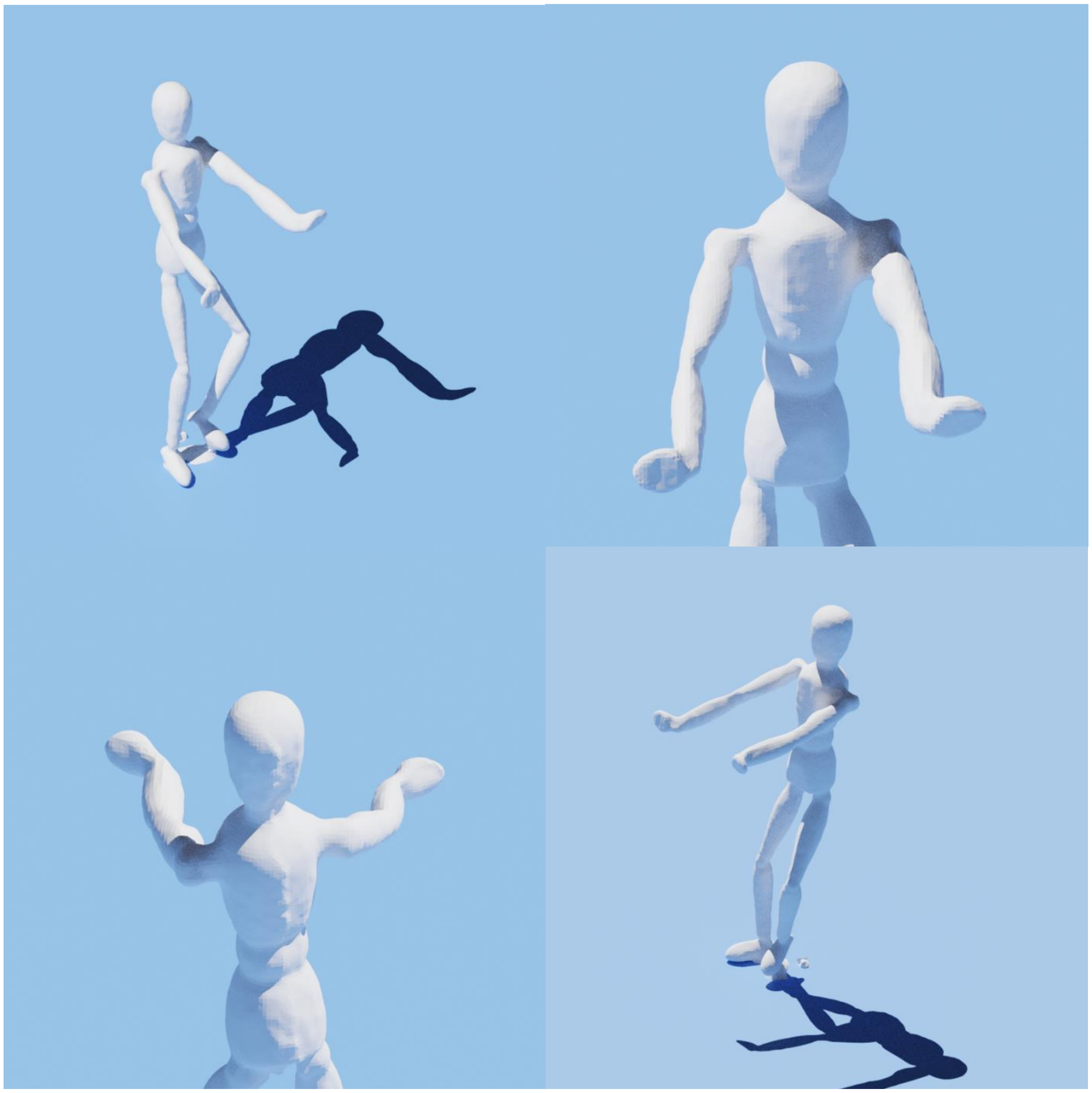
Step 7: Poisson surface reconstruction

Using YouTube, I learned about MeshLab’s built-in Poisson Surface Reconstruction filter. First, I merged all the aligned meshes into one mesh by flattening the layers, and then I ran the surface reconstruction on the new merged layer. This not only filled the holes of the mesh to make it watertight, but it also smoothed the entire mesh and blended the colors between each mesh much more evenly to look more natural. Lastly there was an issue where there were floating chunks of mesh from the thin pole that the mannequin was help up by. This looked messy, so I used MeshLab’s select tool do manually select the points/faces and removed them from the final mesh.

Step 8: Generate final renderings

I exported my finalized mesh into a ply file and opened the ply in Blender. I unfortunately could not figure out how to get the vertex colors to appear in Blender; nonetheless, the lighting and rendering options in Blender made for some really cool images of Manny.



Step 9: Refactor

The last thing I did was refactor the code. I moved a couple functions from the camutils.py to the meshutils.py because it made more sense there. I also made sure to add some docstrings and inline comments to make my code more understandable.

Important Algorithms:

Here are some pseudocode algorithms that I used in this project:

Triangle Pruning algorithm:

```
for each triangle in all_triangles:
    for i=0 to 3:
        vertex_1_index → all_triangles[i]
        vertex_2_index → all_triangles[(i+1) mod 3]
        point_1 → points[vertex_1_index]
        point_2 → points[vertex_2_index]
        if distance_between(point_1, point_2) > threshold:
            remove triangle from all_triangles
            break inner loop
```

Mesh Smoothing algorithm:

```
for each repetition:
    for x=0 to amount_of_all_points:
        pointX → all_points[x]
        nbr_idx → set(all indexes in pointX's triangles)
        nbrs → points[nbr_idx]
        average → mean(nbrs)
        all_points[x] → average
```

Assessment and Evaluation:

Overall, the project was challenging but very fun to work on. If I had more time, I probably could have developed a stronger smoothing algorithm that better preserved the flatness of certain surfaces. If I had taken the images myself, I could have grabbed some more image angles to make my final mesh more accurate. In the end I am very satisfied with the results and I think I made as much progress as I could have given the limitations of the project. I was very impressed with how well MeshLab aligned everything and reconstructed the parts of the mannequin that the images did not pick up.

Appendix:

Functions I wrote from scratch:

- meshutils.create_mesh
- meshutils.smooth_mesh

Functions I modified from my previous assignments:

- meshutils.prune

Functions I modified from the instructor:

- camutils.decode
- camutils.reconstruct
- visutils.vis_scene

Functions/Classes/Modules I used as is from the instructor:

- meshutils.writeply
- camutils.makerotation
- camutils.Camera
- camutils.triangulate
- camutils.residuals
- camutils.calibratePose
- calibrate.py