Daniel Du
45054949

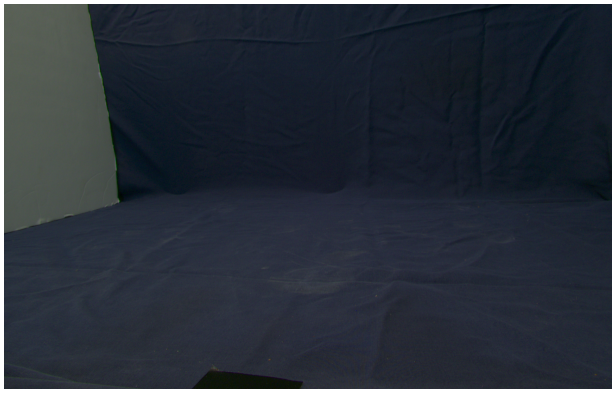
CS117 Final Report

Overview:

The overall goal of this project is to take multiple scans of an object to reconstruct a 3D mesh. Since the scans of the meshes were already provided, there are only two major steps to accomplish this goal: writing code to read the scans and convert them to triangulated meshes, and piecing those meshes together in Meshlab to form the final model.

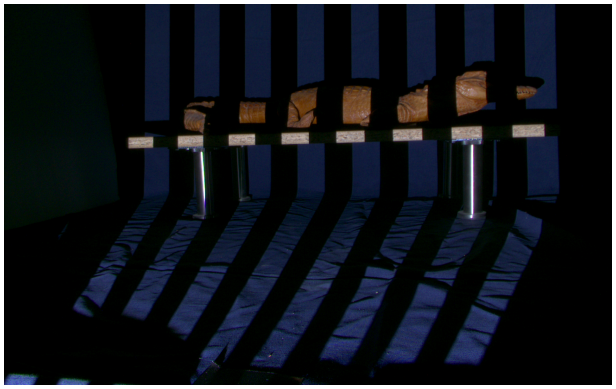
Data:

I used the provided scans as the data for this project, which included multiple chessboard images for camera calibration, as well as scans for 4 different objects. Each of these objects had 5 to 7 groups of images consisting of background and foreground color images, as well as structured illumination scans, with each of these groups representing a different angle of the object. Below I've provided screenshots of an example of the foreground, background, and structured illumination scan for the crocodile object.



Background Image

Foreground Image



Structured Illumination

In addition to the provided scans, I also ran the script in calibrate.py to get the intrinsic parameters of the camera, using the provided chessboard images in calib_jpg_u.

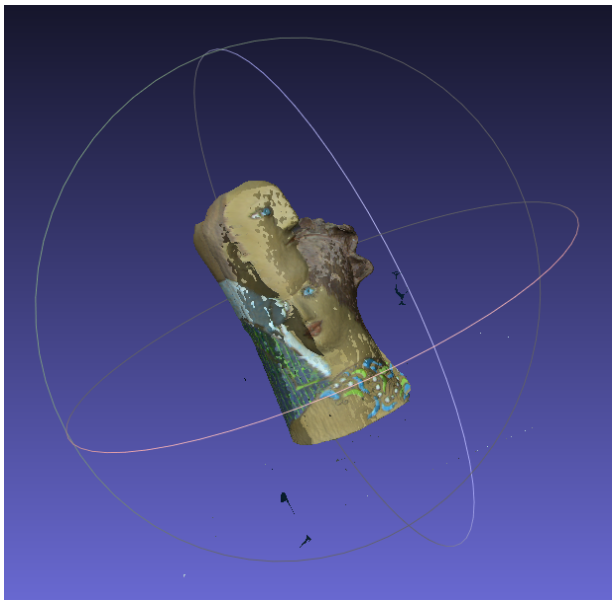
Algorithms:

Multiple functions were provided in the python scripts and were used in the camera calibration and reconstruction. The camera calibration only used open cv and the provided functions. The two functions used for reconstruction were Decode and Reconstruct. Decode converts the provided structured Illumination scans from 10 bit gray code to a grayscale image, along with an image mask. This function is used in Reconstruct along with the provided triangulate function to generate the 3D points of the 2D images provided with the camera parameters. In addition, Reconstruct took the difference of the foreground and background color images provided to generate a mask that could be combined with the decode mask to get a good representation of the object's background. I also used the average color of each 3D point in the color images to record the color for the given 3D point in the mesh.

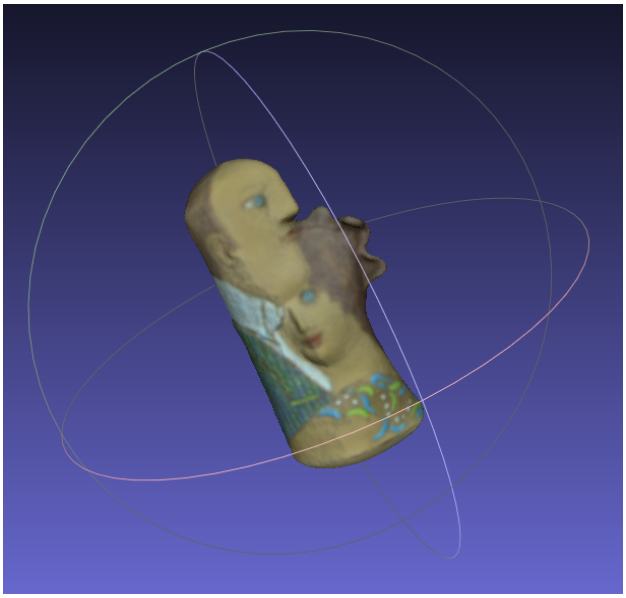
After reconstruction, the scipy library is used to generate triangles to connect the 3D points into a mesh. I used a for loop to filter out any triangles that had edges longer than the threshold I had of 1 unit. I then deleted any points that weren't part of the triangle mesh. Before exporting the mesh, I applied a smoothing algorithm, which found the set of all points connected to each 3D point, and set that point equal to the average of the set of points. After saving the meshes to .ply files, all that was left was to align them in Meshlab.

Results:

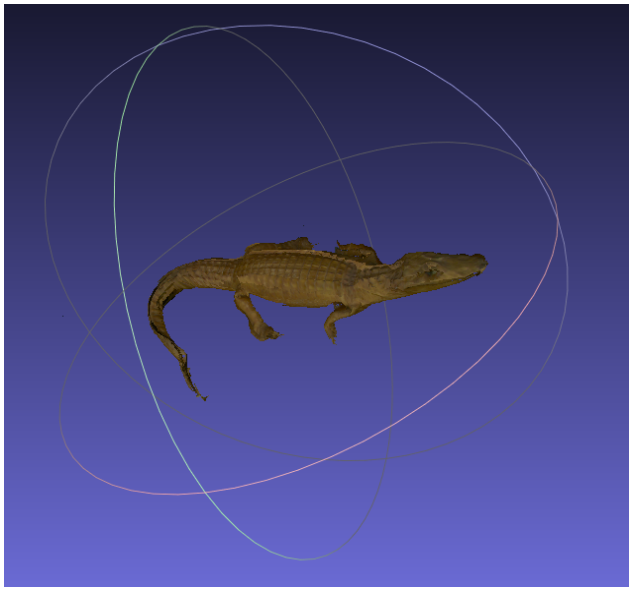
I imported the meshes into Meshlab and cut out some of the noise. I then used the align tool to line up the meshes to form the 3D mesh. The final step was to apply the Poisson Surface Reconstruction to combine the meshes. Below I've provided screenshots of the aligned meshes as well as final reconstructions for all four objects. Because the Poisson Surface Reconstruction combines the meshes into one object, the final reconstruction tends to look blurrier than the aligned meshes, while the aligned meshes will have multiple gaps or protruding parts that the final reconstruction eliminates.



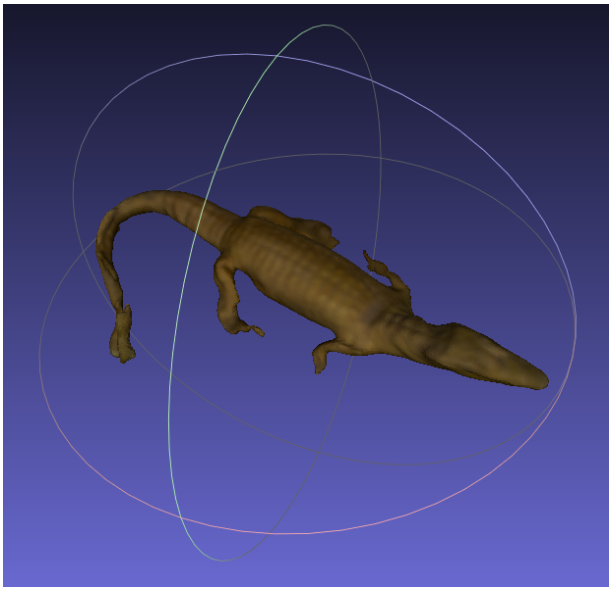
Couple Aligned Meshes



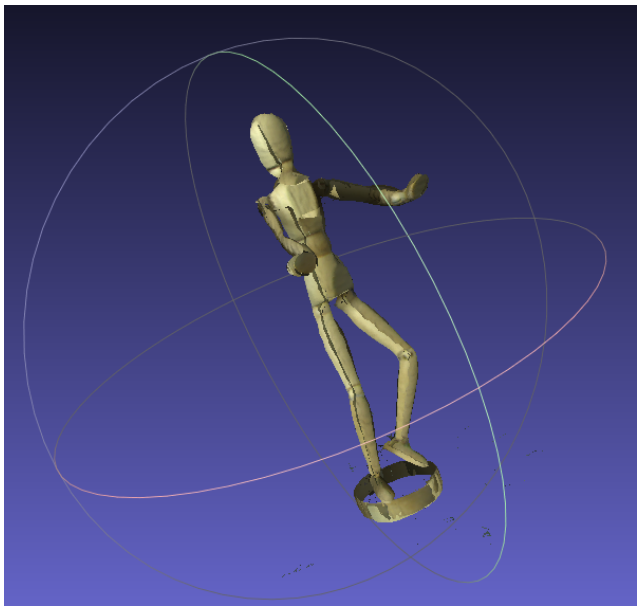
Couple Final Reconstruction



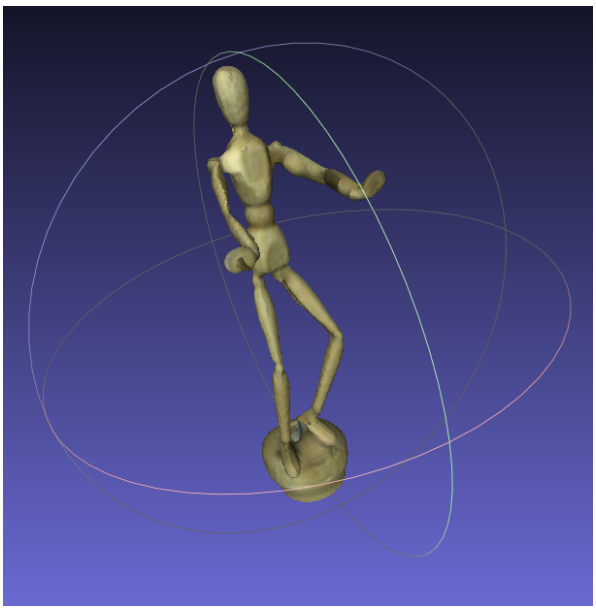
Crocodile Aligned Meshes



Crocodile Final Reconstruction



Mannequin Aligned Meshes



Mannequin Final Reconstruction

Teapot Aligned Meshes

Teapot Final Reconstruction

Assessment and Evaluation:

The final project for this course was relatively easy given the fact that much of the work was completed in previous assignments. In general, it tended to be more time consuming, rather than difficult, because it required running the code on multiple scans, as well as stitching them together with Meshlab.

One of the biggest struggles was getting suitable meshes for aligning in Meshlab. Since the scans were provided, the only thing I could alter to get different meshes was the parameters for the code. Because of this, it was hard to find alignments for some of the meshes, namely the crocodile, because the meshes didn't line up very well with each other. On the other hand, some of the meshes lined up pretty well and resulted in pretty realistic models, like the teapot.

Given more time or another chance, I would want to figure out how to take my own scans, and try using those with my code. This way, if my meshes are being difficult to align, I can take more scans, or ones at different angles to generate better meshes. This way I would have more options to improve my meshes besides just trying to perfectly optimize my code's parameters.

Another thing I would change is implementing my own alignment function. If I were to have made my own alignment function as opposed to using Meshlab, I would be able to have more control over how the meshes lined up. Since some of the meshes, like the crocodile as I mentioned earlier, didn't really match up, having a better understanding of how alignment works as well as having my own function to edit would give me more options to align the meshes instead of repeatedly trying out different sets of points until the meshes happened to line up.

Appendix:

All of my code is in the jupyter notebook named final_project.ipynb. In the appendix, I'll list out what is contained in each cell.

Imports: The first cell contains most of the imports used throughout the project, such as numpy, opencv, and matplotlib.pyplot.

Camera Calibration: The second cell uses the provided chessboard images to calibrate the camera. This code was taken from assignment 3, where we used the chessboard images to calibrate the camera for reconstruction. This segment takes the calibration.pickle file generated from the calibrate.py file and calculates the extrinsic parameters. The only difference from the assignment 3 code was that I removed the visualization of the chessboard images.

Decode: The third cell is the decode function, taken straight from assignment 4.

Reconstruct: The fourth cell is the reconstruct function, taken from assignment 4. There were two adjustments to this function. One was the implementation of a mask created by subtracting the background from the foreground image. The other was saving the color of the points in a separate array that would be returned as well.

Run Reconstruct: The fifth cell runs the reconstruct function on all the scans.

Cleanup: The sixth cell is taken from assignment 4, and cleans up the mesh. I removed the box limits because it was troublesome to get parameters for all the different objects.

Smooth: The seventh cell runs a function called avgpts 5 times of the mesh. This function sets each 3D point to the average of the connected 3D points. This effectively smooths the mesh before exporting it.

Export: The final cell using meshutils to write the mesh to a .ply file before displaying the mesh with trimesh.