Matthew Ayres
28394916
CS 117 Final Project Report

Project Overview

The main problem attempting to be solved in this project is being able to create 3D models of an object utilizing colored 2D image scans on said object. These scans are taken from various angles rotating around the object using different frames of coded light. Using these 2D images of objects covered in coded light the goal is to create a decently accurate 3D model that is similar in color and mesh topology to the real-life object.

Data

Two main data sets were used in this project to achieve our goal of creating a 3D object from image scans. Our first data set, calib_jpg_, contains 21 frames of chessboard scans for each left and right camera. The image set is used to get the intrinsic and extrinsic camera parameters to be used in the object reconstruction and mesh creation. The second image set is a collection of 7 image sets containing 7 different angles of the object covered in the different strips of coded light. Each set contains scans from the left and right camera, as well as a background scan from each camera and as well a scan of the object not covered in scanned light. These are used to help create an object mask for reconstruction and to get accurate color data from the pixels. Each of the 7 object image sets contains 40 scans for each camera. Both data sets have a resolution of 1920 x 1200 pixels.

Algorithms

Calibrate.py script was provided by the instructor and is used to gather the intrinsic camera parameters from chessboard images from a provided directory, calib_jpg_u. The intrinsic parameters calculated from this script are then used to find the extrinsic camera parameters.

Using OpenCV's findChessboardCorners function we are able to get the extrinsic parameters. The function is proved with an image of the chessboard and its row and column dimensions, it returns the estimated extrinsic coordinates of the camera. This is done for both the left and right camera.

The function calibratePose is given by the instructor it takes the camera's created from the intrinsic parameters of the calibrate.py script and updates them using least squares and the difference of the chessboard corners coordinates, from the OpenCV function, and the camera's location to estimate and update the camera's extrinsic parameters to have the smallest error possible, updating the camera's rotation matrix and translation vector.

Decode is a function provided by the instructor that loads in a set of grey coded images and returns an array which is the same size as the image where each element is its decoded counterpart, it also returns a binary image mask of pixels that were correctly decoded based on a user defined threshold. The function converts standard 10 bit grey code into standard binary to XOR the bits, we then produce a final decimal value using a binary to decimal conversion.

B is the binary created from the grey coded. $\sum 9n=0\ B[9-n]*2n$

Matthew Ayres
28394916
CS 117 Final Project Report

Triangulate is an instruction-provided function. It triangulates the pair of left and right coordinate points returning 3D coordinates relative to the global positions of the two cameras. It does this by solving for the z point of each point in the 2d coordinate plane in a system of equations for the left and right camera using a least squares estimate, updating the translation and rotation matrices.

The function reconstructColor is an updated version of the reconstruct function written in assignment 4, using techniques discussed in lecture. The function performs a matching and triangulation of points on the surface using the decoded structured light images. It calls upon the decode function to decode the structured light patterns, it creates an object mask to differentiate the pixels of the object in the foreground of the images, the object we are trying to recreate, and the pixels in the background of the image, that we don't need for recreation. Using the masks we match the pixels with corresponding points in the left set of images and points from the right set of images. It then calls upon the triangulate function to turn all the matched left and right points into points in a 3D coordinate system. We save the average RBG value from the decoded left and right sets of images in order to add proper color back to our mesh later. Lastly, we save the color array, the left 2D points, the right 2D points, and the triangulated 3D points into a pickle file for easy testing of mesh generation parameters.

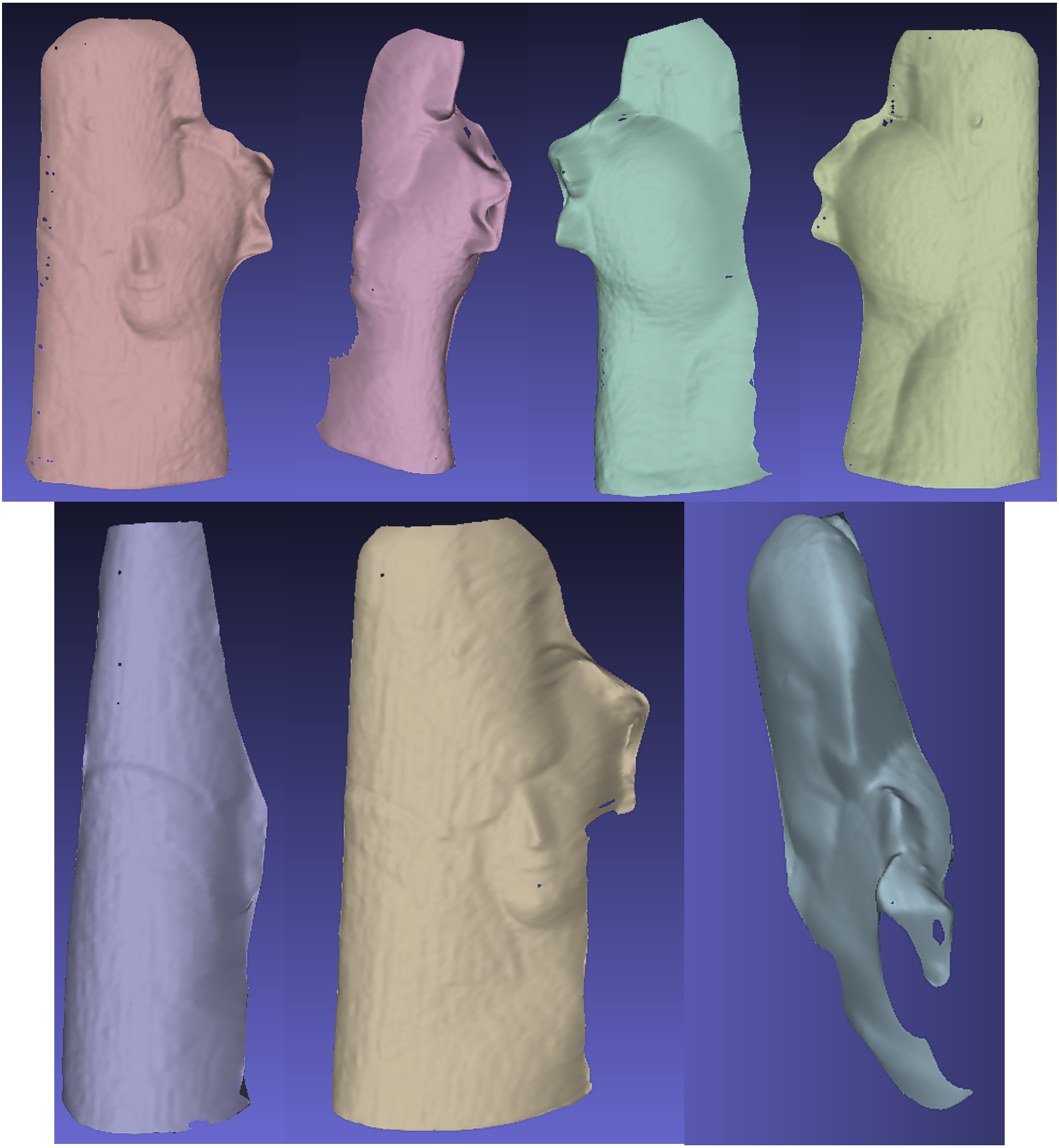
The function meshGen is a modified-wrapped version of the mesh generation code from assignment 4. We are given the 3D coordinate points, the left and right 2D coordinate points, a bounding box limits array, a triangle pruning max threshold, and the array of the pixel colors. Using the limits set in the bounding box array we remove all the points that fall outside of the bounding box. We do this for the 3D coordinates, both 2D coordinates and the color array. Next we create a list of triangles by calling upon the function scipy.spatial.Delaunay on the left 2D coordinates Delaunay helps create a nice triangulation by choosing a triangulation whose smallest angle is as large as possible. We then keep the indices of the points forming the simplices in the triangulation. Next we perform triangle pruning which removes triangles on the surface of the mesh that are longer than our threshold parameter, we also make sure to remove any neighbors of points that have been removed from the triangle pruning. Now that a lot of points have been removed, we remap old indices to new indices in our 3D coordinate set, both of the 2D coordinate sets, the colors array and the tri's array of simplies. Lastly, now that all the points we wanted removed are gone we smooth our mesh. Using a function called find_neighbors we get a list of all the neighbors of the vertices in tri. We now smooth our mesh by adjusting the points in the 3D coordinate set to settle on the average of their neighboring points, which we found using the find_neighbors function. We perform this smoothing process 6 times. Finally we export our results: 3D coordinate array, both 2D coordinate sets, the colors array and the tri(the list of the indices of the simplices); to a .pickle file for testing.

Lastly, writeply function provided by the instructor saves out triangulated mesh from the meshGen function to a .ply file so that it may be imported into meshLab for alignment and poisson reconstruction.

Matthew Ayres
28394916
CS 117 Final Project Report

Results

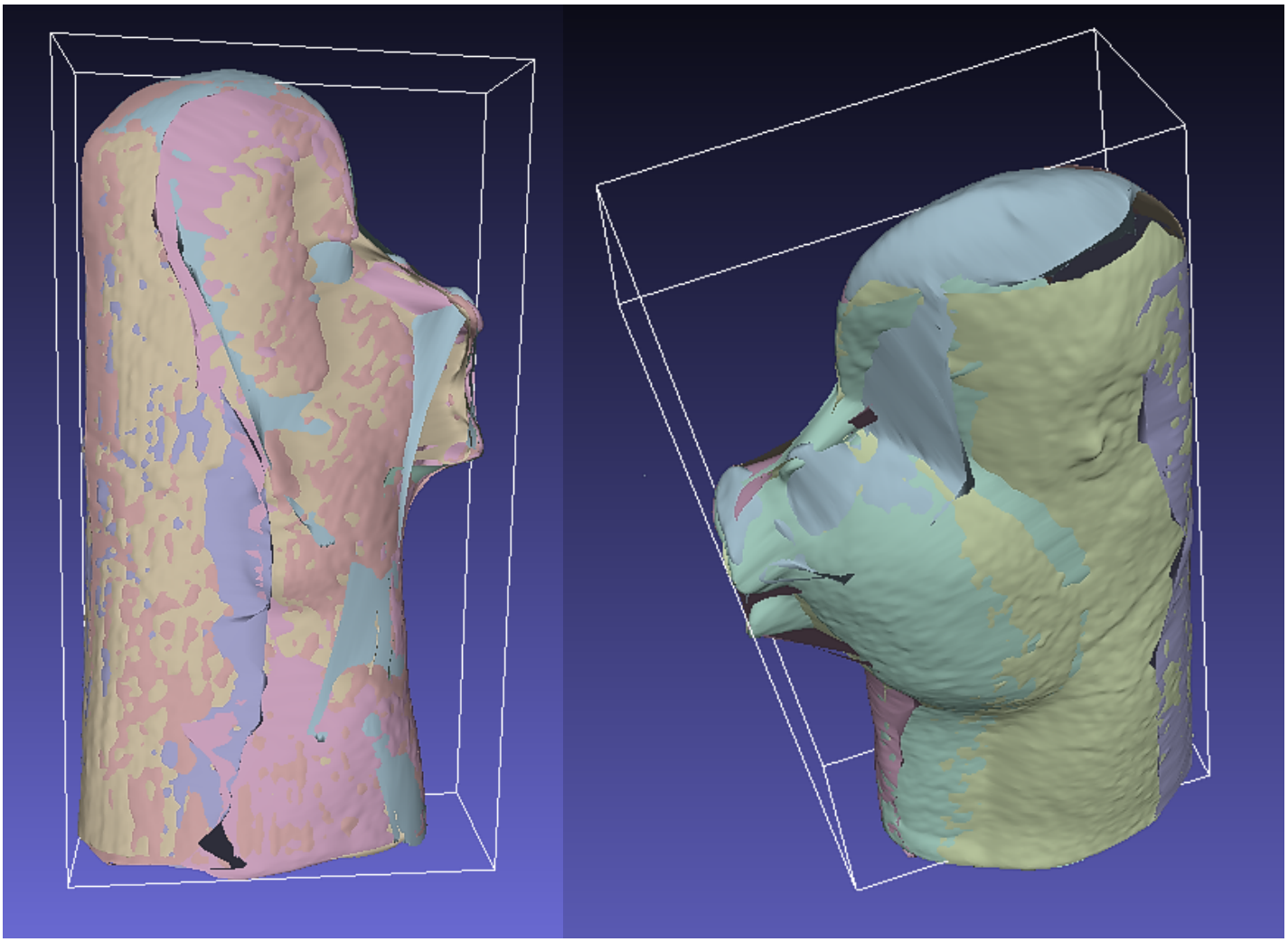
Matthew Ayres
28394916
CS 117 Final Project Report



All meshes before alignment from grab_0_ to grab_6_u

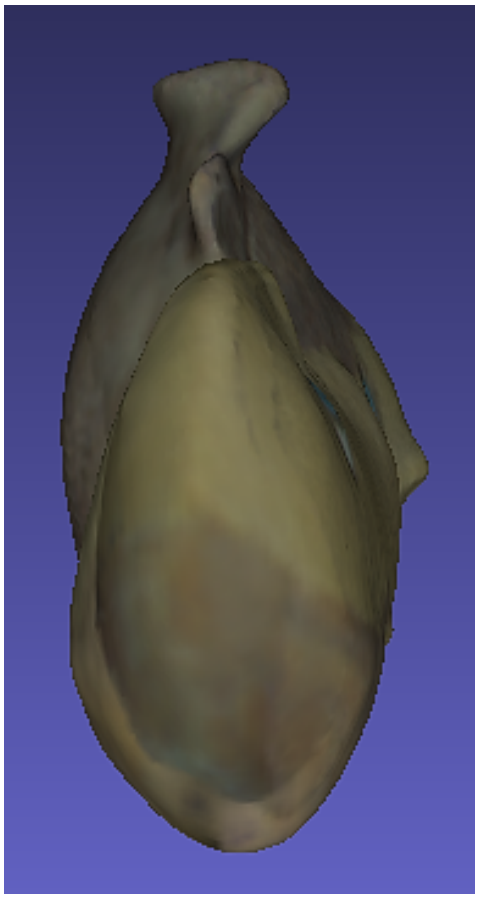
Matthew Ayres
28394916
CS 117 Final Project Report

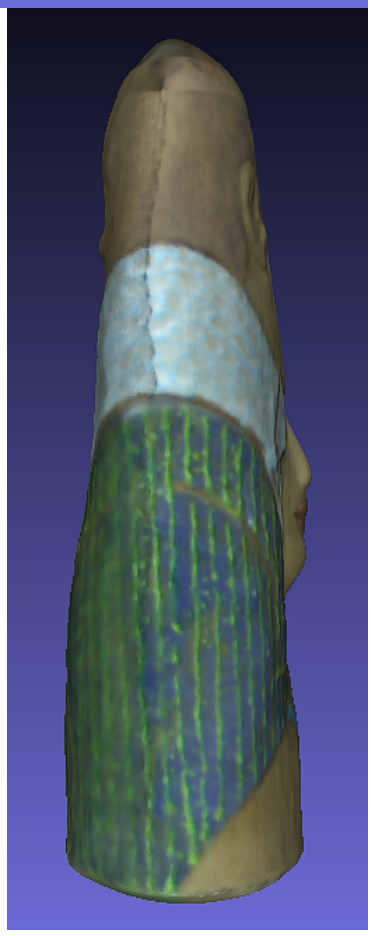
All meshes after alignment before running Poisson reconstruction in MeshLab



Matthew Ayres
28394916
CS 117 Final Project Report

Model previewed in MeshLab after running Poisson reconstruction from 4 different angles: left, right, front, above, and behind





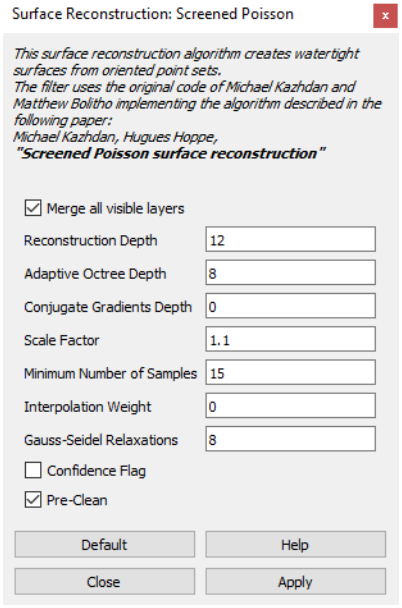
Matthew Ayres
28394916
CS 117 Final Project Report

Assessment and Evaluation

Overall I am decently satisfied with my results, however the final product does leave room for improvement. The most difficult part was finding the best parameters for the mesh generation function. In my testing I would often end up with a mesh that has holes or have awkward edges extruding from the mesh. In the combined meshes after alignment there were awkward gaps that affected the final result after poisson reconstruction. This is most noticeable in the top head of the object in which the gap created a lumped looking final mesh. Other than gaps, main areas of issues were from the curved and varying depth parts of the object by the nose and hair. These areas proved to be difficult on the algorithm for reconstruction and mesh generation. However in the final result they look a lot better than the raw mesh generation before poisson reconstruction. Fine-tuning the mesh generation came from finding the best fighting parameters for bounding box pruning and the optimal length for triangle pruning.
At some points in the alignment process MeshLab was very uncooperative, in my head I can visualize where I believe the points should align given points, however meshlab would have its own idea of where that is. Often moving the mesh to an unreasonable location and angle. Sometimes getting 2 meshes to align would take multiple tries before coming to a result that looked accurate.Another issue that appears in the final result is the texture of mesh has scars from the edges of the uncombined mesh visible. These are remnants of the coloring being different from the uncombined meshes. When combined, there is an outline where you can tell that the colors are slight variations of each other rather than the same. One way to improve the final result would have been if I was given more camera angles rotating around the object at a more consistent degree. This would create more individual meshes to align, but overall more data to work with when trying to create a perfect recreation. Given the time and equipment this would have been possible to do, creating my own image sets, everything would have been able to be tailored to my specific preferences.

Appendix

- Parameters used in MeshLab’s Surface Reconstruction Screened Poisson



- Calibration of intrinsic parameters utilized instructor-provided file and function on images from instructor-provided calibration directory.
- Calibration of extrinsic parameters using a combination of findChessboardCorners function from OpenCV and calibratePose function from instructor-provided code to set intrinsic and extrinsic camera parameters.
- Function reconstructColor is a modified version of the reconstruction function written from assignment 4.
    - Added computing of object mask by taking the difference of the foreground object and the background in which the difference is the object. This is added to the decoding mask.
    - Added code to store the RGB values in a 3xN array.
    - Added saving results to an output file .
- Function meshGen is a modified version of the code from the meshing section from assignment 4.
    - Modified code to take user inputted parameters.
    - Added code about remapping indices as described in lecture 12.
    - Added code for mesh smoothing utilizing a created get_neighbors moving points to their mean of their neighbors, inspired from a piazza post and based on topics discussed in a stackoverflow thread.
    - Added saving results to an output file.
- Created code to run reconstructColor using input parameters saving all results to be used in finding the best parameters for meshGen.
- Created code to individually test boundingbox limits and triangle pruning threshold in order to find the best parameters.
- Utilized instructor-provided writeply function to save all meshGen data to a file format viewable in MeshLab.