Daniel Diaz

74393336

6/10/2015

Final Report

Project Overview:

My project consists of creating a pipeline for creating 3D models from 2d scans using structured light. In the following paragraphs I will enumerate the steps involved in the pipeline.

The first step of the pipeline is to use an external resource called the Camera Calibration Toolbox. To use this resource we first must navigate to the folder containing the calibration images, then load the toolbox onto our path. These images consist of a planar checkerboard pattern arranged at different orientations in the space that will be scanned. Using this toolbox and the instructions located on the website (http://www.vision.caltech.edu/bouguetj/calib_doc/) / (http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html) we will be able to extract the intrinsic and extrinsic camera parameters. It is important to realize that our Rotation and Translation vectors are in the form of OM and T so we must convert OM into a proper rotation matrix by calling the Rodrigues function. After this, we will have all the necessary camera parameters needed to run the next step of the pipeline (focal length, center, translation, and rotation of the left and right cameras).

The next step of the pipeline is to reconstruct the 3D coordinates from the sets of images. To do this we will run “diaz_reconstruct.m”. This will load our stereo calibration results

(teapot/calib/stereo_calib_results.mat). Note in my implementation I hard coded the camera calibrations in the diaz_reconstruct file. This file is based on the work down in the previous assignment. It consists of calling decode on the scans to decode the horizontal and vertical images for each camera. This works by first creating an array containing the pixel coordinates, then identifying pixels which have both good horizontal and vertical codes, then pulling out the x,y coordinates of those matched pixels. Once we have corresponding pixels we can use triangulate to get the 3D coordinates for the set of pixels by using our left and right pixels and our left and right camera parameters. We can view our temporary results by calling plot3.

In our pipeline script we loop through each set of images, and save the resulting .mat file containing the variables X, xColor, xL, xR, camL, camR and scandir to reconstructions/scandata.

The next step in our pipeline is to generate the relevant meshes from the 3D point clouds saved in our .mat files. To do this we will need to call diaz_mesh on all the triangulated point clouds.

The function diaz_mesh takes the iteration number, and the reconstruction directory. Within this function we load the reconstruction variables saved by diaz_reconstuct. Our first step is to remove points outside a known bounding box. I went with (-150,30) to (-100,100) to (550,800). Without this bounding box, we may incorrectly compute triangles for points we don't care about or accidently increase our compute time. We then remove triangles with long edges and apply some simple smoothing (nbr_smooth).

We can visualize our results by calling trisurf, but more importantly we will save the generated meshes to meshes/meshdata in a .mat format

The last matlab step is to convert the meshes to a ply file. To do this we loop through each matlab file generated by our diaz_mesh function, and call a function called pointCloud2mesh on the inverted matrix Y, which was the resulting matrix after the smoothing. Then we call makePly and save it to teapot/plyFiles/set.

We need our meshes in a ply format so we can you an external application called meshlab. We import the meshes into meshlab and manually align them by selecting corresponding points. Once we have a pretty close estimation of the final model, we can call poisson reconstruction from within meshlab to create a watertight model.

After generating the model we can save it as a ply file and open it in any 3d modeling software to generate a flyover video. For this project I used Blender.

After running the camera calibration, then the custom pipeline, then aligning the meshes and performing a poisson reconstruction, we will have a final 3d reconstructed model from 2D scans.

Data Sets:

For the datasets, I used the teapot images.

I have included an image of the folder structure as I used in in the pipeline.

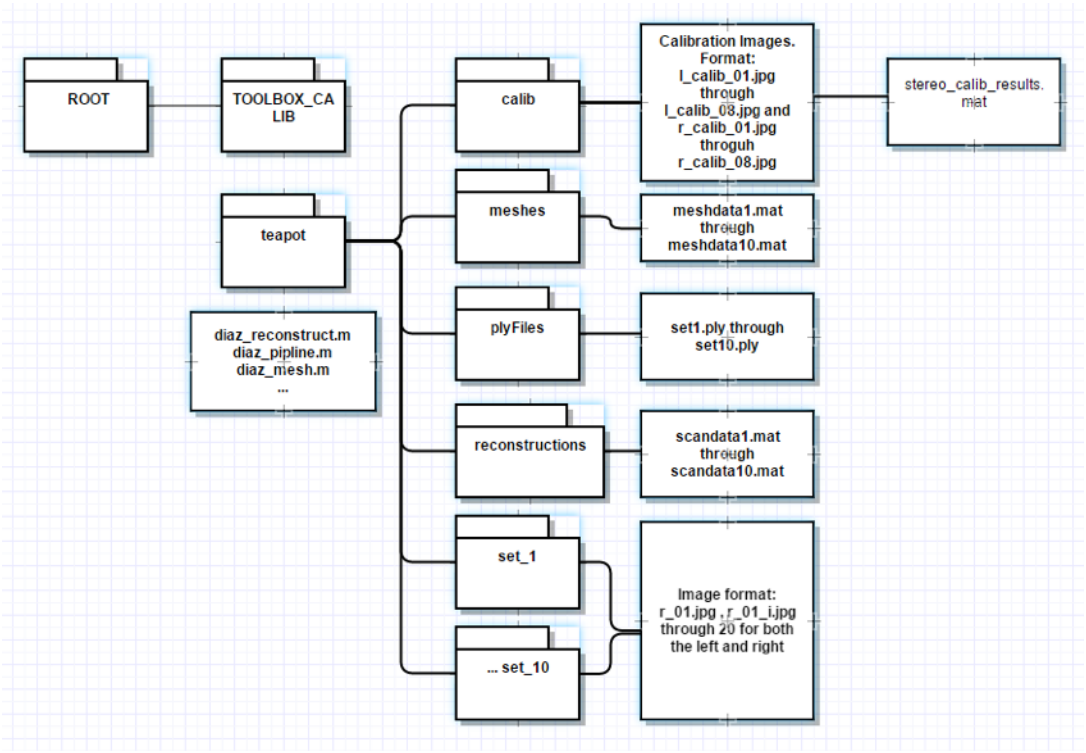
The l_calib_01.jpg for example is an image of a checkerboard pattern; With each image being at a different orientation. The l and r specify whether it was the left or right camera that took the photo.

The meshes folder contains the mesh data files generated by diaz_mesh.m

The reconstructions folder contains the scan data files from the diaz_reconstruct.m

The “sets” contain images with the teapot at different orientations and with the structured light illuminated on its surface. The i after the number is used to indicate when the inverse pattern is displayed.

Folder Structure:



Algorithms:

For this project needed to implement Mesh Cleaning and Mesh Alignment.

Mesh Cleaning consisted of several techniques for cleaning up the scanned data including threshold distances, user assisted pruning, hole filling, and smoothing. Mesh alignment used user input to give an initial estimate of alignment and then use the ICP algorithm to do the fine tuning of the alignment in meshlab. Also when combining the meshes, I used the Poisson Reconstruction available in meshlab.

ICP, also known as Iterative Closest Point, is a method for aligning 3D data. As described in lecture 9 of Professor Fowlkes lectures it involves 3 main steps:

-Estimating Corresponding points.

-Aligning points by computing R,T which brings points closet together (minimize the mean squared error). The optimal t is chosen so as to align the center of masses of the two sets of points. The optimal R can be found by computing the eigenvectors of a 3x3 matrix constructed from the point coordinates

-Repeat until convergence.

ICP was very important when it came to mesh cleanup.

Poisson Reconstruction is another algorithm that was pivotal in this project. As mentioned in Professor Fowlkes lecture slides, the main idea behind Poisson Reconstruction is the idea that the normal vectors on a surface are the same as the gradient of an implicit function F defining the surface. This algorithm is important when combining the meshes as it gave us a "water tight" 3D model.

Results:

Camera Calibration:

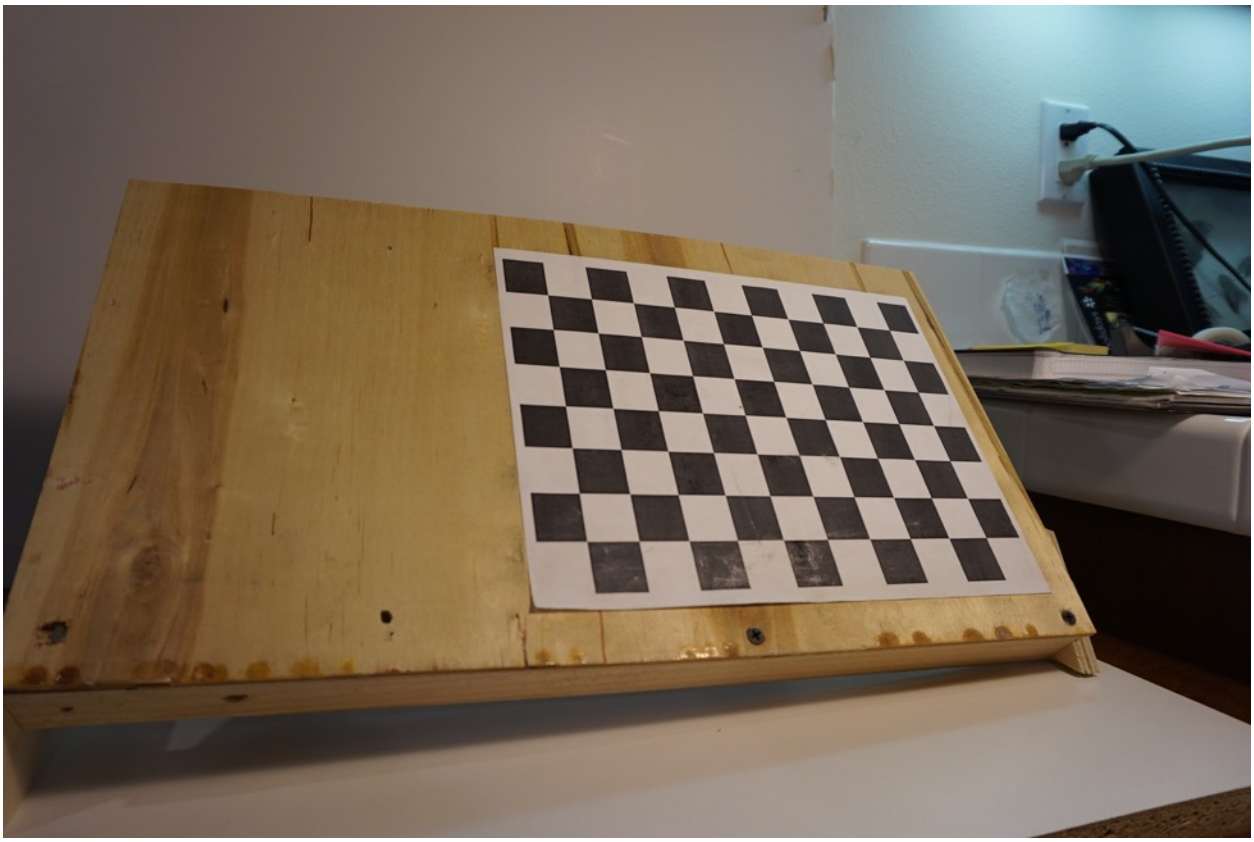
camL.f = 1856.90593; camL.c = [1403.64475 869.11654]; camL.t = [0 0 0]'; camL.R = eye(3);

camR.f = 1844.49359; camR.c = [1376.02168 942.95776]; camR.t = [-438.85342 -5.70884 188.20991]';

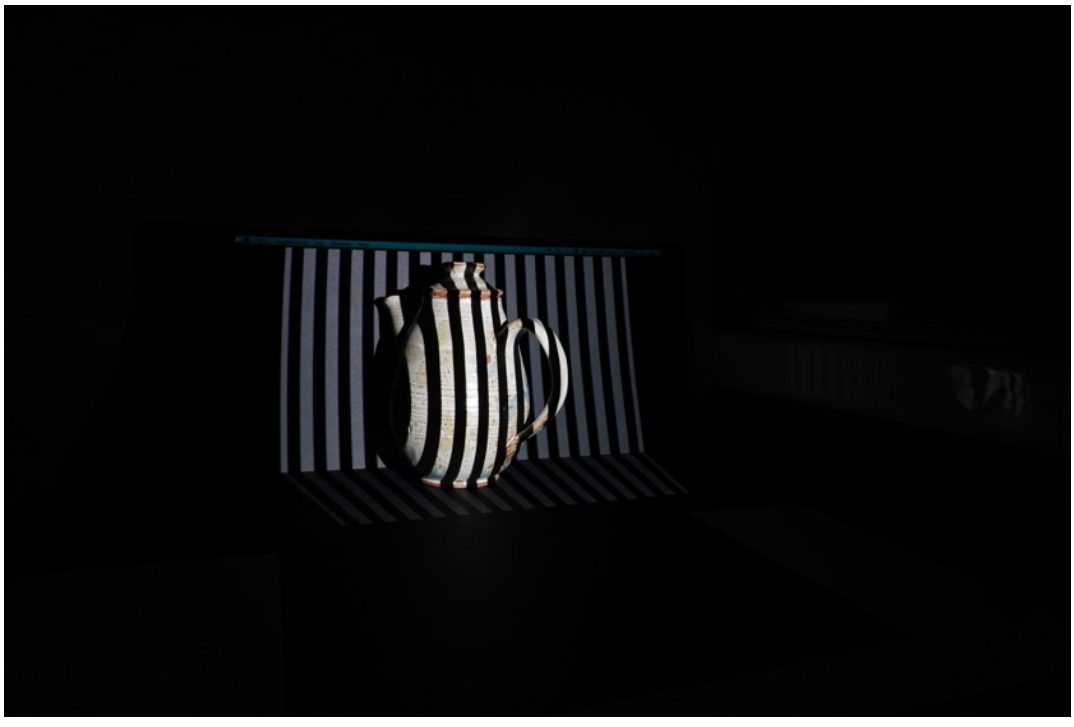
camR.R = [0.7273 0.0027 0.6863;
0.0258 0.9992 -0.0313;

-0.6858 0.0405 0.7266;];

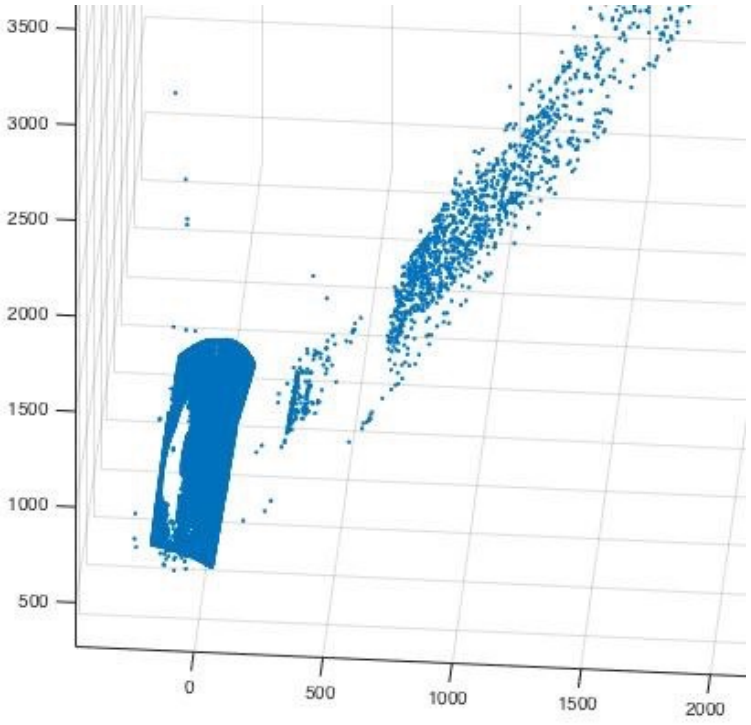
[Photo of calibration image]

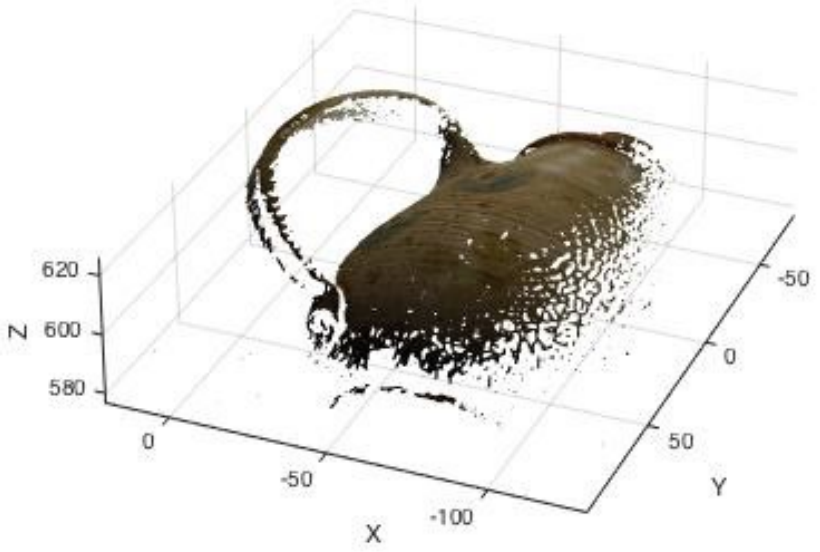


[photo of teapot scan]

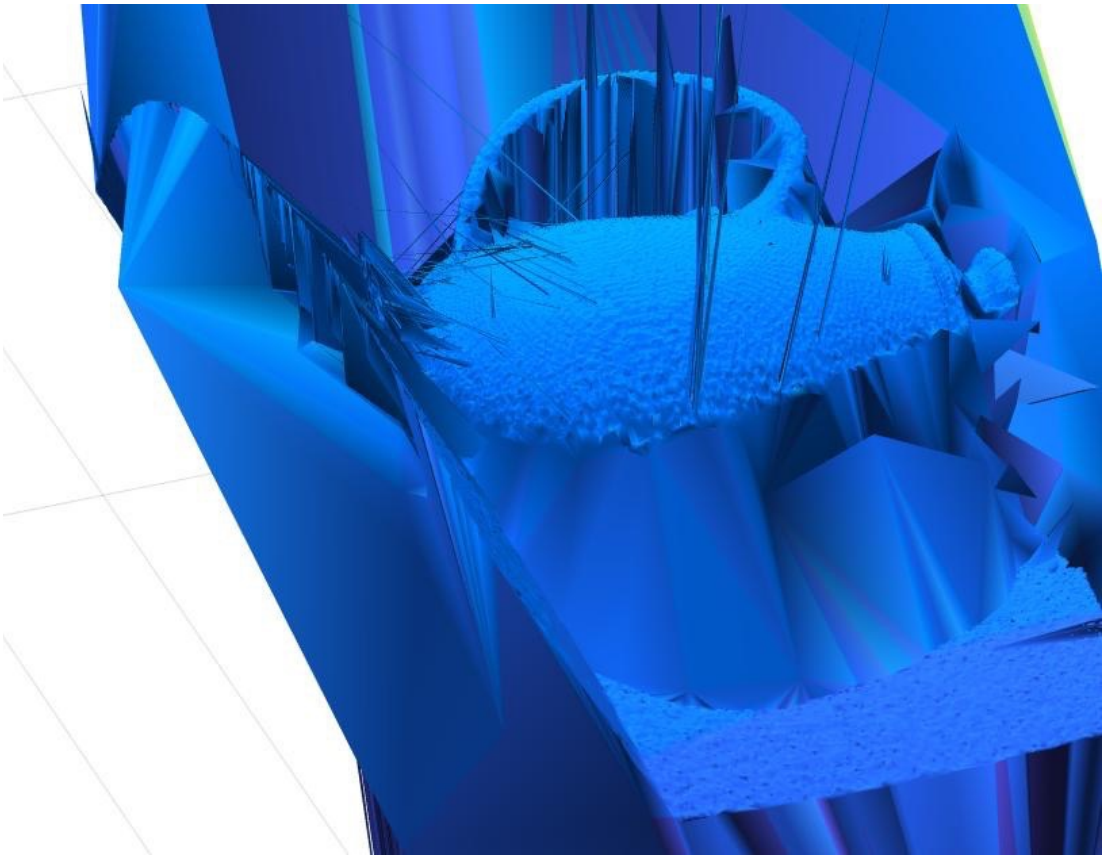


[photo of 3d points with view3d]

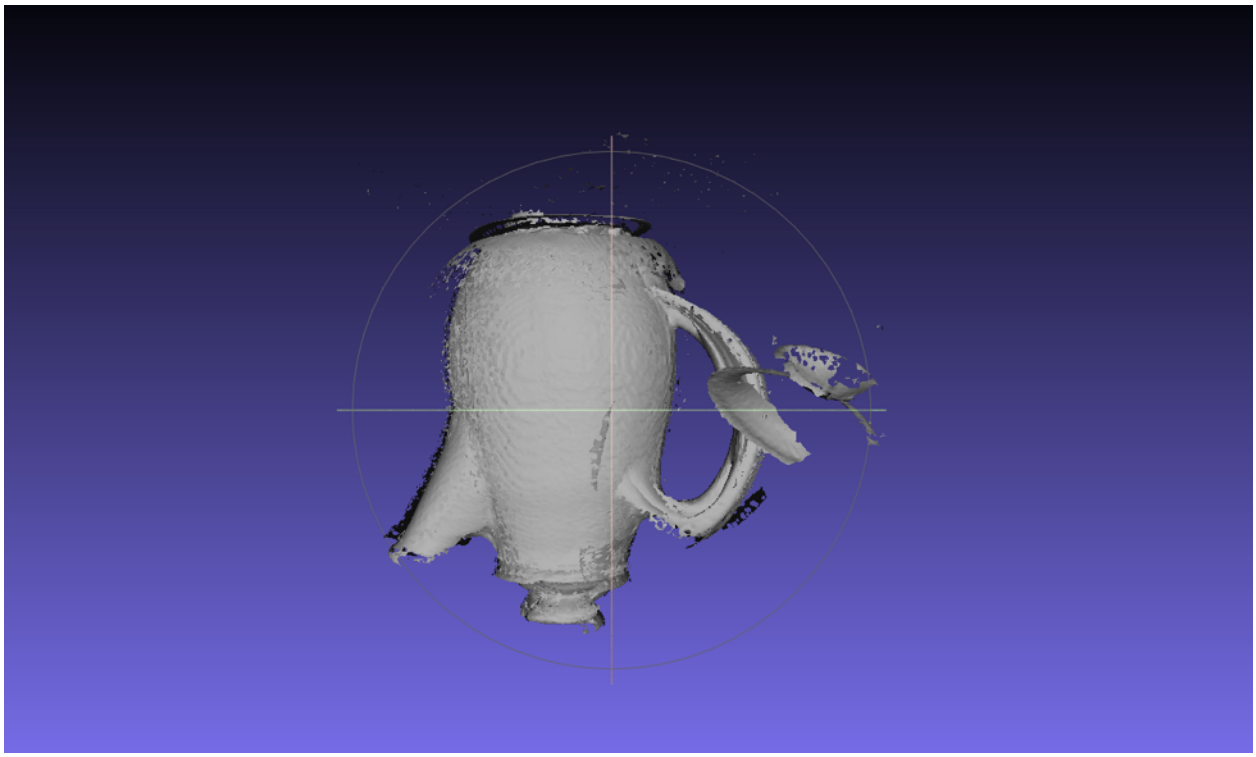


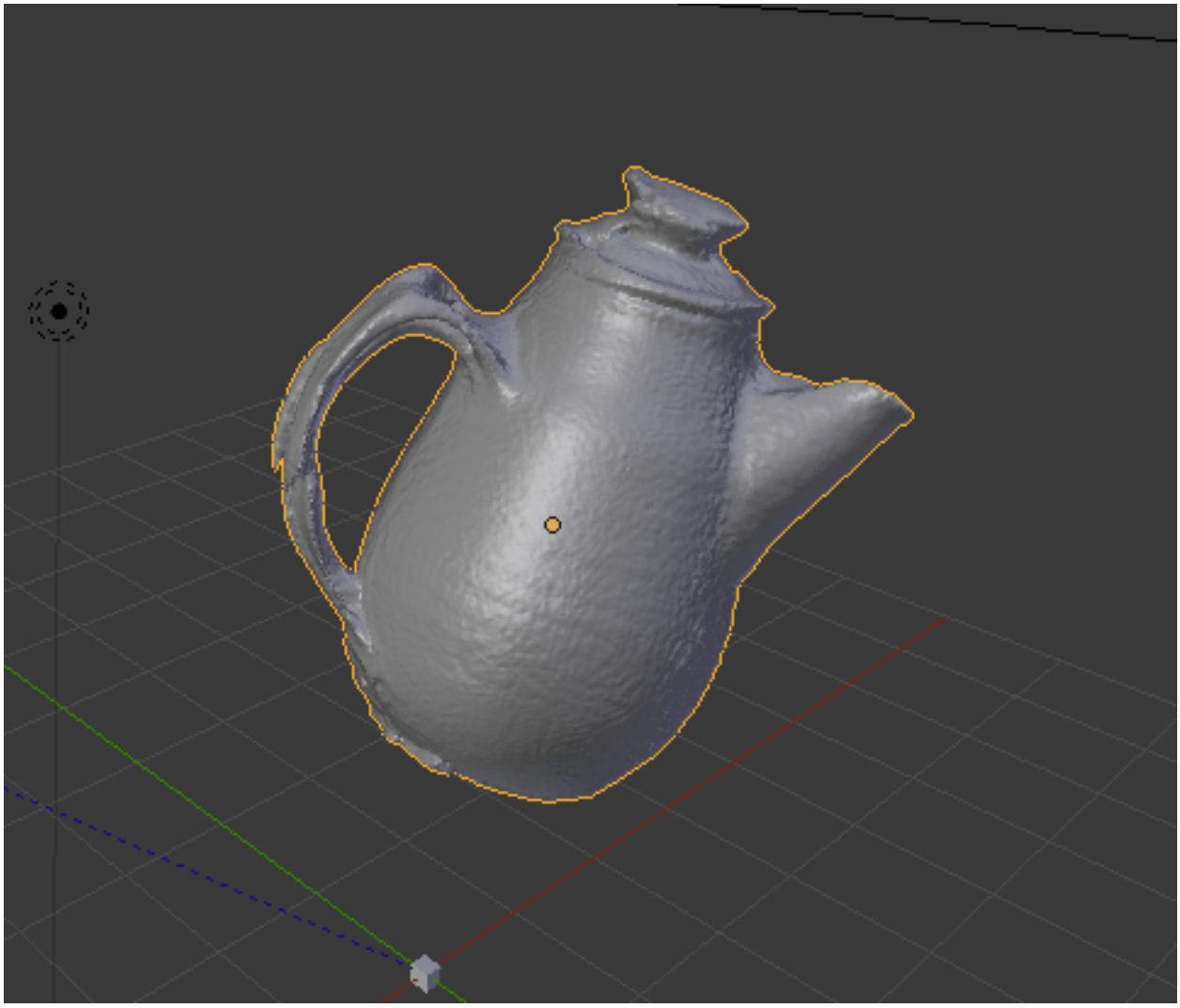


[picture of mesh uncleaned]



[picture inside meshlab]





[Photo of final reconstruction in blender]

[photo of actual object]

Note: I have also generated a 3d flyover. It will be located in the zipped file that is turned in.

Assessment and Evaluation:

A majority of this project built upon previous assignments which made the initial hurdle not as daunting as it may have been otherwise. Some points where there could have been some improvement were the automation of my own pipeline. When developing the final meshes I would just run each step incrementally to check the results are somewhere within the realm of possibility. If a different object were to be used with this pipeline there would some issues because my pipeline uses a hardcoded camera calibration as well as a tightly defined bounding box. The advantage of that bounding box being faster computation but the disadvantage being it may break the pipeline if you were trying to use it with a different set of images.

One limitation of the data, was that because the teapot was smooth it made aligning meshes rather difficult. With a smooth surface it's hard to find corresponding points when there are not that many landmarks to reference. Given another change, more sets of photos

would have created more meshes to align, but those meshes would have been easier if the all included the handle or other identifying feature in them.

Also another issue with the alignment for the teapot was that one mesh represent the bottom of the teapot, wich was extremely hard to match up. Given another chance it would have been more helpful to split that mesh into its corresponding bottom and top. Instead I just let the part of the mesh that represented the top of the teapot be cut off by the reconstruction because it was hanging of the model. In the ned the poisson reconstruction did a surprisingly good job and smoothing over the top of the teapot even though I did not place a mesh there.

At the end of the day I am satisfied with my results. I was able to import them into blender and create a 3D flyover video. The fly over looked pretty good, but the exact path was hard to nail down because I had some trouble with the blender controls.

- **Appendix: Software.** List the main MATLAB functions (plus code in any other languages) that you wrote for the project. Clearly indicate whether each function is either
  - written entirely from "scratch" by you for the project
    - diaz_pipeline.m
    - demoscript.m
  - a modified version of a function you wrote earlier for an assignment
    - diaz_reconstruct.m
  - a copy of a function provided by the instructor (or a modified version of the instructor's code)
    - diaz_mesh.m

        - decode.m
        - mapnearest.m
        - mesh.m
        - nbr_error.m
        - nbr_smooth.m
        - tri_error.m
        - triangulate.m
- code obtained from the Web or some other similar source.
        - TOOLBOX_calib
        - pointCLoud2mesh
        - makeply