CS117 Final Project

Jason Le

CS117

Professor Fowlkes

UC Irvine

Abstract

This document will contain a report on the final project for the course, CS 117 Project in Computer Vision. In this project, we will seek to reconstruct a 3D model based on scanned images of a teapot. This report will contain a project overview, data analysis, algorithm implementation, analysis of results, and an assessment and evaluation portion.

Project Overview

This project aims to create a high-quality 3D reconstruction of an object using structured light scans. The possible objects provided by Professor Fowlkes include a teapot, a manny, and a couple. This report will explore the 3D reconstruction of the teapot object. The goal is to create a detailed and accurate representation of the object by combining multiple scans obtained from calibrated cameras. By implementing various techniques and algorithms learned throughout this quarter of Machine Vision, the project seeks to overcome challenges in reconstruction such as noisy pixels, background interference, and misalignment between the scans. The outline of the objectives of the project includes calibrating cameras, modifying the decoding and reconstruction process, mesh generation, mesh smoothing, alignment of scans, Poisson surface reconstruction, visualization, and rendering. By successfully completing these project goals, the project aims to contribute to the field of 3D reconstruction and provide valuable insights into the challenges, techniques, and outcomes associated with generating accurate and detailed 3D models from structured light scans.

Data

The data used within this 3D reconstruction project of the teapot includes chessboard images for calibration and image scans of the teapot from different angles. To start off, the calibration images were contained in the folder “calib_jpg_u” which contains 40 images of a chessboard from various angles. There are two image prefixes “frame_C0” and “frame_C1”, where C0 is the left camera and C1 is the right camera. As seen in Figure 1, this is the image used for camera calibration. This image represents the position of the left camera. While in Figure 2, this is the image also used for camera calibration but for the right camera instead. The collection of these images is to correctly calibrate the camera to obtain accurate intrinsic camera parameters for the scanner camera. The image scans of the teapots are found in the “teapot” folder within the project directory. The “teapot” folder contains seven folders named “grab_0_u” through “grab_6_u” which all contain colored images and normal scans: horizontal and vertical. Colored scans provide information about the object’s surface texture, reflective properties, and color variations. This is useful in visualization due to the ability to note the color palette to assist in reconstruction. Each “grab” folder contains different images and colored scans based on the

Figure 1: “frame_C0_01.jpg” which is the first chessboard image used for calibrating the left camera

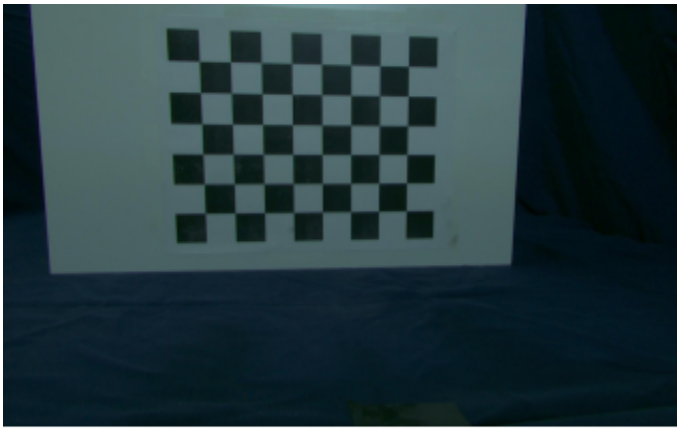

Figure 2: “frame_C1_01.jpg” which is the first chessboard image used for calibrating the right camera

Figure 3: "color_C0_01.png", a colored scan of the teapot lying down

orientation of the teapot. These pictures include the teapot sitting upright, lying down, and various sides of the teapot. Figure 3 is an example of one of the scans from the "grab_5_u" folder which contains images of the teapot lying down with the top facing the camera. These are the data sets that are used within this project in order to perform a 3D reconstruction.

Algorithm Implementation

There are many algorithms that are needed in order to perform a 3D reconstruction from the images provided. To start off the first algorithm used is camera calibration, which was provided by Professor Fowlkes. This script needs the input of the directory on your local machine where the calibration images are located in order to work. The function is in "calibrate.py" which utilizes OpenCV in order to calibrate the cameras. This algorithm is used to obtain the camera's intrinsic parameters that we need in order for the other steps.

The next algorithm implemented is a *decode* function. Previously, this function has been completed in other assignments but it did not include color masking. The decode function processes a set of images displaying a projected gray code pattern. It decodes the pattern to generate an array of decoded values and a binary mask indicating reliable decoding. We

```
im1 = read_image(colorprefix + "00.png")
im2 = read_image(colorprefix + "01.png")
colors = ones array of shape (h, w)
colorDiff = sum(square(im1 - im2), axis=-1)
thresholdMask = colorDiff > colorthreshold
colors = colors * thresholdMask
```

Figure 4: Pseudo Code for Modification to Decode

also compute a “colors” array in order to avoid triangulating parts of the image that are contained the background as is shown in Figure 4. The function compares pairs of images to recover the gray code bits and marks undecodable pixels based on a threshold. It produces binary images for each bit and converts them to a decimal image using binary-to-decimal conversion using logical XOR in order to do these conversions. The function provides decoded values and reliability information for subsequent 3D reconstruction steps.

Another algorithm that is implemented within this project is “reconstruction”. The *reconstruct* function performs a 3D reconstruction based on triangulating matched pairs of pointed encoded with a 20-bit gray code. It takes an input image for the left and right cameras, along with threshold values. The function utilizes the *decode* function to obtain decoded codes and masks. The masks are merged using this formula for the left and right cameras: C = H + 1024*V. Next, the function identifies the corresponding pixels between the left and right images using the “np.intersect1d” function. These coordinates are passed into the triangulate function to get the 3D coordinates of the points. Additionally, the function retrieves the RGB values of the corresponding pixels in the color images. It iterates over the matched points and extracts the RGB values from the color images. Then the arrays are averaged to get the final RGB values for each point.

The next algorithm implemented in this project is the mesh generation function named *meshGen*. The *meshGen* function takes the output of the *reconstruct* function, along with additional parameters such as thrithresh (threshold for max allowed edge length) and box limits (bounding box dimensions), to generate a triangulated mesh of the reconstructed scan. The function begins with pruning the points outside the specified box limits. Points that lie outside of the limits are removed from the variables: pts3, pts2L, pts2R, and RGB. Then, the points in

pts2L are used to perform Delaunay triangulation using the “scipy.spatial.Delaunay” function. This generates a list of triangles representing the surface mesh after calling “.simplices” after Delaunay triangulation. You will also need to improve the mesh quality by mesh smoothing and triangle pruning. The mesh smoothing technique utilized the method of iterating through each point in pts3 in order to calculate the mean of its neighboring points in the mesh and updates the points accordingly. This smoothing step helps to refine the surface representation. This implementation only includes smoothing the points once due to the possibility of over-smoothing the meshes. There is the option to smooth more than once depending on the quality of the meshes. After smoothening the meshes, triangle pruning is performed. The lengths of the triangle edges are calculated via Euclidean distance as shown in Figure 5. Triangles that have edges longer than the specified thrithresh value are pruned from the mesh. The remaining triangles are stored in the tri array. Then, the function removes any points that are not referenced by any triangle in the mesh to ensure that the final mesh only contains vertices that are part of the surface.

```
dist1 = np.sqrt(np.sum(np.power(pts3[:, tri[:,0]] - pts3[:, tri[:,1]], 2), axis=0))
dist2 = np.sqrt(np.sum(np.power(pts3[:, tri[:,0]] - pts3[:, tri[:,2]], 2), axis=0))
dist3 = np.sqrt(np.sum(np.power(pts3[:, tri[:,1]] - pts3[:, tri[:,2]], 2), axis=0))

pruning = np.all([dist1 < trithresh, dist2 < trithresh, dist3 < trithresh], axis=0)
tri = tri[pruning,:]
```

Figure 5: Pruning Code

Then we had to perform 3D alignment for the meshes and Poisson reconstruction in MeshLab. After generating 7 meshes to make a reconstruction, we will need to load the mesh.ply files into a program named MeshLab. This tool will help us combine the meshes into an aligned scan. Also, in MeshLab we can perform Poisson reconstruction which is a technique used for surface reconstruction from unorganized point clouds. This helps complete the mesh by filling out and smoothing gaps in the aligned mesh. MeshLab has a tool that allows for Poisson reconstruction to be done easily.

Results

The results of calibrating the cameras via the “calibrate.py” script were successful. The re-projection of the chessboard and the points all fit the corners of each checker on the board. As

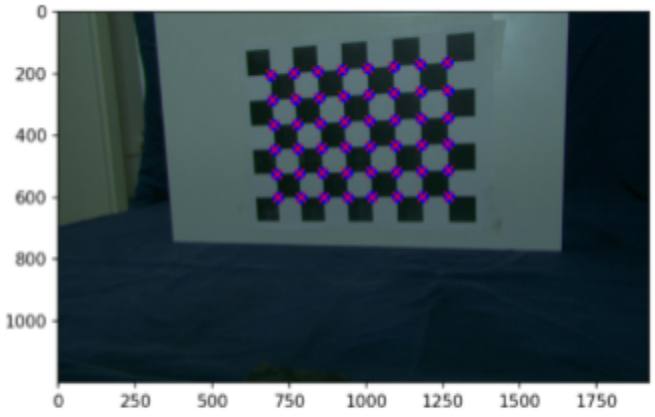

Figure 6: Calibration Results

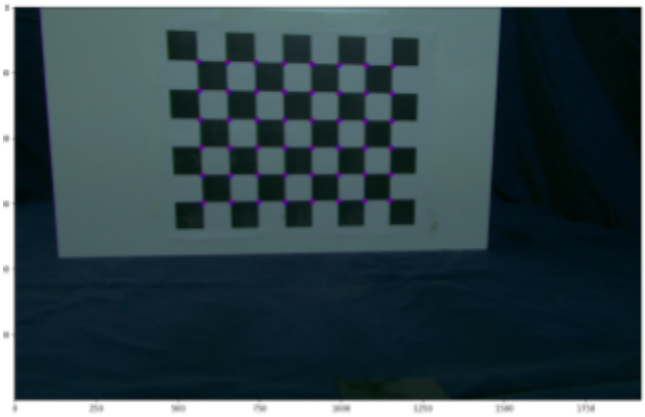

Figure 7: Calibration Results

shown in Figures 6 and 7, we can see the images of the calibration. The results were expected due to a previous assignment’s need for camera calibration.

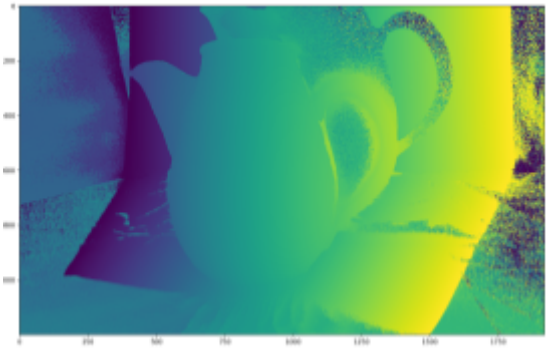

Figure 8: Code Results

Next, the results from the *decode* function. The returned images, code, mask, and colors, are plotted in Jupyter Notebook as shown in figures 8, 9, and 10. These images show successful results after calling decode.

Figure 9: Mask Results

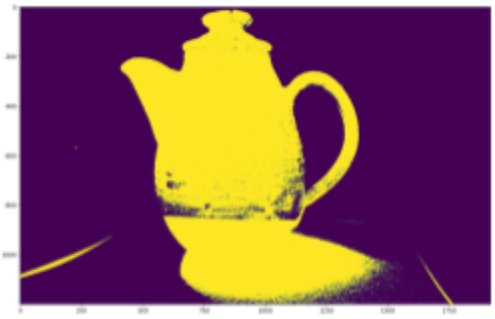

Figure 10: Colors Result

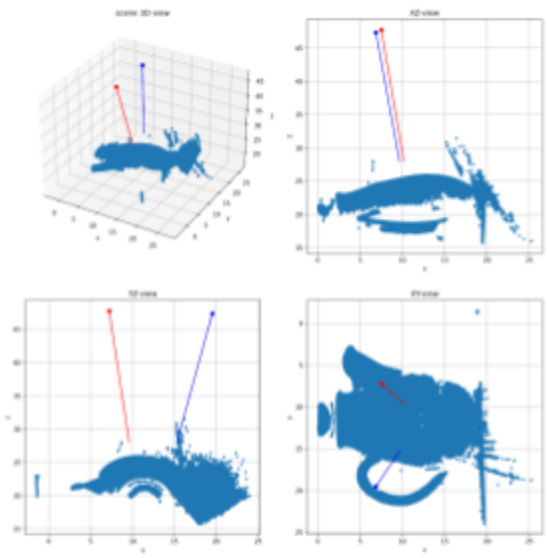

Figure 11: Reconstruct Visualization

The results of the *reconstruct* function are shown in Figure 12. This function provides the point of view from a 3D perspective, XZ-view, YZ-view, and XY-view. These reconstructions are a bit noisy but in mesh generation, these points will be removed in smoothing and pruning.

Figure 12: Meshes First Loaded into MeshLab

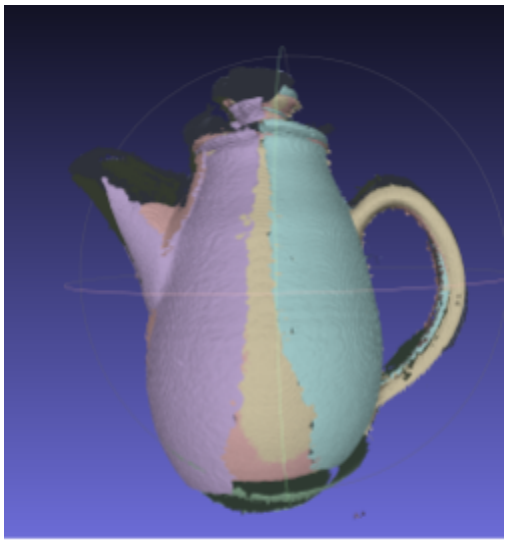

Figure 13: 3D Alignment of Meshes

After getting all of the 7 meshes, I put them in MeshLab in order to start aligning the meshes. Figure 12 shows the 7 meshes unaligned in MeshLab. Then, Figure 13 shows the attempt at aligning the mesh using point-based gluing. In Figure 14, the image shows the meshes combined and Poisson reconstructed to clean up and fill in gaps within the mesh. Figure 15 shows an image rendered in Blender which adds more detail to the teapot.

Figure 14: Poisson Reconstruction

Figure 15: Blender Visualization

Assessment and Evaluations

When reflecting back on completing the project whose goal is to reconstruct a 3D render of the teapot object based on the image scans, I did meet that goal and rendered a model of the teapot as shown in the Results section. The steps in order to reconstruct this image were very challenging and there are some areas where I could have gotten better results. When looking back at the algorithms I implemented, the results from calibration, decoding, and reconstructing were good. They performed as expected but in mesh generation and 3D alignment, the accuracy of the results was not the best. When looking at the data given such as the scans of the teapot, there were some lighting differences in the picture which would have attributed to noisy points and deficiencies in coloring. I encountered the most challenges when trying to align the meshes in MeshLab. The point-based gluing technique was very time-consuming which took me a long time to achieve a decent mesh. When looking at the mesh in Figure 13 it is evident, that the mesh is not perfectly aligned and there are some gaps within the mesh. Trying to align the meshes in MeshLab was the most difficult part when completing this project. It was hard to select the correct points so that the meshes were orientated correctly. I also used Blender to try to manually

Figure 16: Blender Alignment

move the meshes within a 3D space to achieve the best alignment as shown in Figure 16. This method proved beneficial giving me a more refined and cohesive mesh. While the mesh is still not perfect, it does attack some of the problems with point-based gluing. The results after the Poisson reconstruction were reasonably good. The image provided in Figure 15 does represent the teapot and has similar but darker color compared to the original scans. The deficiency in my final render could also be due to mesh generation leaving out some points during pruning or smoothing that would fill some of the holes in the mesh. I feel that the mesh alignment is my weakest link in this project. With a better mesh alignment, a better reconstruction would be produced. Provided more time, I could have kept refining my mesh either in MeshLab or Blender in order to produce better results. With more time at hand, I could have tested different threshold values for decoding, coloring, and trithresh to improve my results.

Conclusion

Overall, this project demonstrated the successful reconstruction of a 3D teapot model using structured light scans. The implemented algorithms such as camera calibration, decoding, and reconstruction return valid results. Although challenges were encountered during mesh alignment, the project provided valuable insights into the process of generating an accurate 3D model. Further improvements could be made in mesh alignment and testing threshold values for

enhanced results. This project showcased the subject of 3D reconstruction and the possibilities and challenges in reconstructing objects from scans.

Appendix

- Decode Function
    - This function is a modified function from the previous code I wrote from Assignment 4 to include a colored mask and improved from the code given by the professor.
- Reconstruct Function
    - This function is also a modified function from Assignment 4. I modified based on the base code given by the professor in “camutils.py” to add processing RGB values.
- Mesh Smoothing
    - This technique is written by me, based on lectures, and online resources.
- Mesh Generation Function
    - This function is written by based on the mesh generation code used in Assignment 4. But I modified it based on implementing mesh smoothing and keeping track of the RGB values and modifying them accordingly during pruning. I did use insight from lecture material for this function.
- Mesh Generation of Teapot Scans
    - This function written by me involves looping through each of the grab folders, 0 through 6, in order to generate .ply files to render in MeshLab or Blender.