Lessons Learned in 3D Scanning

A Final Project in Computer Vision

Pablo Kang

1. Introduction

When I first wrote my initial project proposal, back in the middle of May, I had originally intended for my project to be a test of the potential of 3D scanning by attempting to model a highly-detailed and complex object of my choosing. To that end, I chose to scan a Nintendo Bowser amiibo figure, a small figurine sold as a peripheral game accessory by Nintendo for their suite of console game titles. However, after looking at the photos that resulted from my scanning session with the class TA, I realized that this project I had planned was not going to be as easy as I thought it would.

2. Data Collection: Woes of Scanning

This section will explain background information on the process I took to collect my data sets.

2.1 The Fall of Bowser

I knew exactly what I wanted to do for my final project the moment the assignment was announced: I was going to scan and generate a 3D model of the little Bowser amiibo figure (shown below) that sits on the desk in my room every day. I had received the miniature king koopa as a present from my significant other, so I was quite excited about the prospect of involving it in such an interesting computer vision project. During my scheduled scanning session, a Tuesday, the TA and I worked hard to try and take adequate scans of the little guy, but we were immediately faced with several glaring issues that threatened to ruin my planned project. The first was that, despite his intimidating appearance and fierce posture, Bowser is quite a small as an amiibo figure. In fact, he doesn't even reach 4 inches in height. Because of the way the scanning equipment was set up, this meant that there was no way to take scans of Bowser as close as the photo in Figure 1 was taken. We had to place him fairly far away from the camera just so he would be covered by the structured light bars and visible in both cameras. The second issue we faced was that the webcams used in the scanning setup, while also not capable of producing images as sharp as the ones provided by the professor's sample scans, had their auto-focus setting enabled by default. Due to his Bowser's small size in comparison to the image plane of each photo taken,

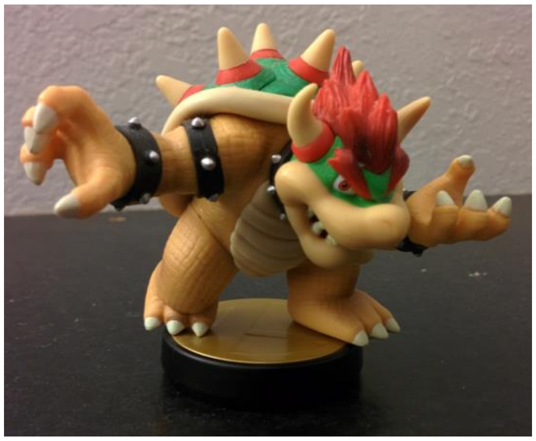


Figure 1. My Bowser amiibo figure.

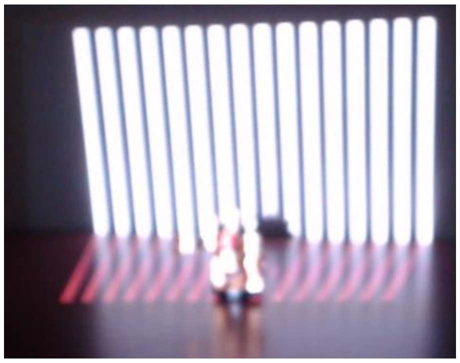


Figure 2. One of many useless scans.

as well as the rapid changes in light between shots, this caused most images to be blurry beyond possible use for 3D scanning, as seen in Figure 2. In the face of all these problems, I took the TA's counsel and decided I would come back with a larger object to scan. If only it had been that easy.

2.2 The Fall of the Lion

When I returned to the computer vision lab the following Monday, I was carrying the noble and fierce King of the Jungle in my arms. Or rather, I was carrying a statue of a lion, shown in Figure 3, which I had purchased on a trip to Mazatlán one summer long ago. Seeing as how he was easily as long as my forearm and three-times as tall as poor Bowser, I felt confident that I was going to get some great, high-quality scans that day. Since I had practiced during the previous scanning attempt, I volunteered to do the majority of the scanning work by myself, so that Phuc could enjoy a little downtime and finish his lunch in peace. After completing the scans, he and I gave them a quick look-over and were glad to see that all of the images were fairly sharp; no blurry blobs in sight.

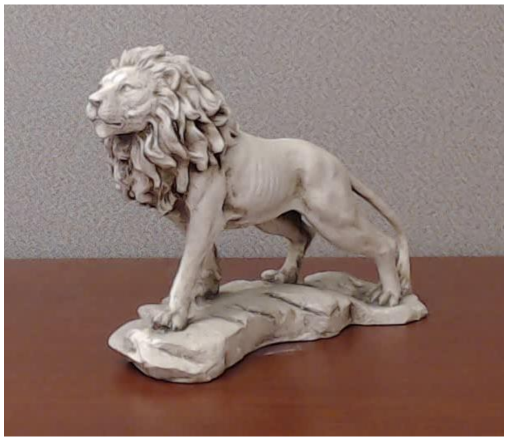


Figure 3. The lion of Mazatlán.

Unfortunately, what we didn't realize at the time was that, while the lion solved our size issue, it had brought an entirely new problem to my project: he was white. While that statement might sound racist when out of context, it was ultimately what forced my project to using the default monkey scans provided by Professor Fowlkes [1]. Simply put, the lion's white-gray tones were brightened to the point that they matched the illuminated wall behind him. This caused portions of his lit silhouette to become indistinguishable from the background, as evident in the left image of Figure 4.

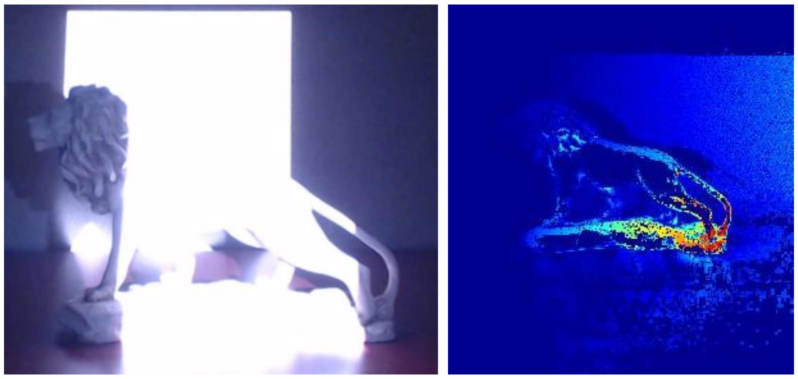


Figure 4. Left: An unfortunate background color caused the lion's back to become invisible. Right: R_h for set_04 shows an absence of lion pixels.

By this point in the project, I had finished writing the foundation of my pipeline and it was able to load and perform decoding and reconstruction on any range of image sets (provided calibration had been performed prior). What spat out of the reconstruction function, the right image in Figure 4, should have alerted me to the glaring fault that lay in my lion scans, but I assumed it was “just how it was supposed to look”. I told myself that I would wait until I had integrated and customized the professor’s mesh.m function from Homework 4 into my pipeline, so I could see the actual mesh that resulted from my scans, before making any judgements about my lion scans. Unfortunately, because so much of the lion was essentially invisible to the scanning pipeline due to the issue mentioned earlier, the first mesh to be generated, shown in figure 5, sealed my projects fate. In the interest of using what time I had left for the project, I decided to abandon my plan to use my own object and modified my project’s setup function to use the professor’s monkey scans [1].

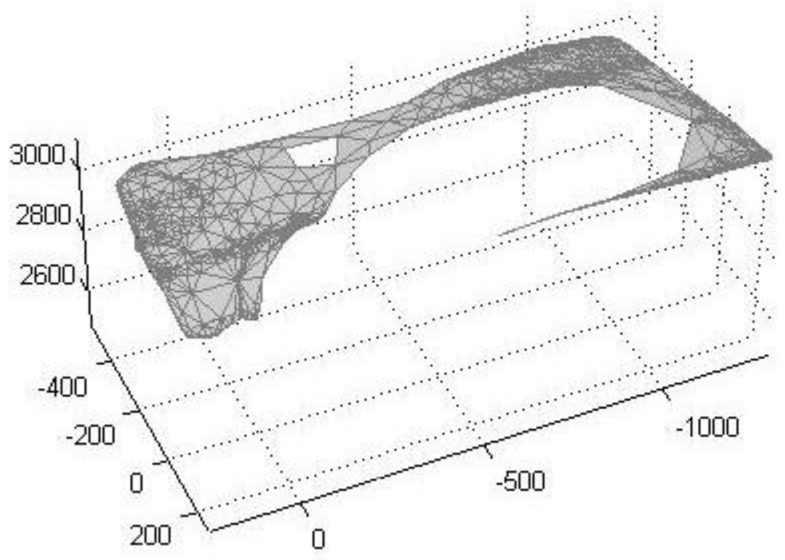


Figure 5. The mesh for set_01, showing a notable lack of lion, but plenty of background.

3. Scan Data Processing

In this section, we will discuss the major algorithms used in the mesh-generation pipeline.

3.1 Calibration

Calibration of the lion and monkey scan data sets was done using the Camera Calibration Toolbox, provided for public use by Caltech [2]. The toolbox is comprised of a set of MATLAB functions and scripts that perform various useful calibration algorithms. The two functions I used during my project’s calibration phase were calib_gui() and stereo_gui(), both of which provide an easy-to-use GUI interface to the user. The first, calib_gui() serves to perform calibration on one specific camera at a time, allowing users to perform input-assisted corner detection and outputting mat files containing all the information calculated from the calibration process. The second, stereo_gui() takes the left and right camera calibration data sets produced by calib_gui() and assembles them into one stereo calibration file, which I then used in my pipeline.

3.2 The Pipeline: Demoscript

While the majority of my pipeline's core code came from the Homework 4 functions provided to us by Professor Fowlkes [3], it took a lot of adding, tweaking and recoding before I was satisfied with their integration into my pipeline. I started my pipeline by first designing setup(), a means of allowing a user to easily modify the settings of the pipeline so they could readily alter things like the source directory for scans, the destination directory for data results, and which range of sets to work on. Initially, setup() featured lengthy prompts and user input for the string names of directories and regex formats, but that quickly proved to be too tedious to deal with during testing, so I gave setup() a default set of settings variables that can be modified by simple opening the setup.m file and editing their values. These preset settings greatly improved the generality of my code, as it made it extremely easy to switch from one set of scans, such as *lion*, to another, such as *monkey*, without having to go through several function files to modify file paths and variable names. After setup() is finished, it displays the current saved settings to the user via the console so they may be aware of what the pipeline expects its file structure to be like .

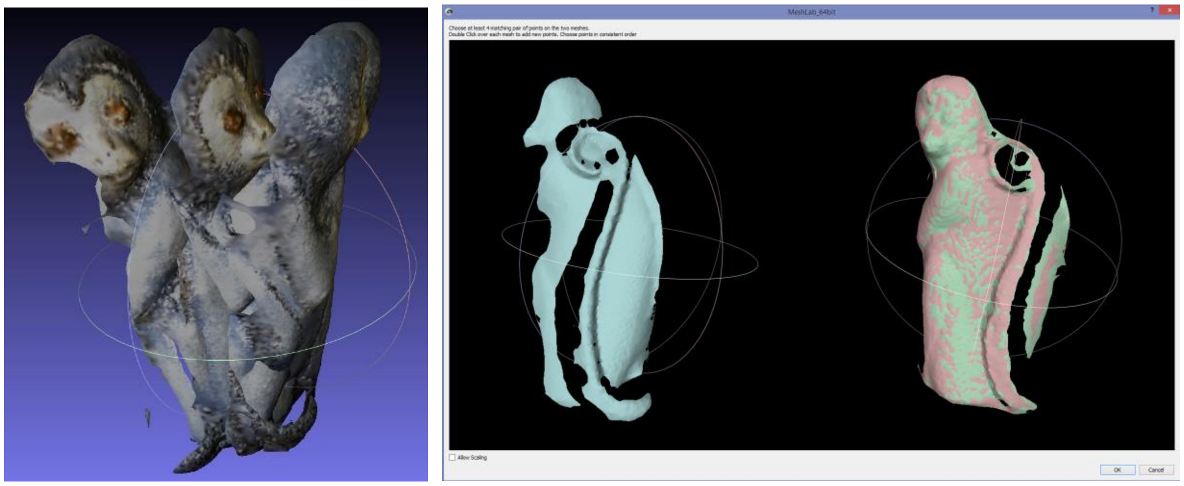
Next, I designed the pipeline that would call setup() and all of the other functions of the mesh-generation process, which I named demoscript.m (pseudocode in Figure 6). Due to the fact that I had heavily modified them to work in a more automated and cooperative fashion within the pipeline, reconstruct() and make_mesh() (originally named mesh() by the professor) saved their processed results to a shared data folder as mat files. This, for example, allows reconstruct() to be unnecessary if the scandata_0n.mat files it produces were filled on a previous run (assuming the data set has not changed). To keep with my theme of user-assisted automation and efficiency, I decided to implement a rather simple system of user confirmation before any of the three major steps (reconstruction, mesh creation and ply file saving) are executed. The pipeline script prompts the user at each step if they would like to continue or skip to the next step, and does so according to their answer. Like I said, nothing too fancy.

```
Function: demoscript.m
    settings = setup()
    if user wants to perform reconstruction
        for i = settings.setStart to settings.setFinish
            reconstruct(i)
    if user wants to (re)generate meshes
        for i = settings.setStart to settings.setFinish
            make_mesh(i)
    if user wants to (re)save meshes as ply files
        for i = settings.setStart to settings.setFinish
            Load meshData mat from file
            ply_data = tri_mesh_to_ply(meshData.Y, meshData.xColor, meshData.tri')
            ply_write(ply_data,meshPath,'ascii')
```

Figure 6. The pseudocode for my pipeline. Not shown is how data at each step is saved.

5. Mesh Processing

After my pipeline finished generating all seven meshes for the monkey figurine (shown unaligned in Figure 7), I imported them all into MeshLab, an open source piece of software meant for the processing and editing of 3D triangular meshes [4]. After figuring out how MeshLab worked, thanks to Mister P.'s MeshLab Tutorials on YouTube [5], I was able to use the alignment tool to transform each of the 7 meshes so that they were aligned to form the loose, unmerged shape of the final monkey model. The alignment tool first asks users to give it an initial guestimate for the alignment by asking them to pick 4 or more pairs



Choose at least 4 matching pair of points on the two meshes.
Double Click over each mesh to add new points. Choose points in consistent order

Figure 7. Left: The homunculus that is the 7 monkey meshes unaligned. Right: The alignment tool as I align the blue mesh to the already-aligned red and green meshes.

of corresponding points between two sets of meshes, a process shown in Figure 7 above. This allows the tool to align the two sets based on this "best guess" using the Iterative Closest Point (ICP) algorithm, whose pseudocode is shown in Figure 8. From the best guess, it iteratively adjusts the transformation matrices relating the two sets of meshes until it reaches the point of closest correspondence between the two.

```
ICP Algorithm | Input: Meshes M and N. Output: Rotation R and Translation T.
1) Start with initial guess.
2) Iterate
    a) For each point in M, find closest point in N.
    b) Find best transformation for this correspondence.
    c) Transform M.
```

Figure 8. The pseudocode for the Iterative Closest Point (ICP) algorithm.

Once all seven meshes were aligned with one another, I flattened them down into one mesh that had the rough appearance of the original monkey, which can be seen in Figure 8.

6. Evaluation of Results & Conclusion

In order to make the final model a bit easier on the eyes, I chose to smoothen it down and remove the rough mesh artifacts left over from the previous meshes by running the Poisson Reconstruction function from the Filters drop-down menu. This algorithm uses Poisson surface reconstruction on a triangular 3D mesh, considering all oriented vertices at once, to smoothen and reshape it. Poisson reconstruction also has the added benefit of automatically filling any holes. This resulted in the final model, shown in figure 9. Unfortunately, Poisson Reconstruction does not account for color values at vertices, so the final model was devoid of color. Despite this and all of the other hardships I endured, I am satisfied with this project and its results.

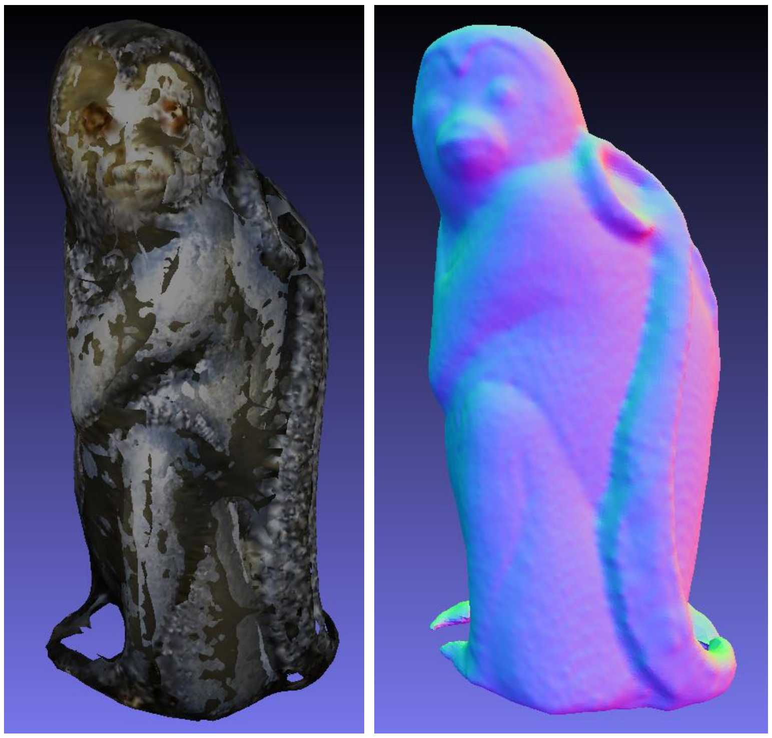


Figure 9. Left: Newly merged meshes with color. Right: Smoothened final model.

References

[1] Monkey Scan Data, Professor Fowlkes, http://www.ics.uci.edu/~fowlkes/class/cs117/data/

[2] Camera Calibration Toolbox, Caltech, http://www.vision.caltech.edu/bouguetj/calib_doc/

[3] Homework 4 Functions, Professor Fowlkes, http://www.ics.uci.edu/~fowlkes/class/cs117/code/project/

[4] MeshLab, Visual Computing Lab, http://meshlab.sourceforge.net/

[5] Mister P. MeshLab Tutorials, YouTube, https://www.youtube.com/user/MrPMeshLabTutorials

Appendix: Software

I wrote setup(), MakeModel() and Demoscript().