3D Reconstruction for Smartphones

Trevor Macks
ECE Department
University of Alabama
Tuscaloosa, AL
tfmacks@crimson.ua.edu

Yuxin Zhuang
ECE Department
University of Alabama
Tuscaloosa, Al
yzhuang7@crimson.ua.edu

3D reconstruction from a dataset of 2D images is not a new or revolutionary concept. It has been done before, using a plethora of different algorithms and techniques. This paper proposes an approach centered around a simplified physical capture environment for the collection of the 2D image dataset. Through the simplification of setup, several common design parameters can be reduced or removed from the processing stage of the 3D scanning pipeline. This allows for a faster system with similar reliability to other current implementations of 3D scanning. Overall, the project accomplished the implementation and integration of dataset import functions, 2 background removal functions, and feature detection and matching functions. If the project were to be pursued further, the next steps would involve structure estimation and depth mapping of the dataset images.

I. Introduction

Currently the 2 primary approaches for 3D model generation of a tabletop object involve using IR sensors (e.g. time of flight measurements) or camera based image capture. Out of the processes and approaches found for 2D image capture to 3D model construction, the majority are focused on real time scanning applications or simply a mobile camera with static object. To reduce the computations involved in having a nonstatic camera, this paper proposes a static camera and rotating object capture environment. This allows the later computations to use already known data about camera angle, position, and distance from the object. Once the images are captured, they are processed through the feature detection algorithm Speeded-Up Robust Features (SURF), matched together, and then processed further to generate a structure and depth for the model.

II. Physical Capture Environment

The physical capture environment is the centerpiece of what sets this reconstruction pipeline apart from other published research in the subject. Multiple variables present in non-static camera applications, such as real time 3D scanning, can be removed from the processing stage as known values. These variables include camera angle, distance, and position, object rotation angle, and the background. An ideal setup would consist of a rotating turntable with an object centered on it, camera at a 30° angle above it, and image capture every 3°. Due to funding limitations and delays in an associate's progress, the physical capture environment had to be simplified further for algorithm development and to provide a base dataset for testing.

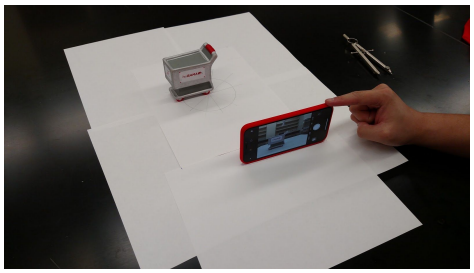

Figure 1: Physical Capture Setup

In Figure 1, the simplified physical capture setup is shown. A circle was drawn on a white background, with angles marked every 30°. The object was then centered on the circle and rotated for 90°, with images being captured every 3°. The camera remained in a fixed location during this process, with no change in angle or height. The lighting was selected to be bright and diffused, which helped minimize the shadows on and around the object.

Once a dataset is captured, it is sent to a function to import it into the MATLAB workspace. This function was coded to be robust to datasets of varying sizes and self terminates once all of the images are read in.

III. Background Removal Functions

In order to only capture relevant foreground feature vectors and descriptors, the background of the images must be removed first. Two different approaches to this problem were tried, with each having its own unique benefits. The first approach relies on eigenvectors, and the second approach is dependent on the background color.

A. *Eigen-Value Based Background Removal*

In most cases, capturing an object with a smartphone camera may not have a perfect background environment. In this case, isolating the target object from the background is challenging. For example, the image's quality may be affected by lack of proper illumination due to how today's phone camera is not designed to handle image capture under low light. Another significant factor on producing a satisfactory result is the presence of an intricate background. For instance, if the object and the background have similar coloration or the background has a complex design, software may sort these regions into one group, thus resulting in a main object with background pixels still attached and holes in it due to misclassification. In this case, the Eigen Based Background Removal function is an excellent candidate. The Eigen function takes input from the user to let them select which part of the image is unneeded, and then uses multiple functions and matrices to remove the background. This process provides high DOF and works exactly on giving the desired result to the user. However, the Eigen function needing user input to select background reference points is a major downside, and greatly slows down the project's processing speed . In order to output decent images, the Eigen function has to ask the user to provide specific points for each image, which increases the time taken for processing and inconveniences the user. To illustrate this, a computer with a 3.5Ghz clock speed processor takes approximately 30 seconds to process on one image, not including the time spent selecting reference points. The main logic of the Eigen function is discussed below.

1. The images are loaded from the database.
2. Next, the images are resized into a smaller size for faster processing speed.
3. Conversion from RGB to grayscale occurs.
4. Images are converted to LAB color space data in order to determine frontiers.
5. The user selects 12 to 20 reference points, preferably at least 4 points in each direction: above, below, left and right of the object.
6. The difference between each point is calculated, and the points are sorted from low to high in height, width, and length.
7. 5 pixels are added around each user selected point and a segment contour is generated.
8. Then the modified object is transferred to a temporary memory address, pixels are verified, and any pixel which is not in setting range is removed. This helps to minimize noise in the final image.
9. Lastly, a fuzzy optimization function provides a soft edge on the modified object.

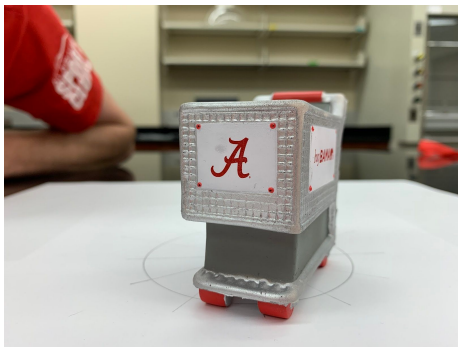

Figure 2: Image Before Background Removal

Figure 3: Eigen Based Background Removal

Figures 2 and 3 show a before and after view of the Eigen-based background removal function operating on a shopping cart dataset. As can be seen, the function effectively removes the background despite the variety of colors and objects present. It is also important to note that the function did an excellent job removing the object's shadow, with only trace parts remaining.

Although the current implementation of Eigen-valve based background removal is a manual process, automatic processing on large amount of images could be added by adding a Findlink function in. The Findlink function compares two close points from two different images and analogizes the next several points in the similar location. In addition, the Findlink function can be used in depth mapping and fusion, which is a key step in reconstructing 3D models from 2D data.

B. *Simplified Background Removal*

While the Eigen-value based background removal is very effective at separating the foreground out, the downsides in its current implementation are too great to make it a feasible choice for this project. Therefore, a new background removal function needed to be developed to reduce processing time. This simplified removal function relies on the background being a solid binary color (e.g. white or black). If the object is composed of darker tones and colors, the white background works best. By extension, white or lightly colored objects should have a black background placed behind them during image capture. The function takes in the dataset, number of

files in it, and background color as inputs, and then outputs the image's foreground or region of interest. Only a few steps take place to make this happen:

1. The image is converted to grayscale from RGB
2. Using Otsu's method a global threshold is computed and then used to binarize the image
3. If the background color is white, the image is inverted by subtracting it from a ones matrix
4. If the background color is black, the binarized image is not changed
5. Hole filling helps correct any gaps that may have formed and an opening function removes any objects consisting of 10,000 pixels or less. This helps to remove noise and any persistent background interference that remaining.
6. Lastly, the original RGB image is multiplied into the processed image to give color to the region of interest that remains.

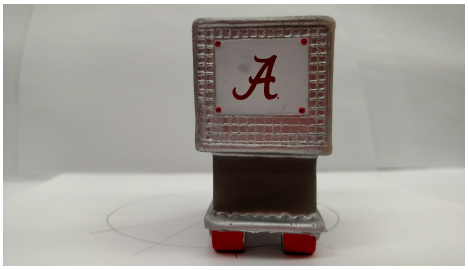

Figure 3: Shopping Cart Dataset Image

Figure 4: Simplified Background Removal

Figure 4 shows the result of the simplified background removal function on the image in Figure 3. Of note is the white background, which is needed for this function to work properly. Although dependent on the background color, this method is far faster than the Eigen-based function, with a runtime of approximately 1 second per image. A tradeoff of the speed of this approach is slightly worse quality in the processed image. In Figure 4, it can be seen that the shadow of the object has not been effectively removed, and some of it persists by the shopping cart's wheels.

Figure 5: Poor Camera Focus Effect on Background Removal

While the simplified background removal function is fast and fairly effective, approximately 6% of images in the dataset are affected by the quality of the photos taken. Poor camera focus on parts of the object can lead to blurring of that region and misclassification of that region as being part of the background. In Figure 5, the effects of poor camera focus can be seen on the left of the shopping cart and its handle. This effect can be attributed to camera focus due to adjacent images in the dataset not having this issue. A potential solution for this is to improve the Eigen-based method to be automatic and faster. Another option would be further development of the simplified background removal function. This could be done by tuning the threshold manually or working on defining a border for the object so the hole filling can remove these types of errors.

IV. Feature Detection

At its core, a feature is essentially a point or region of interest in an object that can be found and tracked across multiple adjacent images. There are multiple algorithm options available for feature detection, and out of these, three were considered for this application: SIFT, SURF, and FAST.

A. Scale-Invariant Feature Transform (SIFT)

Out of the three algorithms mentioned, SIFT is the oldest and slowest at computing feature points and vectors in 2D images. However, the slower runtime is offset by SIFT producing the largest number of feature points in dataset captures. SIFT functions by computing a difference of gaussians (DoG) to determine feature scale, and a histogram of the gradients surrounding a feature points. This histogram forms the feature vector of the point. Due to processing time being a concern, SIFT was not selected for the project.

B. Speeded-Up Robust Features (SURF)

SURF has a rough basis in the SIFT algorithm and was developed a few years later. SURF speeds up its processing by making several approximations in place of hard computation. For example, in place of the difference of gaussians (DoG) in SIFT, a box filter approximation is used. SURF does produce less features than SIFT, although its shorter runtime and similar performance made it a viable option for the project. In

the end, SURF was selected to get a good mixture of accuracy and runtime for the final product.

C. *Features from Accelerated Segment Test (FAST)*

FAST was considered for a while due to its fairly robust performance and very quick runtime. FAST is typically used in real-time applications, and has been successfully implemented in 3D reconstruction applications [1]. Due to the fixed camera location in this project and benefits of SURF, it was decided that FAST would not be used.

D. *SURF Feature Detection Result*

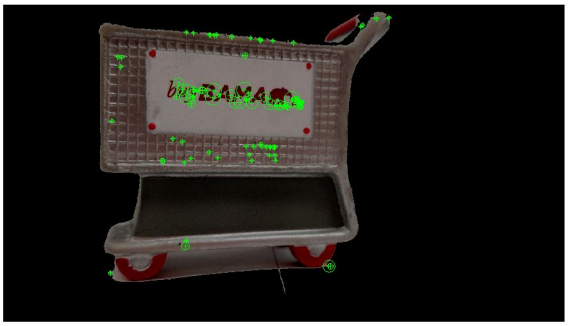

Figure 6: Feature Point Detection

Figure 6 shows the strongest 100 feature points found for the 27th image in the shopping cart dataset. The SURF detector does best with unique areas of the image like the "BAMA" text. Other areas which have less defined contours and color changes are not easily detected by the function. It is important to note that Figure 6 is simply displaying the indices of the features-- the green points are not the features themselves. However, showing the indices can give us information on how well the function is handling different objects and surfaces.

V. Feature Matching

Once feature points and vectors are found and calculated, matching of features between adjacent images needs to occur. The matching function compares the feature vectors for two adjacent images, and returns matched pairs for each feature that is similar enough to another feature. There are several criteria used to determine if a feature match is valid or not. These criteria can be manually tuned by the user, and their default values allow most matches through. The function was coded to allow the user to tune the strength, distance, and scale of the matches.

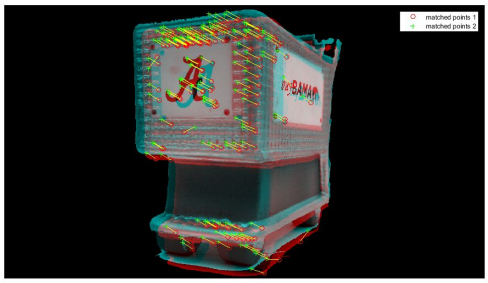

Figure 7: Matched Features

In Figure 7, the matched features between 2 adjacent images are shown. The green points correspond to one image, the red ones to the other, and the yellow lines connect paired features. After tuning, many unwanted matches were removed, as the default parameters can lead to matches forming across the image (much farther than the object actually shifted or rotated).

VI. Next Steps

Once feature matching is done between the images in the dataset, rough structure estimation and depth mapping need to occur. During development of this project, issues got in the way of the structure estimation and depth mapping implementation. Hopefully this will be completed in the future, but for now this project is presented without those crucial components. For reference, structure estimation essentially generates a rough 3D point cloud of the surface of the object, and depth mapping generates a depth for each image in the dataset. Ideally, the project would take that information and fuse the depth maps together to create a morphological 3D model. The last step would involve texture and mesh generation, which smooths the surface of the model and makes it look more like the object it is based on.

VII. Summary and Conclusion

Overall, this project was successful at implementing a simple, yet effective image capture environment, automatic dataset importation function, 2 background removal functions that each have their own perks, SURF feature detection, and feature matching between adjacent images. The results will continue to be used in development of a 3D reconstruction pipeline for processing 2D images into a print ready model. Specifically, a senior design project focused on 3D scanning will reference this project. In closing, the simplified setup proposed here is a feasible solution to the 3D reconstruction from 3D images problem, and hopefully the groundwork laid here will help future work on this topic.

Works Cited

[1] O. Muratov, Y. Slynko, V. Chernov, M. Lyubimtseva, A. Shamsuarov and V. Bucha, "3DCapture: 3D Reconstruction for a Smartphone," *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Las Vegas, NV, 2016, pp. 893-900

[2] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys, "Turning Mobile Phones into 3D Scanners,"Computer Vision and Geometry Group.

[3] E. Risten and T. Drummond, "Fusing points and lines for high performance tracking,"An introduction to biometric recognition - IEEE Journals & Magazine.

[4] D. Mistry and A. Banerjee, "Comparison of Feature Detection and Matching Approaches: SIFT and SURF,"ResearchGate, Mar-2017.

[5] E. Karami, S. Prasad, and M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images," arXiv.

Appendix

Work Log:

1. Physical capture setup and dataset capture: Trevor and Yuxin
2. Dataset import function: Trevor
3. Background Removal Functions
    a. Eigen Based: Yuxin
    b. Binary Background Based: Trevor
4. Feature Detection Function: Trevor
5. Feature Matching Function: Trevor