6-13-2017

Point Cloud Framework for Rendering 3D Models Using Google Tango

Maxen Chung
Santa Clara University, mhchung@scu.edu

Julian Callin
Santa Clara University, jcallin@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 8, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Maxen Chung
Julian Callin

ENTITLED

Point Cloud Framework for Rendering 3D Models Using Google Tango

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

[illegible]
Thesis Adviser

[illegible]
Department Chair

Point Cloud Framework for Rendering 3D Models Using Google Tango

by

Maxen Chung
Julian Callin

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2017

Point Cloud Framework for Rendering 3D Models Using Google Tango

Maxen Chung
Julian Callin

Department of Computer Engineering
Santa Clara University
June 13, 2017

ABSTRACT

This project seeks to demonstrate the feasibility of point cloud meshing for capturing and modeling three dimensional objects on consumer smart phones and tablets. Traditional methods of capturing objects require hundreds of images, are very slow and consume a large amount of cellular data for the average consumer. Software developers need a starting point for capturing and meshing point clouds to create 3D models as hardware manufacturers provide the tools to capture point cloud data. The project uses Googles Tango computer vision library for Android to capture point clouds on devices with depth-sensing hardware. The point clouds are combined and meshed as models for use in 3D rendering projects. We expect our results to be embraced by the Android market because capturing point clouds is fast and does not carry a large data footprint.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Many mobile devices have sought to allow humans to capture the world in digital form. Devices evolved from supporting only audio recording, to capturing still photographs, and then to capturing video. The next frontier for mobile device developers is providing users with methods of capturing and displaying three dimensional representations of objects they encounter every day. The advent of virtual and augmented reality has spurred the desire to reproduce real-world objects. The need for hardware and software capable of scanning and storing three dimensional models on mobile devices required a proof-of-concept software demonstration.

Currently, mobile hardware constraints are blocking the development of advanced 3D scanning applications. Phones and tablets do not possess depth perception, infra-red, or LiDAR sensors to gather data required to build models. Instead, devices rely on collecting hundreds of images taken of an object at different angles in order to stitch together and render a model. In addition, devices do not possess the computational power to build models and rely on cloud services to assemble the images in three dimensions. The device must maintain a fast and reliable Internet connection, the user must wait many minutes to receive their rendering, and the user must conduct the tedious process of taking many photos of the object at different angles.

1.2 Solution

Our solution leverages the Google Tango software as well as advances in mobile device hardware to capture three dimensional models in Android applications. Our solution will improve upon existing solutions by allowing for fast, lightweight, and simple model assembly. Our framework will help developers use three dimensional models scanned from the real world in the next generation of mobile devices.

Chapter 2

Literature Review and Assessment of Previous Work

2.1 Introduction

The process of generating a 3D model poses a challenge; it is difficult to represent the data in an understandable way. In order to overcome the challenge, 3D data is rendered into a mesh. A mesh is a collection of vertices, faces, and edges that are compiled to form a visual representation of the surface of a scanned environment or object. This allows the user to view a 3D model of the subject, complete with accurate data points, on a two-dimensional medium such as a screen. Shading and contrast are often used to display variances in the texture of the objects surface. The implementation of the mesh is crucial to providing the user a clear and accurate rendering while retaining a limited storage footprint.

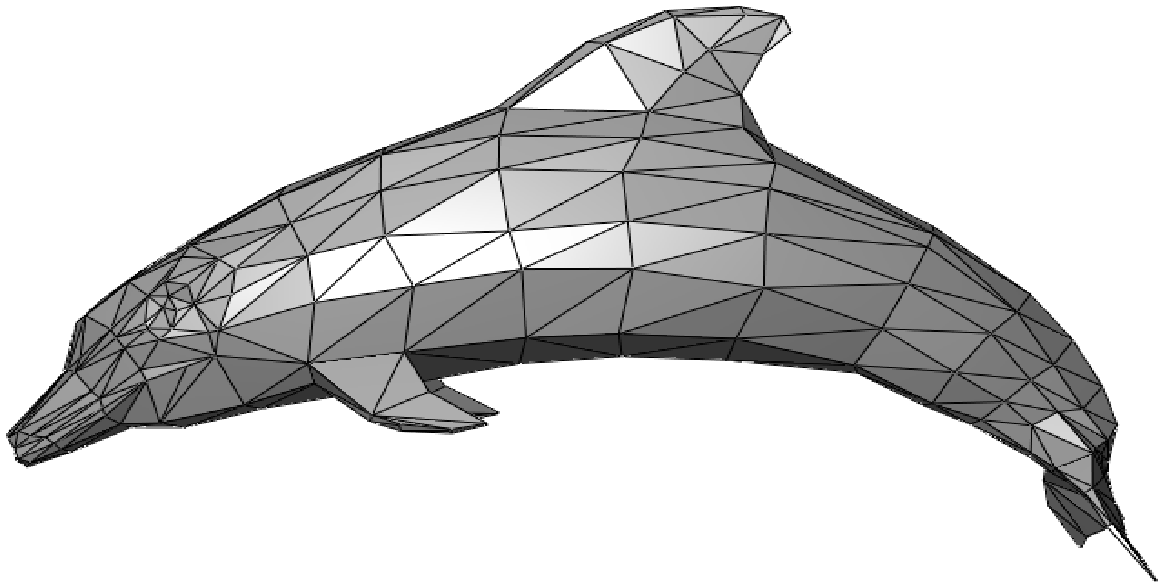


Figure 2.1: A point cloud mesh model of a dolphin

2.2 Stitching Images to Create 3D Objects

Work in the point cloud capturing field of computer vision has been primarily confined to spatial mapping. Automobile manufacturers require cars to know their surroundings as they become more autonomous. Furniture retailers allow their customers to scan their rooms and visualize new furniture pieces virtually. The world of object capture has not been explored extensively in the mobile market. Most companies have adopted a method of three-dimensional object capture known as stitching in which the user captures up to hundreds of photos of an object from different angles and the software uploads the images to the cloud for assembling. Autodesk is an example of a company whose mobile application 123D Catch utilizes this type of capture.[1]

2.3 Project Tango: Beyond Imagery

Google Tango is a software framework which lets Android devices with advanced sensor suites capture point clouds, depth, and movement data. Since June 2014, developers have fiddled with the Project Tango Development Tablet and software. Many consumers did not have access to a device capable of running the Tango software because the only mobile device on the market was the expensive developer tablet. This changed when Lenovo announced their Phab2 phone, a large Android smart phone with numerous cameras and infrared sensors that could make use of the Tango software. The phone debuted in November 2016 for the consumer market and developers have doubled their efforts in applying the Tango framework to their own applications. The Phab2 will be the primary platform for testing and development of this project.

2.4 Google's Contributions to Tango on Android

Google has provided developers with great examples of Android applications running various components of Project Tango on the Tango Github repository.[2]. Our project utilizes code for the point cloud preview, rendering preview, and user interface components that were taken directly from Googles Project Tango repository. The bulk of the innovation of this project comes in combining the capture aspects of Tango with the processing capability of the open source computer vision library Open Computer Vision (OpenCV).

[1]Autodesk has discontinued 123D Catch as of January 2017
[2]`https://github.com/googlesamples/tango-examples-java`

2.5 OpenCV for Processing Point Clouds

OpenCV provides the tools needed to interpret, optimize, and mesh a raw point cloud captured an Android device using Tango. Once the point cloud is captured, OpenCVs Point Cloud Library (PCL) enables the storage, preview, and meshing of the point cloud for developers and consumers to utilize. Large amounts of formal research are being done on efficient ways of assembling and meshing point clouds. We will present a couple examples of cutting-edge point cloud processing and meshing techniques but will focus on the abilities of OpenCV to accomplish our goal without delving into complex mathematics.

2.6 An Application Perspective

Our project focuses on the capture and concatenation of 3D Point Clouds using the Google Tango platform. As we continue refining our product, understanding our applications of meshing can help us achieve the best implementation of our product. An article from ACM Computing Surveys, Visual Contrast Sensitivity and Discrimination for 3D Meshes and their Applications, focuses heavily on the best applications for object meshing and how to maximize efficiency and effectiveness of the models. The article classifies different applications for 3D object representation and analyzes the uses for models within those categories. 3D models are used regularly in all sorts of industry settings. Everything from product design to gaming to bioinformatics optimizes the use of 3D models. Attene (2013) classifies the two main data sources for meshing as the digitization of real-world objects and the tessellation of virtual, synthetic data typically produced by a computer (p. 2)[2], and we imagine our framework being used on mobile devices to capture 3D Point Clouds of objects in real-time. One of the major issues with digitization is accurately representing surface holes, gaps and other geometric imperfections of an object. The shadows from these geometric anomalies make the initial image capture difficult. These issues often lead to small misrepresentations in the polygon mesh; however, Attene (2013) explains that for pure visualization of an object, these small imperfections generally do not present a significant problem as only the existence of significant holes [are] generally deemed unacceptable (p. 11). On the other hand, if the model is being used for prototyping purposes, the mesh must be free of errors or missing data to maximize the effective testing of the prototype. In order to achieve a model that can be used for a wide variety of applications, specifically ones where accurate representation is critical, the polygon mesh may need to undergo local repairs post image-capture. As we continue refining our framework and imagining applications for it, we must investigate more of the applications that we see as potential fits for our product. If our product is used in situations that require rigorous attention to detail, we must spend more time refining our capture and repair process to create the most accurate Point Clouds necessary for eventual meshing. However, if our framework is to be used more commonly in everyday situations for visualization purposes, than the accuracy of the Point Clouds are not as critical.

2.7 Mesh Construction Process

Several different approaches are taken to create meshes out of point cloud data. According to Boubekeur (2015), the standard for doing such a meshing is an indexed triangle mesh (p. 151)[2]. Visually, this creates a lattice-like structure of triangles which can further help a user construct an image of the surface defined by the points. Computationally, it is represented as a list of vertices and a list of the triangles. While we will not be directly dealing with the actual triangulation, we will be utilizing a backend (Point Cloud Library) which leverages the triangle representation. As such, it is important to understand the (approximate) steps that the software will be taking to mesh the point cloud. Boubekeur (2015) defines four steps for meshing: pre-processing, surface model definition, mesh extraction, and post processing.

N. Wongwaen et al. provide an excellent description of how to build a mesh of a 3D object form a point cloud using triangle planes and three-dimensional cubic segmentation of the object region of the point cloud. This could be extremely helpful if we wanted to write our own meshing algorithm instead of using a computer vision library. This also gives us valuable insight into how meshes are created and does not use complicated mathematics in the process or the explanation. In addition, the cubic segmentation of the object region may be very useful for our feature which provides the user fine control over which points are scanned.[4]

L. Alboul and G. Chliveros outline a method for merging image and point cloud data to make a mesh. Their techniques for meshing the point clouds are very basic and could help us in the early stages of our project, but the publication contains extraneous information such as pre-processing of point cloud data to integrate it well with other data sources such as photo and LiDAR. Also, the publication goes into great detail about using camera angle, aperture, and calibrations to further improve mesh renderings which are essentially useless in our project.[1]

2.8 Global vs Local

Point cloud meshing can be done either locally or globally, each with its own merits and drawbacks. Point Cloud Library uses a local meshing technique, which Boubekeur (2015) describes as looking only at a subset of the input set to t a simple object (p. 152). Basically, meshes are created by taking small parts of the object and forming a surface. This is repeated across the entire object until it is fully defined. Global meshes take the entire object and attempt to wrap a surface around it. This type of mesh is slower because of the large input, but it can deal with gaps and other abnormalities better. Local meshes are faster to create because they are dealing with smaller inputs, however they do not work well irregular shapes. This is important to note when we are looking at what kind of object our framework is able to reproduce.

2.9 Visual Contrast Sensitivity for 3D Meshes

When constructing a 3D mesh, it is important to generate an image that allows the user to discern between varying geometric features of the scanned environment. The current method utilises contrast to show variances in the properties of the scanned environment, therefore allowing the user to interpret the data in a familiar way. It is important to understand how the human mind interprets varying levels of contrast in order to be able generate the best possible mesh. This article focuses on methods of improving current algorithms in order to generate a better contrast model and improve distortion detection by the user. Contrast is used in a wide application of 3D data representation ranging from terrain mapping to MRI image scanning to infrared image generation. In this article, Nader, Wang, Wheeler, and Dupont (2016) studied the brains abilities to distinguish between varying amplitude and acuteness of contrast distortion [3]. As stated in the article, most existing geometry processing algorithms are driven and/or evaluated by geometric metrics like Hausdorff distance or root mean square error (RMS) [and] do not correlate with human perception (p.497). As a result, the implementation of perceptually driven algorithms would have a significant impact on the users ability to comprehend the data shown. Additionally, the methods used to compress the data to a size suitable for most mobile devices results in loss of information and simplification of the data, effectively reducing the resolution of the mesh image. Nader et al. (2016) found that one of the best ways of determining the threshold of human contract variance is by finding the Just Noticeable Distortion (JND) value. The JND gives a quantifiable value that illustrates the point where the human brain can begin to distinguish between data points when using contrast as a means of differentiating values. In order to find the JND, three characteristics of vertex faces must be used, For each of these faces we estimate its contrast (Section 3), frequency (Section 4.1) and visual regularity (Section 4.2). This allows us to compute the threshold [of JND] (p.502). Once the threshold is found, contrast values can then be applied to the mesh image. It is important for us to take the JND into account for our own framework as the localization of the 3D point-cloud requires some form of compression in order to work on mobile devices. As a result, some data is lost and the need for a more efficient and effective 3D mesh image algorithm grows in order to provide the user with clear data while minimizing costly memory usage. Moving forward, we believe we are going to utilize the OpenGL platform to render the mesh into an object, and a cursory search on OpenGL and JND will be part of our research plan.

Chapter 3

Requirements

3.1 Functional

- Allow the user to view and capture point cloud data in real time
- Using captured 3D point cloud data, create a triangulated wrapping mesh
- Export 3-D Render as a CAD model
- Save point clouds and 3D renders as files that can later be accessed by the user

3.2 Non-Functional

- Well documented for ease of use for developers
- Abstract Google Tango
- Minimal data sent to server and back
- Quick to parse the point cloud data

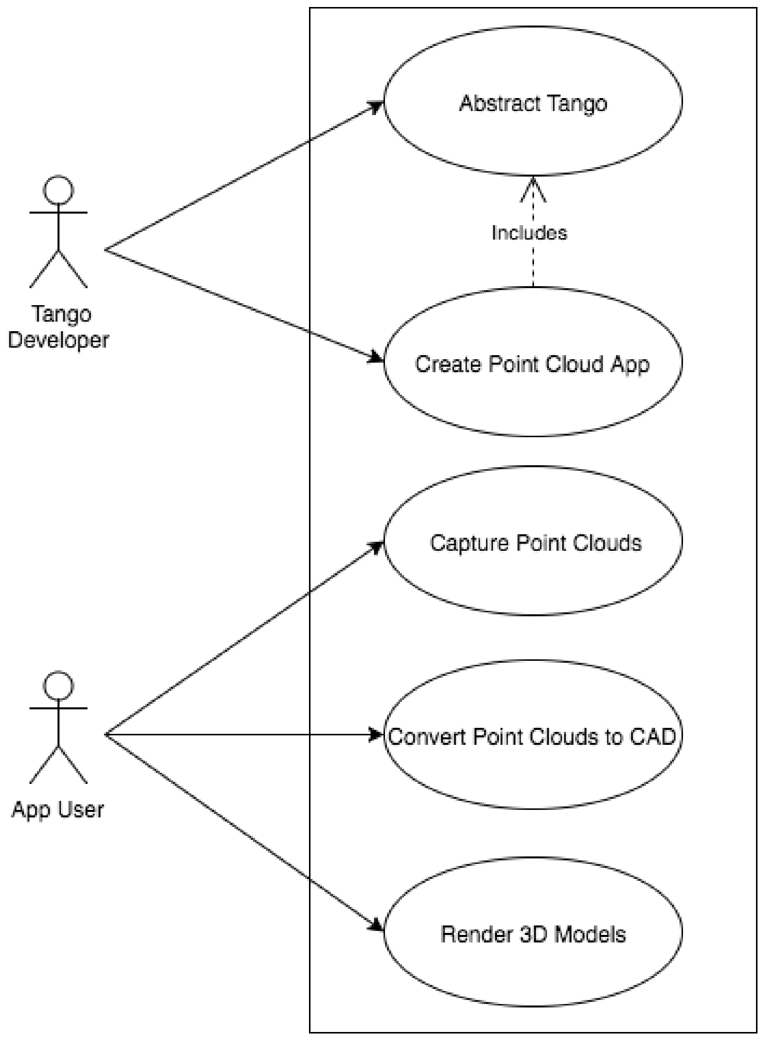
3.3 Design Constraints

- Work on Lenovo Phab 2

Chapter 4

Use Cases

4.1 Diagram



4.2 Explanation

Use Case diagrams illustrate how each user will use different components of the system. Our diagram has two separate users and five different use cases.

1. Abstract Tango Point Cloud Capture

   Goal: Use the framework to simplify the underlying Tango software to produce an easy way to capture point cloud data from the sensor.

   Actors: Tango Developers

   Pre-Conditions: Framework has been added to the project.

   Post-Conditions: Tango sensors have been initialized for point cloud capture.

   Exceptions: Tango cannot be reached or has not been released by another program.

2. Create Point Cloud App

   Goal: Use the framework to create an app that utilizes the point cloud functionality in Tango.

   Actors: Tango Developers

   Pre-Conditions: Framework has been added to the project.

   Post-Conditions: App can capture or analyze point cloud data.

   Exceptions: Tango cannot be reached or has not been released by another program.

3. Capture Point Clouds

   Goal: Use a Tango device to save the point cloud data currently being seen by the Tango sensor.

   Actors: App Users

   Pre-Conditions: Tango device is being used. The object to be captured is within the range of the sensor.

   Post-Conditions: The device has saved the point cloud data to a local storage location

   Exceptions: Tango cannot be reached or has not been released by another program. Conditions (brightness or other) do not allow accurate data to be collected. Not enough space to store the data.

4. Render 3D Models

   Goal: Using previously taken point cloud data, construct a point cloud representation of the object

   Actors: App Users

   Pre-Conditions: Point cloud data has been collected.

   Post-Conditions: A point cloud of the object has been rendered and saved in a local storage location

   Exceptions: Not enough point cloud data has been taken. Point cloud data collected is not accurate. Not enough space to store the render.

5. Convert Point Clouds to CAD

   Goal: Take a point cloud and export the data in a CAD file

   Actors: App Users

   Pre-Conditions: Point cloud data has been collected and/or rendered into a 3D object

   Post-Conditions: A CAD file representing the point cloud has been stored in the user specified location

   Exceptions: The point cloud data is distorted or out of focus. There is no more space to store the CAD file. User has insufficient permissions to store the file in the specified location.

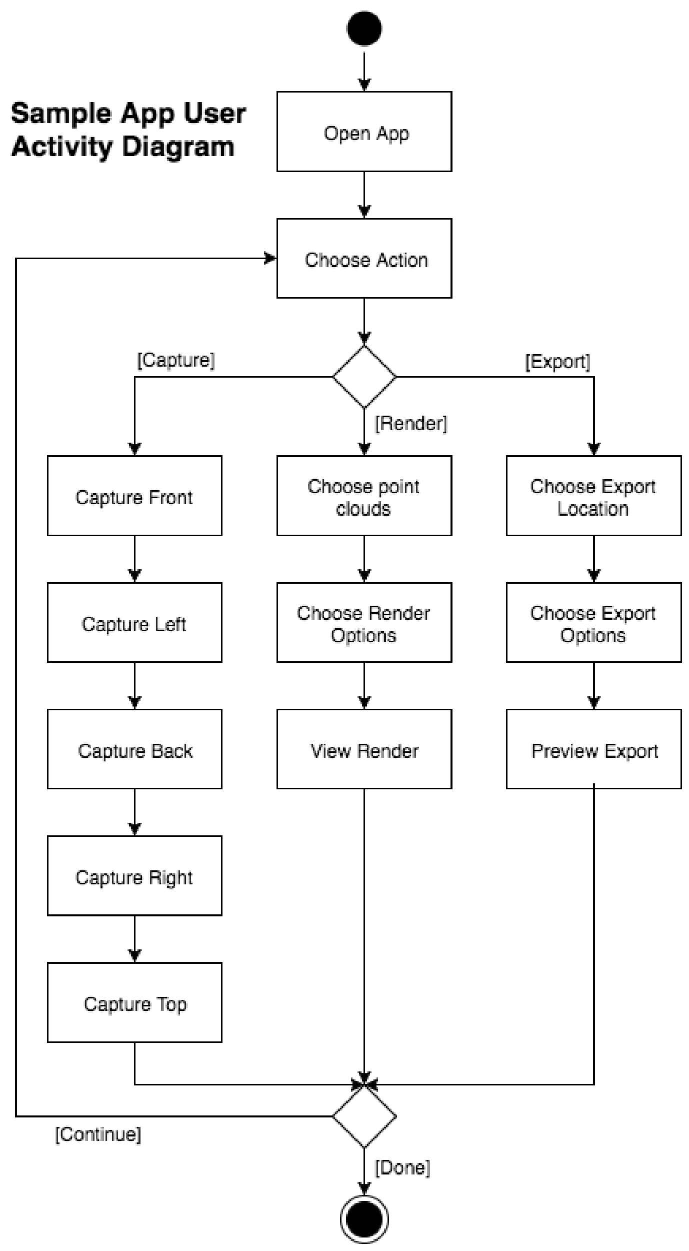
Chapter 5

Activity Diagram

The activity diagram details the flow for each user who might interact with the system.

5.1 Sample App

This actor would be using the sample app that we are going to build using our framework. The diagram details how a user might interact with this app, and gives an idea how a user might interact with other apps that were built using our framework.


Sample App User Activity Diagram
Open App
Choose Action
[Capture]
[Render]
[Export]
Capture Front
Capture Left
Capture Back
Capture Right
Capture Top
Choose point clouds
Choose Render Options
View Render
Choose Export Location
Choose Export Options
Preview Export
[Continue]
[Done]

Chapter 6

Conceptual Model

6.1 Front End Application

6.1.1 User Interface

Application Opens to Start Menu, asking user to choose if they want to capture the front or the back of the object. This can be done in any order.

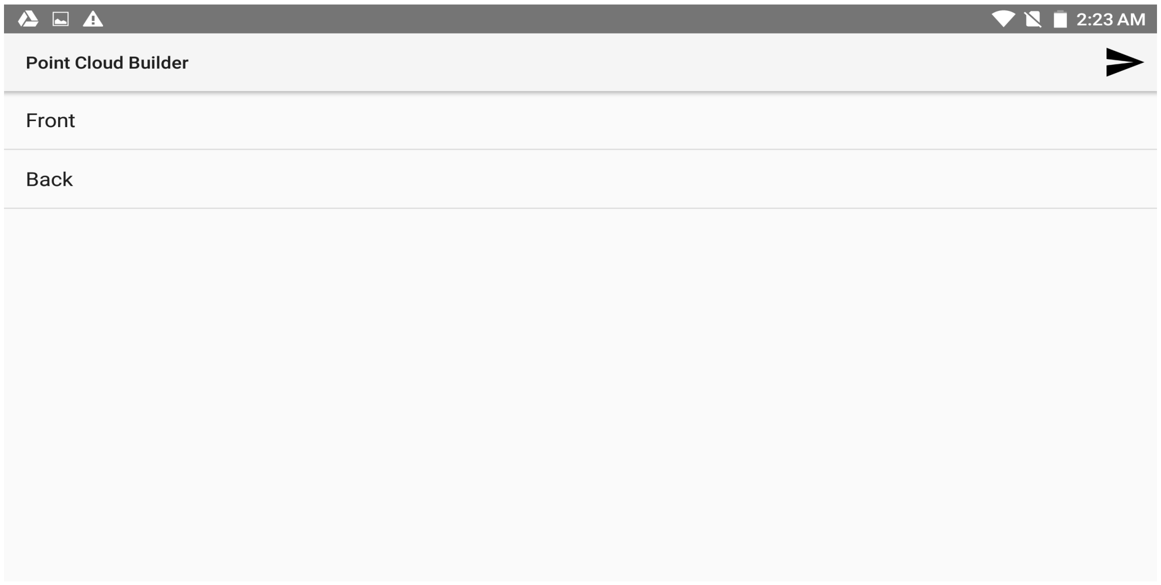


Figure 6.1: Starting State of the Main Menu

Once the user selects which side they are going to capture, they are prompted to input the name they would like the .pcd file to be saved as.

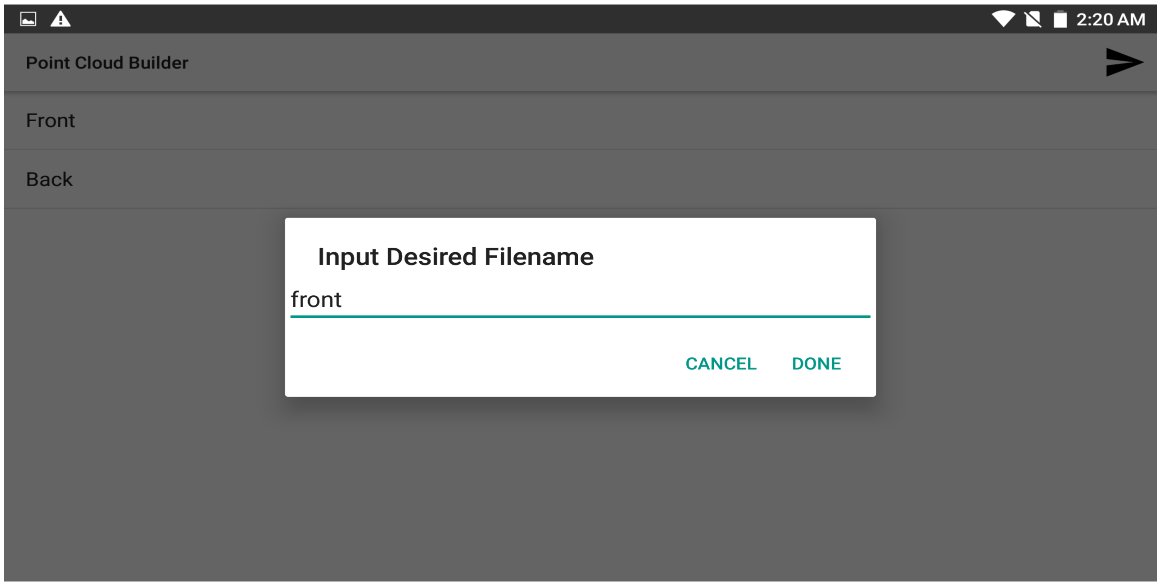


Figure 6.2: Input Filename Dialog

Next, the user captures the object by hitting the save button. Note the point count and the average point distance gives an idea of how many points you would capture at that time. Figure 6.3 shows the interface for capturing a point cloud. The dark scene has many colored dots indicating points to be captured while the camera preview on the top right gives the user an idea of what they are capturing.

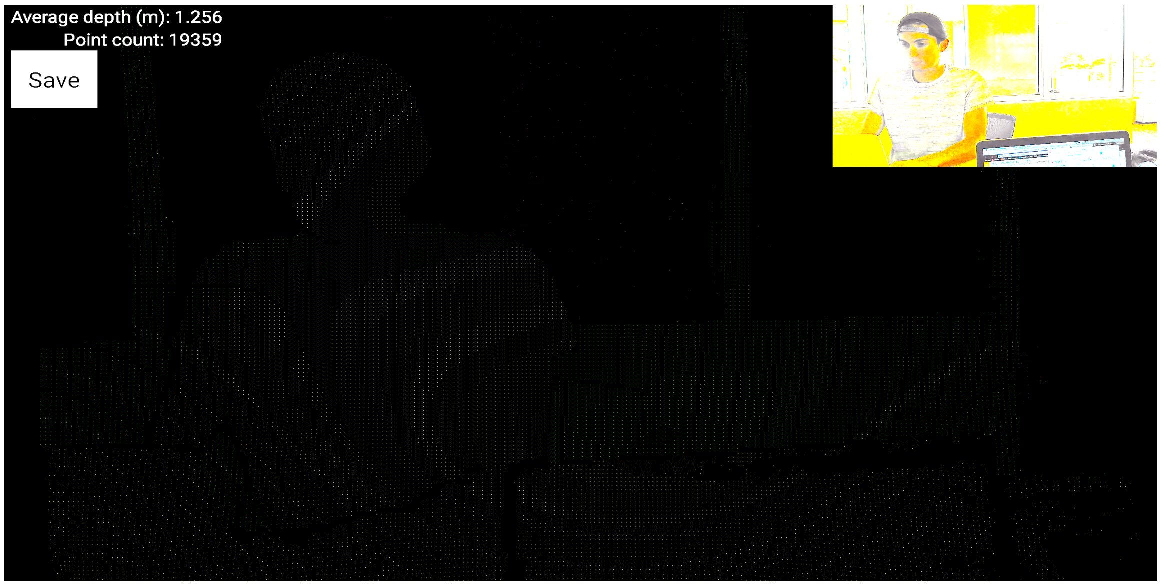


Figure 6.3: Capture Screen

The main menu then indicates which point clouds the user has captured successfully by changing the text color to green

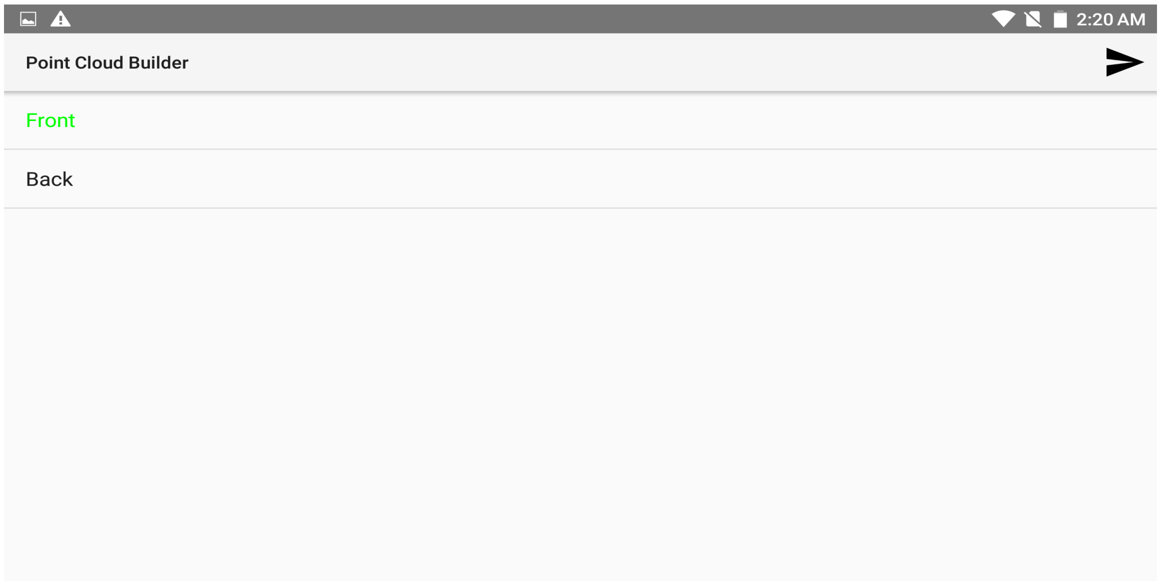


Figure 6.4: Main menu after front has been captured

This is repeated for the other side as well.

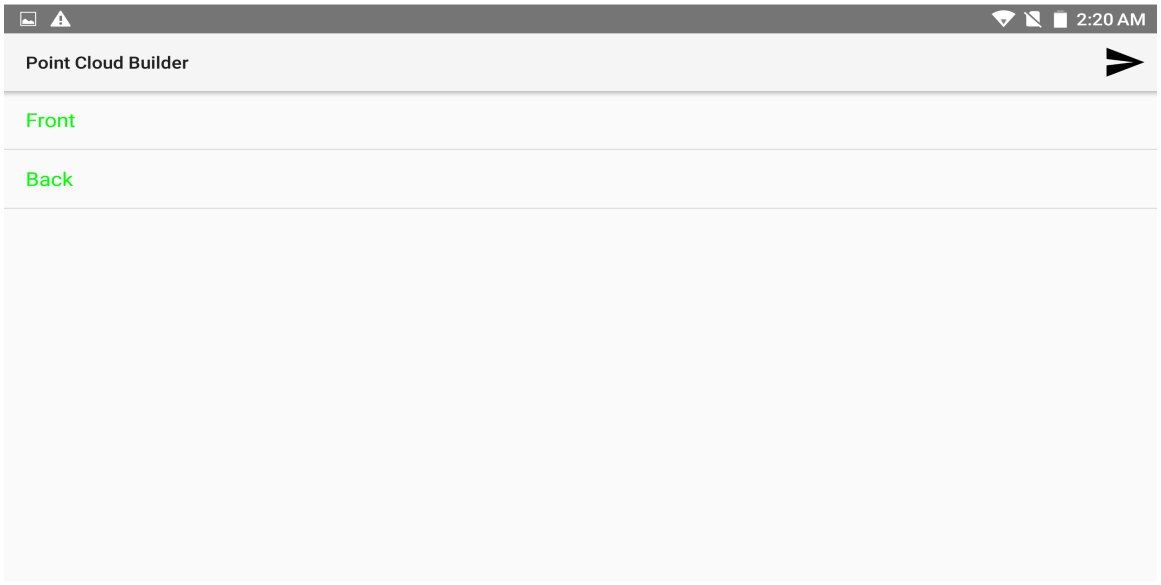


Figure 6.5: Main menu after both have been captured

The user can then hit the send button to send the data to the server, and they will be prompted to enter the filename for the mesh that is returned. They are also prompted for a standard deviation, which tells our algorithm how closely to bring the two halves together.

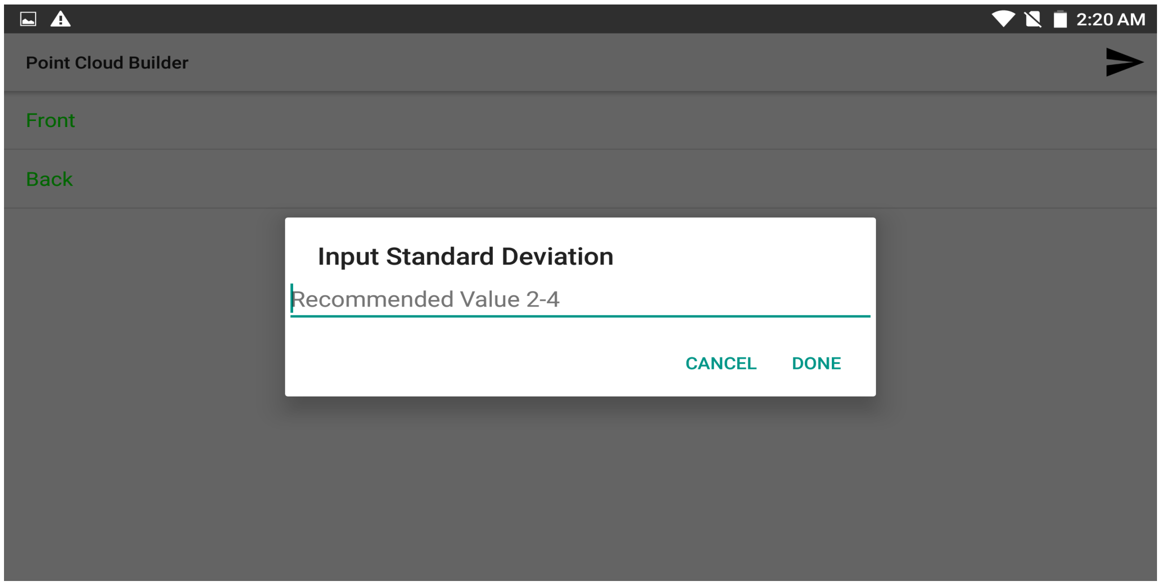


Figure 6.6: Standard Deviation Input

Lastly, the user can then use a 3rd Party Viewer to view the rendered model. In this case, we are using CAD Assistant from the Google Play Store.

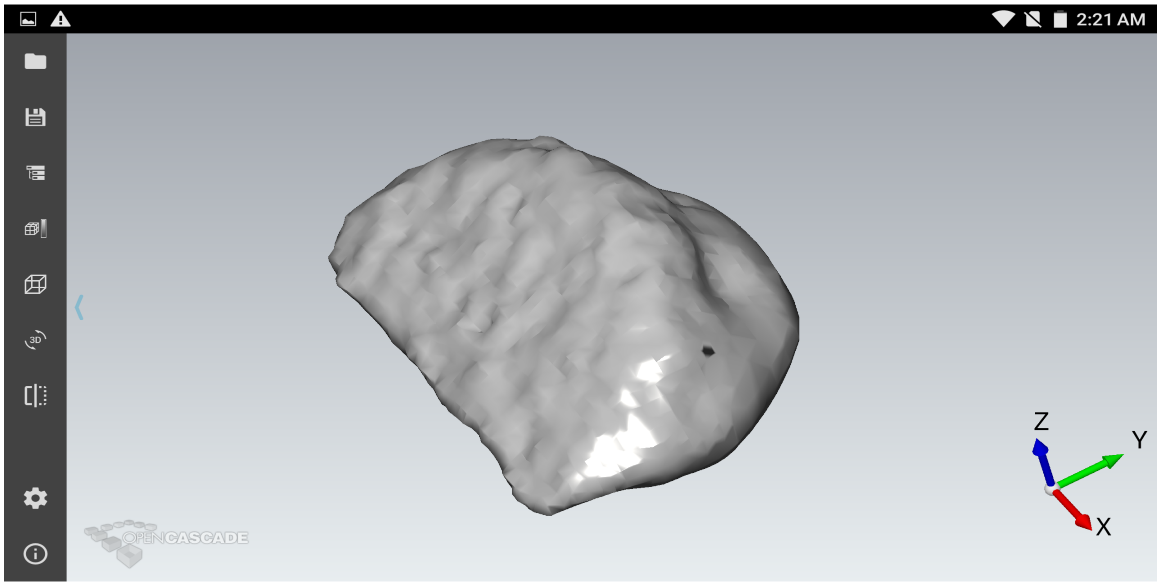


Figure 6.7: Finished Capture

6.2 Back End Application

6.2.1 PCL on AWS

The back end of the application consists of a Amazon EC2 T2 Micro instance running Ubuntu 14.04. The web server is written in C++ and utilizes Point Cloud Library 1.7.1 to process point clouds.

The front end application sends raw point clouds in PCD format[1] to an AWS server as a HTTP POST request with a JSON payload. The server cleans the point clouds, assembles them into a single cloud, meshes the cloud, and sends the mesh back to the client as a .obj file.

The payload JSON object contains three fields: two objects– the front cloud and the back cloud, and 1 meta-data object which contains a single parameter std_dev which is a user defined floating point number generally between 3 and 4.5. std_dev specifies how the object's two sides are fit together. Larger values bring the object's sides closer while smaller values keep the two sides further away.

6.2.2 Processing and Meshing Point Clouds

Upon receiving the payload, each point cloud is persisted on disk in a .pcd file, and then loaded into memory. Each cloud goes through a series of optimizations and noise removal processes. First, a plane is removed from the cloud using PCL's segmentation capabilities in
`<pcl/segmentation/sac_segmentation.h>`.

The method uses a mathematical model of a plane to search for sets of points which fit this model indicating that they belong to a plane. The points that lie on the plane are filtered out of the cloud. This serves to eliminate the flat surface on which the object lies.
Next, the processor searches for individual outliers in the point cloud and removes them using
`<pcl/filters/statistical_outlier_removal.h>`.

Then, the processor selects the largest cluster of points as the final object to eliminate all other points and clusters. This is done using another segmentation method
`<pcl/segmentation/conditional_euclidean_clustering.h>`

Finally, the processor uses Moving Least Squares surface approximation to smooth the surface for the meshing algorithm to work with.[2]
The back cloud undergoes an additional transformation. Since both clouds are captured from the same perspective, the back cloud must be inverted and moved slightly to align correctly with the front cloud. Once this is completed, the front and back clouds are merged
The merged cloud is meshed using Poisson Surface Meshing[2] and the result is saved as a .obj file. The server loads this .obj file as plain text and sends it back to the client after the processor has returned.

[1]`http://pointclouds.org/documentation/tutorials/pcd_file_format.php`
[2]`http://www.pointclouds.org/assets/icra2012/surface.pdf`

Chapter 7

Technologies Used

Technologies used were primarily open source software packages

- Android: A mobile operating system which runs on many mobile devices.
- Google Tango Framework: A framework for gathering and interpreting sensor data from advanced sensors on hardware devices running Android.
- Java: The primary programming language of the Android operating system. Java is widely used and many developer tools and frameworks provide Java APIs.
- OpenCV: An open source computer vision library for gathering and processing point cloud data from the Tango Framework. Point Cloud Library (PCL), included within OpenCV, provides the functionality for working with point clouds.
- MeshLab: An open source mesh viewing and editing software used to quality-check the meshes produced by our software.
- Lenovo Phab2 Phone: A recently released consumer mobile device with hardware capable of capturing point clouds and depth data.
- Github: A system for version control and backup used to ensure that catastrophic data loss does not occur, along with allowing collaboration

// Chapter 8

Design Rationale

8.1 Technologies Used

8.1.1 Android

This project will be implemented on Android devices because the Google Tango software is written for Android and the Lenovo Phab2 phone runs the Android operating system.

8.1.2 Google Tango

Tango will be used because it is the only depth perception software framework provided by Google for Android. Google is the creator of Android so the Tango software will be well maintained. We will be able to find immediate answers from the Tango community for our questions regarding Android.

8.1.3 Java

Java is the most widely used language for Android programming and team members have experience developing on Tango with Java. There are many online resources for solving issues we may run into while using Java.

8.1.4 OpenCV

OpenCV's Point Cloud Library is a widely used framework for processing point clouds. We did not want to implement complex mathematics to capture and process point clouds, so we chose to use OpenCV. OpenCV provides access to Point Cloud Library which housed the methods we used for concatenating optimizing, and meshing point clouds. PCL was the only open-source option for an API for working with point clouds and meshes.

8.1.5 Lenovo Phab2 Phone

The Phab2 phone is the latest hardware release of a phone which can run Google Tango. We would like our project to be developed on the cutting edge of Tango compatible hardware. We chose the Phab2 instead of the Tango developer tablet because it is a consumer product and is considerably newer than the developer tablet.

8.1.6 Github

Github is a popular and easy way to host shared source code and collaborate on the project's development. Github offers a

8.2 User Interface and Framework

The goal of our framework is to provide developers with the tools to both capture and process point cloud data.

Framework

Our framework will be made to give the programmer the ability to create powerful Tango point cloud applications without having to worry about the underlying complexities of Tango. To do this, we will focus narrowly on point clouds, and configuring certain aspects of the system to process solely point clouds. For example, when the Tango sensor is initialized in our framework, it is primed for point cloud capture. The functions that we will choose represent what we believe to be the most high level use cases while still giving the developer options to choose how the functions will be utilized. Additionally, we are going to keep the functions completely separate so that one can be called without calling the others.

User Interface

For our demo app that we will be developing, we will keep the functionality simple and transparent so that developers would be able to see each piece of the framework being used. We will fully utilize the framework, implementing each option that we provide.

Chapter 9

Test Plan

9.1 Unit Testing

Test each of the following functions to ensure that they produce the correct output with the given input. The functions will be completed in numerical order to provide input to the following function

1. tangoPointCloudInit()
2. capturePointClouds()
3. renderPointClouds()
4. pointCloudtoCAD()

9.2 Alpha Testing

To test our framework, we will be developing an Android app alongside it so that we can test the functionality. This will allow us to see what functions may be required beyond those we have already planned for in order to create a functional android app. While testing the framework, we will focus on ease of use from a development perspective, that is, how difficult it is to create an application given our framework. We will develop the framework using black box testing, and move on to do white box testing once the framework is operational.

9.3 Beta Testing

Beta testing will be done by colleagues who have experiencing developing Android apps. We will first ask the testers if they think they could create a working Tango app given only our framework and some instructions. We hope to see if the developers see the framework as helpful, or if it is either too simplified or too complex.

Chapter 10

Cost Estimates

Our project is relatively low-cost due to being largely comprised of software. We will require two mobile devices for each team member to build and test the application on.

10.1 Cost Table

Item	Cost
Tango Development Table	Free (donated from last year)
2 Lenovo Phab2 Phones	$499.99
Total Cost	$1000

Chapter 11

Risk Analysis

11.1 Risk Table

Risks associated with undertaking our project, the potential consequences of these risks, and mitigation strategies

Risk	Consequences	Probability	Severity	Impact
Point cloud assembly is very mathematically complicated	Features cut	.5	9	4.5
Frameworks we are using get updated/deprecated	Progress Delay	.4	7	2.8
Poor time management	Cutbacks on project	.3	5	1.5

11.2 Mitigation Strategies

Risk 1:

- Use OpenCV to process point clouds without delving into low level details
- Familiarize ourselves with common point cloud processing and rendering techniques
- Use mock point cloud data to practice processing with OpenCV

Risk 2:

- Keep up-to-date with all frameworks we are using via the web
- Research development plans for frameworks in the near future

Risk 3:

- Establish deadlines and expectations for each team member's contributions
- Aim to begin all work before the date we have set

Chapter 12

Development Timeline

12.1 Gantt Chart

This graphic shows the various tasks associated with the project, and the team members that worked on each task.

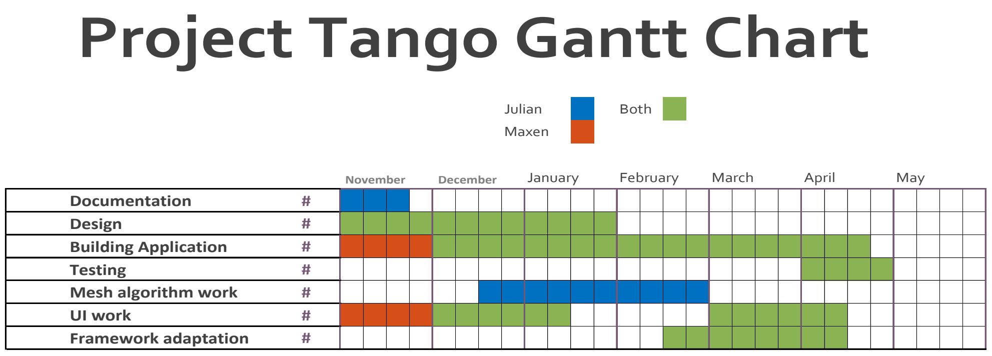


Figure 12.1: Development Timeline Gantt Chart

Chapter 13

Results

13.1 Test object: Mouse from side

Prior to any optimizations, we captured a computer mouse resting normally on a flat table with a flat wall behind. The capture shown below contained the faade of the mouse, the wall, the table, and a large amount of random points in the space between the mouse and the wall.

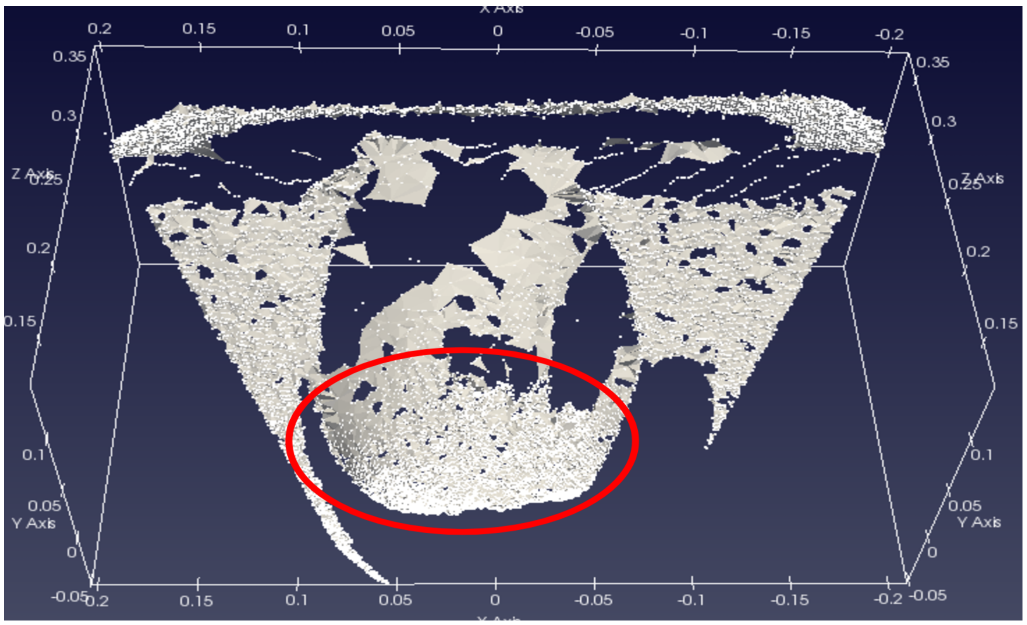


Figure 13.1: Computer Mouse

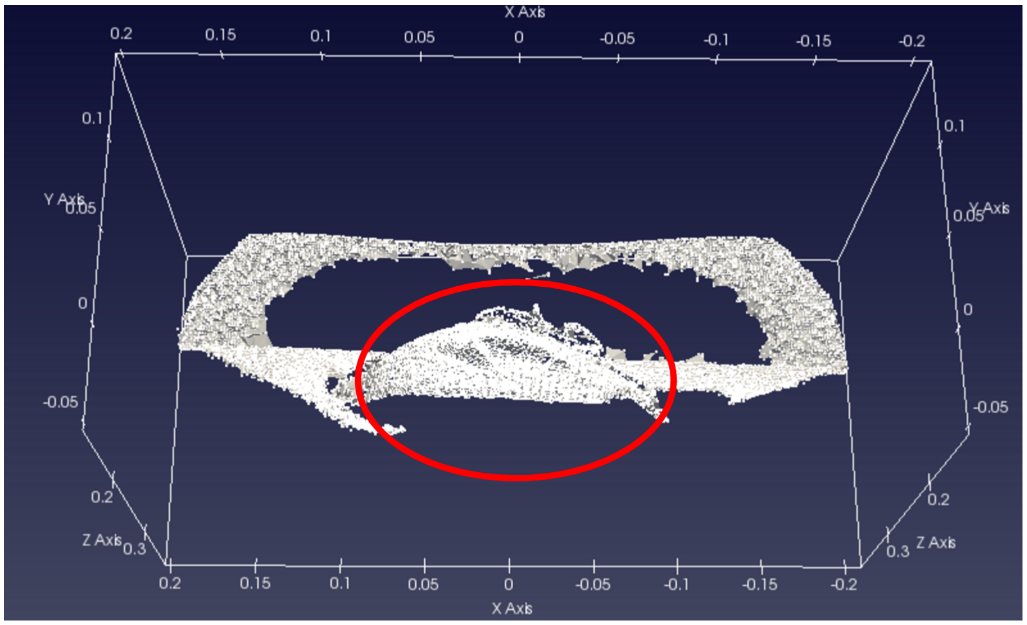


Figure 13.2: Computer mouse from another angle

Next, plane elimination and outlier elimination were used to remove many noisy points along with the table and large parts of the wall.

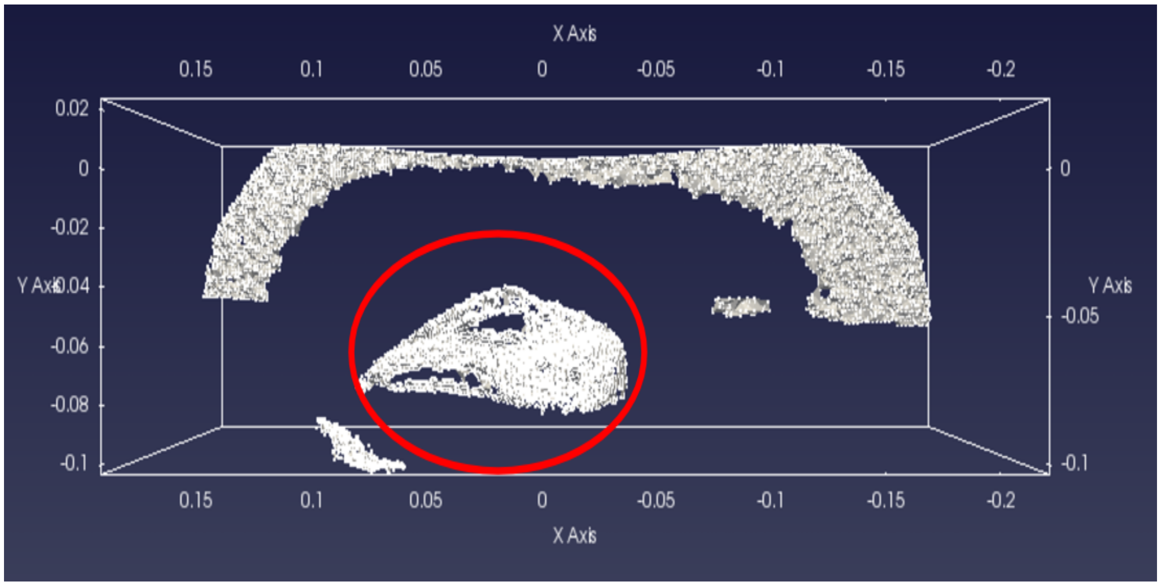


Figure 13.3: Mouse with outlier and plane removal

A model of the mouse can be seen in Figure 13.3. This model was selected by splitting the capture into clusters, selecting the largest cluster, and removing all other clusters. The final result can be seen in Figure 13.4.

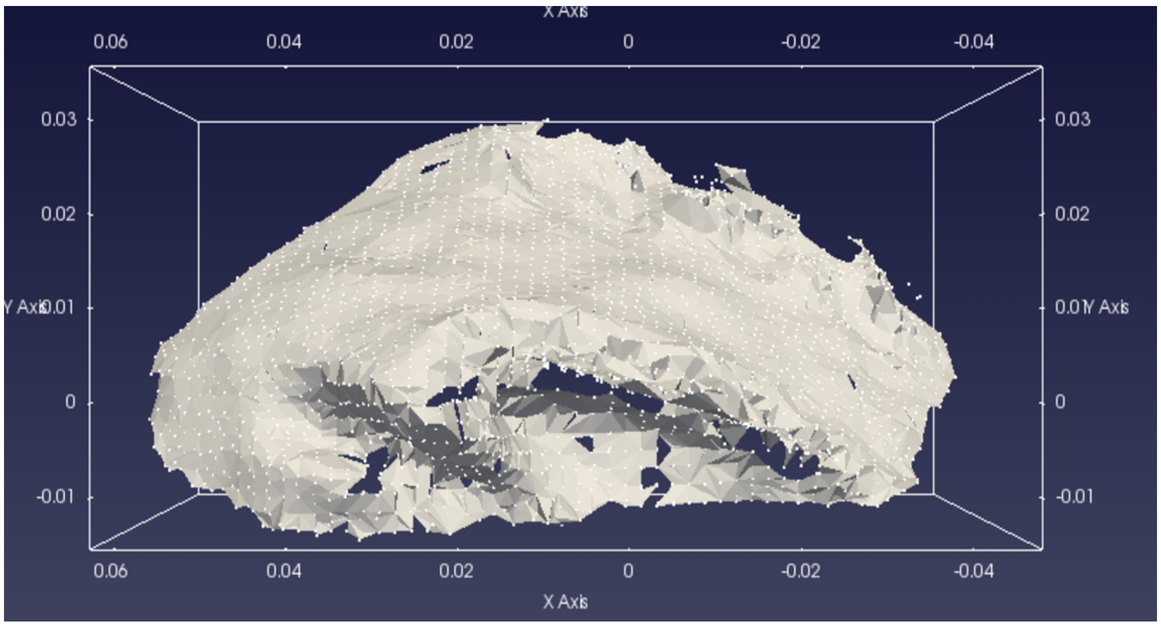


Figure 13.4: Mouse isolated using cluster selection

13.2 Improving the mesh

Figure 13.4 shows a recognizable representation of the side of a computer mouse, but the mesh still has imperfections. To create a better mesh, some parameters within the meshing and optimization algorithms were adjusted. The search distance for outliers was reduced in order to classify a greater number of points as outliers and flag them for removal. The maximum surface angle between mesh triangles was reduced to create a smoother mesh surface. The minimum size for a cluster was also reduced, enabling the cluster selection and elimination algorithm to select and eliminate small clusters close to the main object cluster.

In addition to small changes to algorithms, we decided to capture additional objects from a top-down perspective. This decision enabled the immediate elimination of the wall behind the object. The only plane left in the scene was the tabletop upon which they object lay. This made orienting the camera and eliminating the plane much easier. Figure 13.5 shows the side of a shoe captured from the top-down perspective using the improved algorithms.

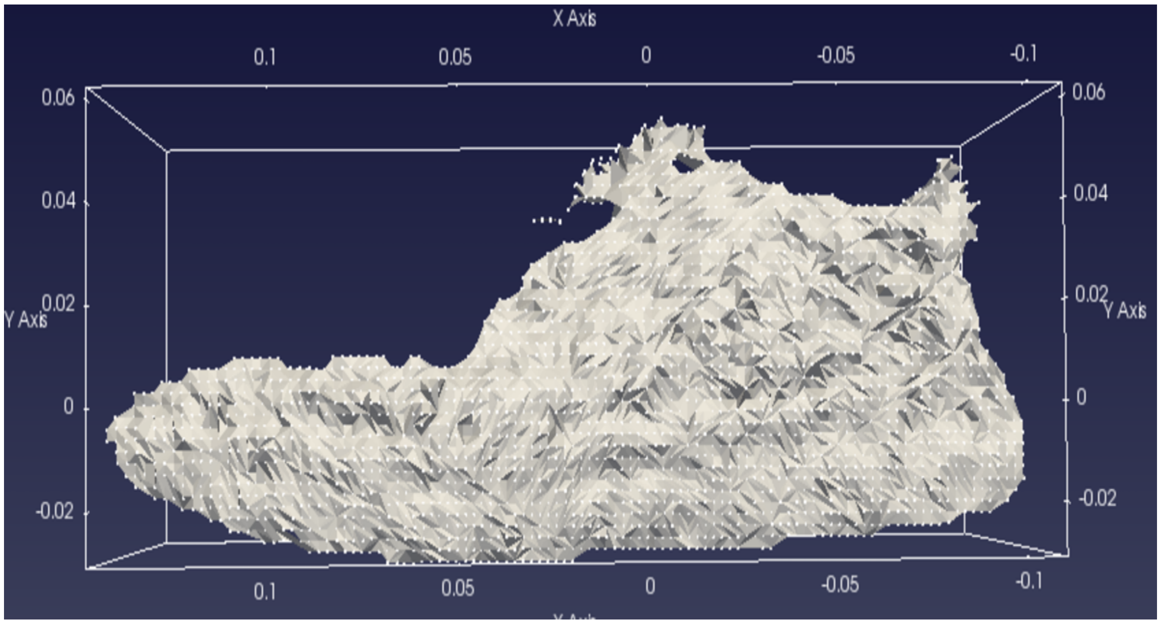


Figure 13.5: Side of shoe captured using improved techniques

13.3 Creating a 3D Object

Once single-surface capture had been perfected, we looked to create a 3-Dimensional model of an object. A single-sided and double-sided approach were taken.

13.3.1 Single-sided Mirroring Approach

The first method of creating the model involved duplicating the capture of one side of a symmetrical object and mirroring the points to create the other side of the object. The points were mirrored by inverting their z-coordinate and translating the points back along the z-axis to align correctly with the first cloud. Figure 13.6 shows the combination of the original cloud and a mirror of the original cloud into a single point cloud to create an object.

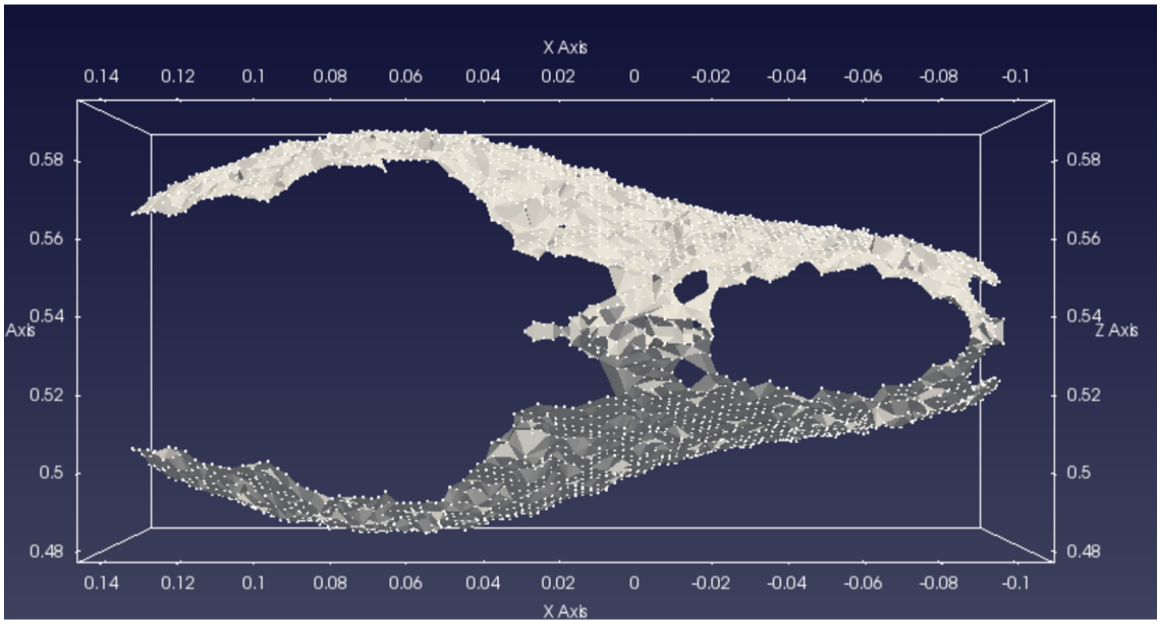


Figure 13.6: Original and mirrored cloud combination

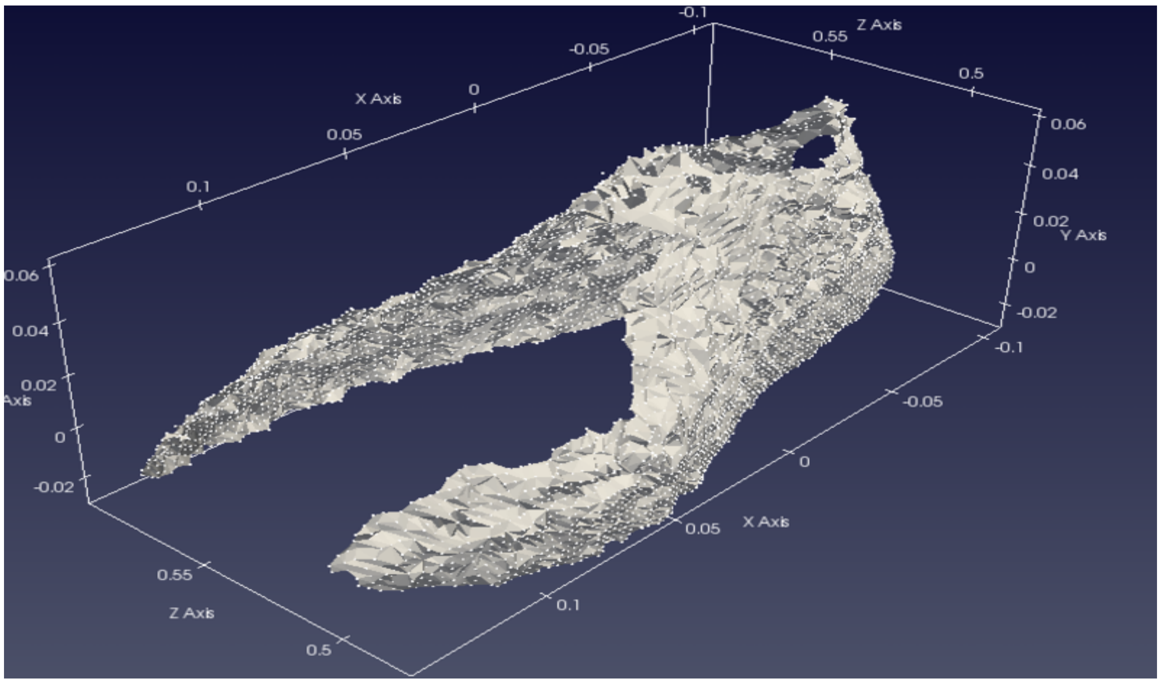


Figure 13.7: Original and mirrored cloud combination(2)

The combined mesh resembled a shoe, but had large holes on the heel, toe, and top of the shoe. This point cloud data could not be easily captured, as a camera aimed at the side of the shoe could not capture its flat top. Combining additional capture perspectives introduced complex mathematical challenges which we felt were beyond the scope of our project.

Despite not being able to capture the top of the shoe directly, we discovered that the surface could be interpolated by using an alternative meshing algorithm: the Poisson Surface Reconstruction algorithm.[1] This meshing algorithm seeks to create watertight meshes for surface and objects. The Poisson meshing algorithm succeeded in filling holes in the mesh, but did so by creating slightly raised and curved surfaces.

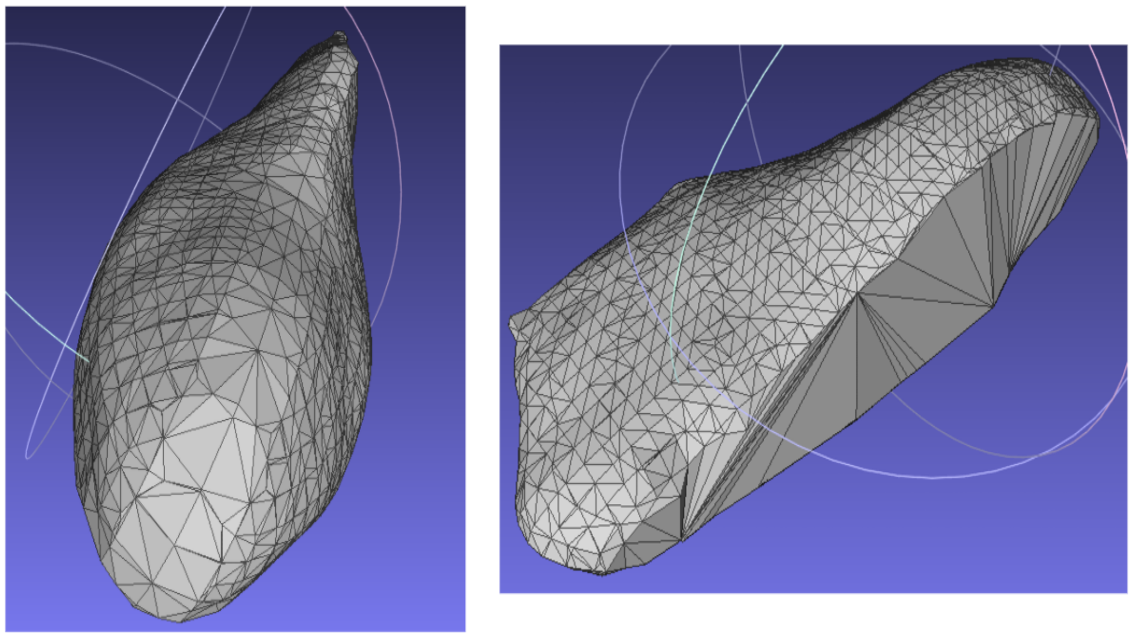


Figure 13.8: Shoe mesh using Poisson Surface Reconstruction

13.3.2 Double-sided Capture Approach

The second method of creating the models involves capturing a face of the model from the top-down perspective, flipping the model to the other side, and capturing the other side from the top-down perspective. These captures were combined using the mirroring method described above. The same mirroring method must be used because both clouds appear very similar as they are captured from the same perspective. This capture method was used on asymmetrical surfaces that required individual captures for each side. The method is demonstrated with a Oakley sunglasses case shown

[1] `http://www.pointclouds.org/assets/icra2012/surface.pdf`

in Figure 13.9.

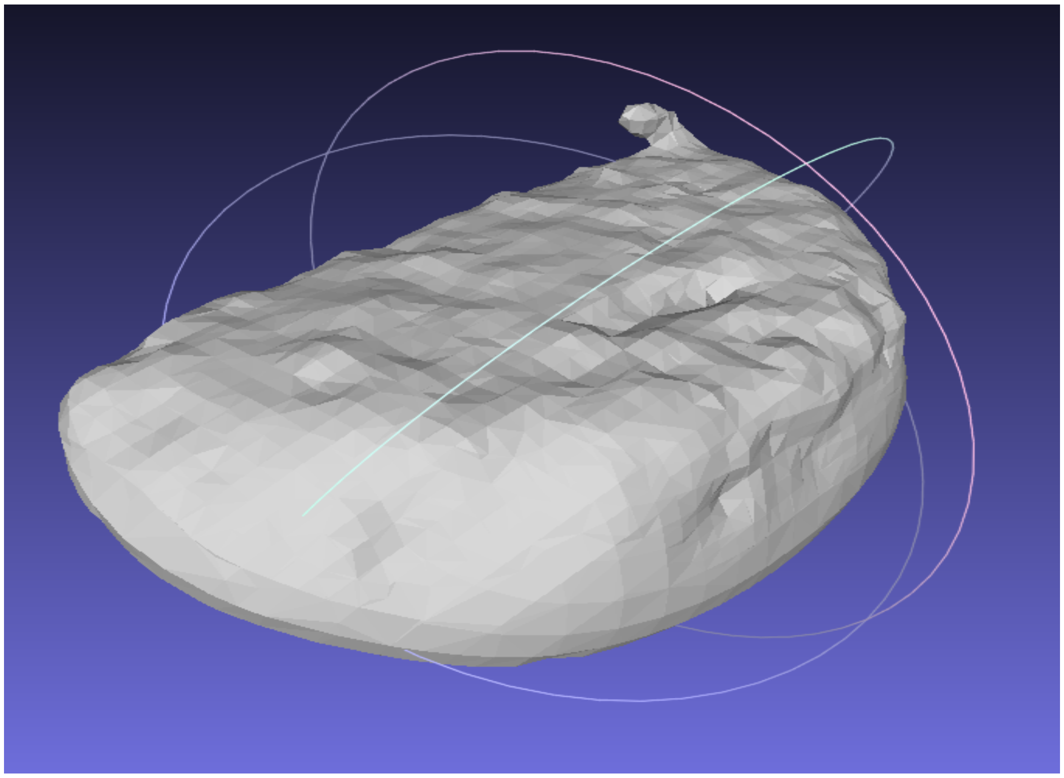


Figure 13.9: Final glasses case mesh

13.4 Benefits of using Point Clouds to Capture 3D Objects

13.4.1 Usability

Throughout development and testing, the usability of the Point Cloud Capture application stood out as a major reason to use point clouds instead of image stitching. Capturing two clouds instead of fifty, capturing from a single angle, and easy elimination of noise and background objects made Point Cloud Capture relatively easy to use when compared with traditional capture methods using image stitching.

13.4.2 Bandwidth

Point Cloud Capture's bandwidth use was often twenty times as efficient as an application sending many photos over a network to be stitched together on a remote server. Each point cloud pair was 2.6 megabytes, compared with 40-90 megabytes for transmission of a set of photos by 123D Catch. To put these numbers in perspective, the 2016 average upload speed for mobile Internet in the United States was 7MBps.[2] A point cloud pair could be sent in just under half a second, while a photo set could be sent in twelve seconds.

13.4.3 Total Processing Time

The largest area of improvement was the time taken to send, combine, optimize, mesh, and send back the point clouds. Autodesk's 123d Catch program often took 20 minutes or more to receive a 3D model, while Point Cloud Capture could receive a model within six seconds. This time was obtained using a combined cloud size of 30,000 XYZ coordinates per-capture.

13.5 Limitations of Point Cloud Capture

13.5.1 Computer Vision in 3D

While computer vision algorithms working in two dimensions have been maturing for many years, three dimensional computer vision is a relatively new and underdeveloped field. Point Cloud Library released version 1.8.0 in June 2016, but development had been declining rapidly since 2012.[3] PCL has not been fine tuned to work with small models and appeared to be better suited to working with point clouds captured from rooms, living spaces, and faades of buildings.

13.5.2 Fixed Viewpoint

Initially, our project sought to present users with an application that could interpret and combine many point clouds taken from different angles around an object to create an accurate model. We imagined users walking around and pointing their phones at objects from a couple angles, casually performing captures. Developing a heuristic for how multiple point clouds should be correctly combined when taken from different perspectives, at different distances, proved to be impossible within the time-frame of our project. The top-down, fixed distance, method of capture was a reliable way of capturing objects with minimally noisy data and consistently correct point cloud combination. Future iterations of point cloud capture software should be able to accurately determine how many faces of an object should come together to recreate the full three dimensional representation.

13.5.3 RGB Support

PCL provides the ability to use RGB (reg/green/blue) color data to enhance point clouds and meshes to show surface color. The Lenovo Phab2 Phone's depth camera was unable to capture RGB data. An optimal digital representation of an object would be a colorized to look realistic. We would

[2] `http://www.speedtest.net/reports/united-states/`
[3] `https://github.com/PointCloudLibrary/pcl/graphs/contributors`

like to see mobile hardware developed with the ability to capture RGB data in addition to XYZ data. Developers would not have to re-color captured objects and users could immediately recognize captures as they appeared in the real world.

13.5.4 Point Resolution

Our tests with 30,000 points took around six seconds for the entire process. Most of this time was processing time on the server. We believe that higher resolution point clouds with 100,000 or more points would not incur a significant penalty to processing time. However, the hardware on the Lenovo Phab2 phone was not able to capture hundreds of thousands of points in its field of view. Meaningful tests would have to be performed with large, real capture data sets to assess whether processing speed would scale linearly with cloud size. We hope to see future mobile hardware with higher depth capture resolution.

Chapter 14

Societal Impact

14.1 Ethical

While our project seems fairly innocuous when used as intended (scanning an object), ethical questions are raised when the app is used to scan a person. A evidenced by the controversy that was raised by the body scanners deployed at airports, people do not want to have their body scanned. If you are scanning someone who you have the consent to scan, then you might worry about the security of the messaging between the app and the server, and you might be worried about what data is stored on the server. To allay some of these concerns, we are not storing any data on the server apart from a log of connections. The bigger ethical question comes from if someone uses our app on a person who they do not have consent from. Realistically, we cannot prevent the app from being used this way, we can only educate people about the possibility of a 3D model of them being captured, something that will become increasingly relevant as 3D camera technology on phones becomes more and more prevalent.

14.2 Social

In the previous section, we focused on some of the negative aspects that could come from our project, so we will examine the positive social impact our app can make. The effect that smartphones and apps have had with regards to social issues stems a lot from democratization. We believe that Tango has the potential to democratize 3D capture if the technology reaches a larger number of phones. Using this technology, more creators will have access to 3D capture, instead of having to rely on expensive and specialized hardware. The hope is that getting more technology into more peoples hands will help those people do more good.

14.3 Political

The political aspects of our project relate to the ethical concerns that we have talked about in previous sections. Laws regarding technology need to keep up with new technologies that reach many people. If the Tango platform reaches more Android devices, then governments may be forced to examine privacy concerns regarding 3D capture. Currently, it takes specialized hardware for 3D capture, so it is not much of a concern. However, in the same way that camera phones put 2D cameras in everyones pocket, Tango (and platforms like Tango) has the potential to make 3D cameras much more widespread, thus requiring privacy laws to be updated to avoid oversights and loopholes.

14.4 Economic

When building this project, our only costs were for the phones that we developed on, and a small cost for the server (AWS). When working on a platform like Android, which has a myriad of devices and versions, applications need to be tested on many different devices. Unfortunately, purchasing all of these phones outright gets expensive, especially as more devices get the Tango hardware. To this end, it is unrealistic economically to develop the application for every device.

14.5 Health and Safety

There are not any health and safety aspects of our project as the software is innocuous, and the sensors only emit harmless infrared light.

14.6 Manufacturability

Realistically this application could be built out a little more to actually create a full application that could be released on the Google Play Store, however, this raises several concerns. Firstly, it might take some time to ensure that we meet all of the guidelines for the application. Secondly, cost issues arise if we gain more users, as the server is one of the cheapest AWS instances we could run. Having more users would require us to upgrade our instance, increasing our monthly costs. Additionally, we would likely not want to monetize our application, so the increase in cost would not be met with an influx of profit. The more realistic approach would be to allow other developers to establish their own backend based on our.

14.7 Environmental Impact

There are not many environmental concerns when developing software, however, there are still some interesting implications about the platform that we are using. For one, as new features like Tango are introduce, old phones may be thrown out prematurely for the newer models. This in turn creates more e-waste, which is difficult to get rid of as the materials have to be dismantled in order for their base materials to be recycled. Additionally, we noticed that the devices that we were working with had poor battery life. This was not due to the size of the battery, but instead due the the firmware and software that was required to run Tango. This means more electricity is required than the average phone to power the sensors and processing that is required when running our application.

14.8 Sustainability

We have already discussed the environmental sustainability concerns, so instead this section will focus on the sustainability of our app itself. Unfortunately, it seems that keeping our app relevant will require constant active development due to changes in the Tango platform and libraries. New devices are being released, which we have not had the opportunity to test our application against. For example, switching from the Tango Development kit to the Lenovo Phab 2 Pro meant that we had to switch everything over from 32-bit to 64-bit. Additionally, new updates to the Tango framework on the phone itself have caused our application to cease to function. Due to all of these factors, our application is not sustainable unless we constantly update it to support the current platform and current devices.

14.9 Usability

We have made strides to make sure that the UI is as simple as possible so that the user does not have many choices when working within the app. For example, the list item will turn green once the item has been fulfilled, allowing the user to at a glance tell what they still have to do. Additionally, the inputting of filenames allows the user to know exactly what their files will be called, instead of relying on some sort of automated naming system based on date and time. However, we do acknowledge that point clouds and how they work is not a simple thing to understand, so we have tried to make it simple to capture, process, and view the point cloud without having to understand how it actually works. In this way, we have created a "black box" of sorts. The only complicated factor is the "distance" measure that the user has to input, which actually feeds our algorithm a standard deviation to base the mesh off of. However, we think it is intuitive enough to figure out what gives you optimal results based on trial and error.

14.10 Lifelong Learning

This project allowed us to explore many different aspects of computer engineering including computer vision, Android, point clouds, Google Tango, and connecting an app with a server. These fields are all very interesting, and contain a lot of depth that we have not even gone into yet. By getting a surface understanding of all of these concepts, we can do a deeper dive into the work that has gone into these fields, as well as improve future projects which might incorporate these technologies. Tango itself is still in its infancy, and we believe that it has a lot of potential to become a staple feature on future Android phones. Getting to develop on this fledgling platform really helped us see how rapidly thing change when a product is being developed, as we continually adapted our application to meet the changing standard for Tango that Google was producing.

14.11 Compassion

While our product may not deal with improving the lives of those in need (it is a niche product after all), we hope we have done something that can improve the lives of people through the advancement of the field. Tango is such a new platform that anybody who creates something with it is probably doing something new, or creating a new approach to an already solved problem. This in turn can then help those who come afterward, benefiting their product. To this end, our code is all public on Github, and we hope that the documentation that we provide allows others to use pieces of our code to build their own applications, and expand the work that we have done with the platform. Because we were beginners working with technologies like Tango and Point Cloud Library, we hope we have created an idea that can be carried on by those who have more knowledge about the technologies and the platform.

Chapter 15

Conclusion

Overall our experience working with the Lenovo Phab 2 Pro, Google Tango and Point Cloud Library has allowed us to understand the state of computer vision with regards to mobile point cloud capture. Based on this experience, there are several conclusions we can draw about both the project, and the tools we chose to use for this project. Firstly, our project was successful in that we were able to produce a fully printable 3D model based on one of our captures with relatively good accuracy. One key factor to the success of this was the ability to remove the background accurately, and we think that this is one of the biggest strengths of Point Cloud Library, and point clouds in general. This has confirmed our beliefs that the best technology for 3D capture is point clouds, instead of image stitching or other approaches. However, mobile capture is still limited by the hardware that currently fits in modern devices. Point clouds resolution will hopefully improve, bringing with it both more points, but also higher accuracy. Additionally, Point Cloud Library was actually not ideal for the actual meshing of the point cloud. While great for background removal, the platform seemed more geared toward working with larger objects like buildings and floor plans as opposed to singular objects.

Because of these factors, we would recommend a few changes be made if someone were to continue or improve our project. Firstly, we would continue to use Tango as it remains the current best mobile capture platform. We believe that because it is a Google product, and because Google produces their own phones, this platform has highest likelihood of reaching mainstream markets. Additionally, because Tango Development is ongoing, the software and hardware will continue to improve.

Secondly, to play off of the strengths of Point Cloud Library, we would recommend that PCL continue to be used to remove backgrounds from images. While PCL was not ideal for the actual meshing, it was great at background removal, and cluster detection. Once the object cluster has been obtained, the meshing should be done by a more specialized platform than PCL. We would also recommend that on the back end, faster hardware be used if more traffic or more data was expected.

In conclusion, computer vision on mobile platforms is still in its infancy. While we were able to successfully combine PCL and Tango, different permutations of front end and back end exist that remain to be explored. Based on our observations and experiences, Tango is primed to become the mainstream 3D vision platform for Android, and should continue to be developed for.

Bibliography

[1] L. Alboul and G. Chliveros. A system for reconstruction from point clouds in 3d: Simplification and mesh representation. In *2010 11th International Conference on Control Automation Robotics Vision*, pages 2301–2306, Dec 2010.

[2] Kobbelt L. Attene M., Campen M. Polygon mesh repairing: An application perspective. *ACM Computing Surveys*, 45:15, 2013.

[3] Htroy-Wheeler F. Dupont F. Nader G., Wang K. Visual contrast sensitivity and discrimination for 3d meshes and their applications. *Computer Graphics Forum*, 35:497–506, 2016.

[4] N. Wongwaen, S. Tiendee, and C. Sinthanayothin. Method of 3d mesh reconstruction from point cloud using elementary vector and geometry analysis. In *2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012)*, volume 1, pages 156–159, June 2012.

Appendix A

Source Code

A.1 Backend

A.1.1 Server: HTTP endpoint

The source code below is a forked implementation of an open source C++ server.[1] The modified portion of the server listens at the /json endpoint to retrieve JSON point cloud data being sent from the phone. After processing, a response is sent as a .obj file inside a JSON object.

```
server.resource["^/json$"]["POST"]=[](shared_ptr<HttpServer::
   Response> response, shared_ptr<HttpServer::Request> request) {
        try {

                // Read the JSON post from the phone/tablet
                ptree pt;
                read_json(request->content, pt);
                //string name=pt.get<string>("firstName")+" "+pt.get
                   <string>("lastName");

                // Extract separate clouds and save to separate
                   files for concatenation
                string front_cloud = pt.get<string>("main_body.front
                   ");
                string back_cloud = pt.get<string>("main_body.back")
                   ;
                float std_dev = pt.get<float>("main_body.std_dev");

                string cloud_path_front = "front.pcd";
                string cloud_path_back = "back.pcd";
                string mesh_path = "mesh";

                std::ofstream out_front("front.pcd");
                std::ofstream out_back("back.pcd");
                out_front << front_cloud;
                out_back << back_cloud;
                out_front.close();
                out_back.close();
```

[1] https://github.com/eidheim/Simple-Web-Server

```
                ProcessCloud(cloud_path_front, cloud_path_back,
                    mesh_path, std_dev, true, false);

                std::ifstream ifs("mesh.obj");
                std::string content((std::istreambuf_iterator<char>(
                    ifs)),(std::istreambuf_iterator<char>()));

                *response << "HTTP/1.1␣200␣OK\r\n"
                          << "Content-Type:␣textl/plain\r\n"
                          << "Content-Length:␣" << content.length() <<
                             "\r\n\r\n"
                          << content;
        }
        catch(exception& e) {
                *response << "HTTP/1.1␣400␣Bad␣Request\r\nContent-
                    Length:␣" << strlen(e.what()) << "\r\n\r\n" << e.
                    what();
        }
};
```

A.1.2 Server: Point cloud processing and meshing

This section of the server code is ran after a client's request has been received and a file is written to the server containing the point clouds to be processed. This is the code for the ProcessCloud function which cleans the point clouds and combines, meshes, and write them to the server as a .obj file which is sent back to the client by the HTTP endpoint code above.

```
//For greedy projection
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/features/normal_3d.h>
#include <pcl/surface/gp3.h>
//#include <pcl/io/vtk_io.h>
#include <pcl/common/transforms.h>
#include <pcl/io/obj_io.h>

//For plane filtering
#include <iostream>
#include <pcl/io/io.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/segmentation/sac_segmentation.h>

//For outlier filtering
#include <pcl/filters/statistical_outlier_removal.h>
#include <pcl/point_types.h>

//For Clustering
#include <pcl/filters/voxel_grid.h>
#include <pcl/features/normal_3d.h>
```

```
#include <pcl/segmentation/conditional_euclidean_clustering.h>
#include <pcl/segmentation/extract_clusters.h>

//Poisson
#include <pcl/filters/filter.h>
#include <pcl/common/common.h>
#include <pcl/features/normal_3d_omp.h>
#include <pcl/surface/mls.h>
#include <pcl/surface/poisson.h>

#include <math.h>

using namespace std;

typedef pcl::PointXYZ PointT;
typedef pcl::PointCloud<PointT> PointCloudT;

PointCloudT ConcatenateClouds (PointCloudT cloud_a, PointCloudT
    cloud_b)
{
        cout << "Concating" << endl;
        PointCloudT cloud_c;
        cloud_c  = cloud_a;
        cloud_c += cloud_b;
        cout << "finsihed␣Concating" << endl;
        return cloud_c;
}

PointCloudT::Ptr FilterPlane (PointCloudT::Ptr cloud)
{

        PointCloudT::Ptr cloud_inliers (new PointCloudT),
                                                cloud_outliers (new
                                                   PointCloudT);

        // Segment the ground
        pcl::ModelCoefficients::Ptr plane (new pcl::
            ModelCoefficients);
        pcl::PointIndices::Ptr            inliers_plane (new pcl::
            PointIndices);
        PointCloudT::Ptr                               cloud_plane (
            new PointCloudT);

        // Make room for a plane equation (ax+by+cz+d=0)
        plane->values.resize (4);

        pcl::SACSegmentation<PointT> seg;
                    // Create the segmentation object
        seg.setOptimizeCoefficients (true);
                    // Optional
        seg.setMethodType (pcl::SAC_RANSAC);
        seg.setModelType (pcl::SACMODEL_PLANE);
        seg.setDistanceThreshold (0.005f);      // Controls how
            strict plane match is
        seg.setInputCloud (cloud);
        seg.segment (*inliers_plane, *plane);

        if (inliers_plane->indices.size () == 0) {
                PCL_ERROR ("Could␣not␣estimate␣a␣planar␣model␣for␣
                   the␣given␣dataset.\n");
                return (cloud);
        }

        // Extract inliers
        pcl::ExtractIndices<PointT> extract;
        extract.setInputCloud (cloud);
        extract.setIndices (inliers_plane);
        extract.setNegative (false);                    // Extract
           the inliers
        extract.filter (*cloud_inliers);                //
           cloud_inliers contains the plane

        // Extract outliers
        //extract.setInputCloud (cloud);                // Already
           done line 52
        //extract.setIndices (inliers);                 // Already
           done line 53
        extract.setNegative (true);                             //
           Extract the outliers
        extract.filter (*cloud_outliers);               //
           cloud_outliers contains everything but the plane

        printf ("Plane␣segmentation␣equation␣[ax+by+cz+d]=0:␣[%3.4f␣
           |␣%3.4f␣|␣%3.4f␣|␣%3.4f]␣␣␣␣␣\t\n",
                        plane->values[0], plane->values[1], plane->
                           values[2] , plane->values[3]);

        return (cloud_outliers);
}

PointCloudT::Ptr RemoveOutliers (PointCloudT::Ptr cloud)
{
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered (new pcl
           ::PointCloud<pcl::PointXYZ>);

        // Create the filtering object
        pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
        sor.setInputCloud (cloud);
        sor.setMeanK (50);
        sor.setStddevMulThresh (0.8);
        sor.filter (*cloud_filtered);
        cout << "Removed␣outliers" << endl;
        return cloud_filtered;
}

PointCloudT::Ptr ExtractLargestCluster (PointCloudT::Ptr cloud)
{
        // Creating the KdTree object for the search method of the
           extraction
        pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::
           search::KdTree<pcl::PointXYZ>);
        tree->setInputCloud (cloud);

        std::vector<pcl::PointIndices> cluster_indices;
        pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
        ec.setClusterTolerance (0.004); // 2cm
        ec.setMinClusterSize (100);
        ec.setMaxClusterSize (25000);
        ec.setSearchMethod (tree);
        ec.setInputCloud (cloud);
        ec.extract (cluster_indices);

  pcl::PCDWriter writer;
        int j = 0;
        int largest_idx = 0;
        int largest_size = 0;
        pcl::PointCloud<pcl::PointXYZ>::Ptr largest_cluster;
        for (std::vector<pcl::PointIndices>::const_iterator it =
           cluster_indices.begin (); it != cluster_indices.end ();
           ++it)
        {
                int cur_size = (*it).indices.size();
                if (cur_size> largest_size){
                        largest_idx = j;
                        largest_size = cur_size;
                }
                j++;
        }

                // For each cluster, add the indexed points from the
                    original cloud to an output cloud
                pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster (
                   new pcl::PointCloud<pcl::PointXYZ>);

                pcl::PointIndices largest_cluster_indicies =
                   cluster_indices[largest_idx];

                for (int i = 0; i < largest_cluster_indicies.indices
                   .size(); i++)
                        cloud_cluster->points.push_back (cloud->
                           points[largest_cluster_indicies.indices[i
                           ]]); //*
                cloud_cluster->width = cloud_cluster->points.size ()
                   ;
                cloud_cluster->height = 1;
                cloud_cluster->is_dense = true;

                // Write the cloud to a PCD file
                std::cout << "PointCloud␣representing␣the␣Cluster:␣"
                    << cloud_cluster->points.size () << "␣data␣
                   points." << std::endl;
                std::stringstream ss;
                ss << "cloud_cluster_" << j << ".pcd";
                writer.write<pcl::PointXYZ> (ss.str (), *
                   cloud_cluster, false); //*

                return cloud_cluster;
}

PointCloudT::Ptr MLSSmooth (PointCloudT::Ptr cloud)
{
        using namespace pcl;
        MovingLeastSquares<PointXYZ, PointXYZ> mls;
        mls.setInputCloud (cloud);
        mls.setSearchRadius (0.01);
        mls.setPolynomialFit ( true );
        mls.setUpsamplingMethod (MovingLeastSquares<PointXYZ,
           PointXYZ>::VOXEL_GRID_DILATION);
  mls.setDilationVoxelSize(0.002);
        mls.setPolynomialOrder (4);

        // Because we are using VOXEL_GRID_DILATION, we dont need
           these parameters
        // mls.setUpsamplingRadius (0.005);
        // mls.setUpsamplingStepSize (0.003);

        PointCloud<PointXYZ>::Ptr cloud_smoothed ( new PointCloud<
           PointXYZ> ());
        std::cout << "MLS␣processing..." << std::endl;
        mls.process (*cloud_smoothed);
        std::cout << "MLS␣processing␣finished" << std::endl;
        return cloud_smoothed;
}

void Poisson (PointCloudT::Ptr cloud, string mesh_path)
{
        using namespace pcl;
        NormalEstimationOMP<PointXYZ, Normal> ne;
        ne.setNumberOfThreads (8);
        ne.setInputCloud (cloud);
        ne.setRadiusSearch (0.005); //.5cm sphere search
        Eigen::Vector4f centroid;
        compute3DCentroid (*cloud, centroid);
        ne.setViewPoint (centroid[0], centroid[1], centroid[2]);
  //ne.setViewPoint (0, 0, 0);

        PointCloud<Normal>::Ptr cloud_normals ( new PointCloud<
           Normal> ());
        std::cout << "Surface␣normals␣processing..." << std::endl;
        ne.compute (*cloud_normals);
        std::cout << "Surface␣normals␣finished" << std::endl;

        PointCloud<PointNormal>::Ptr cloud_smoothed_normals ( new
           PointCloud<PointNormal> ());
        concatenateFields (*cloud, *cloud_normals, *
            cloud_smoothed_normals);

        std::cout << "Poisson␣starting..." << std::endl;
        pcl::Poisson<PointNormal> poisson;
        poisson.setDepth (7);
        poisson.setScale(3.0);
        poisson.setInputCloud(cloud_smoothed_normals);
        PolygonMesh mesh;
        poisson.reconstruct (mesh);
        std::cout << "Poisson␣finished..." << std::endl;

  pcl::io::saveOBJFile (mesh_path + ".obj", mesh);
  //pcl::io::saveVTKFile (mesh_path + ".vtk", mesh);
        return;
}

PointCloudT::Ptr GenerateSecondCloud(PointCloudT::Ptr cloud, string
    cloud_path_back, float std_dev_mul, bool single_cloud){
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_back (new pcl::
            PointCloud<pcl::PointXYZ>);
        if(single_cloud)
                *cloud_back = *cloud;
        else{
                // Load input file into a PointCloud<T> with an
                    appropriate type
                pcl::PCLPointCloud2 cloud_blob;
                //pcl::io::loadPCDFile ("point_clouds/pc_back1.pcd",
                     cloud_blob);
                pcl::io::loadPCDFile (cloud_path_back, cloud_blob);
                pcl::fromPCLPointCloud2 (cloud_blob, *cloud_back);

                // Eliminate a plane from the cloud
                cloud_back = FilterPlane(cloud_back);
                // Remove any remaining outliers
                cloud_back = RemoveOutliers(cloud_back);
                // Cluster extraction to find biggest object
                cloud_back = ExtractLargestCluster(cloud_back);
        }

        float front_max_z = 0;
        float back_min_z = 0;
        float avg = 0;

        int num_points = cloud_back->points.size();

        for(size_t i = 0; i < cloud->points.size(); i++){
                float cur_z = cloud->points[i].z;
                if(cur_z > front_max_z){
                        front_max_z = cur_z;
                }
        }

        for(size_t i = 0; i < num_points; i++){
                float cur_z = cloud_back->points[i].z;
                // Invert the z
                cloud_back->points[i].z = -cur_z;
                // Gather info for average
                avg += cur_z;
                // Gather info for max
                if(-cur_z < back_min_z){
                        back_min_z = -cur_z;
                }
        }
        avg /= num_points;

        float std_dev = 0;
        for(size_t i = 0; i < num_points; i++){
                float cur_z = pow((-cloud_back->points[i].z - avg),
                   2);
                std_dev += cur_z;
        }
        std_dev /= num_points;
        std_dev = sqrt(std_dev);

        // Translate the points back up to match the original cloud
        for(size_t i = 0; i < num_points; i++){
                cloud_back->points[i].z += (2 * avg) + (std_dev_mul
                   * std_dev);
        }

        return cloud_back;
}


void ProcessCloud(string cloud_path_front, string cloud_path_back,
   string mesh_path, float std_dev, bool poisson, bool single_cloud)
{
                // Load input file into a PointCloud<T> with an
                   appropriate type
                pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::
                   PointCloud<pcl::PointXYZ>);
                pcl::PCLPointCloud2 cloud_blob;
                //pcl::io::loadPCDFile ("point_clouds/pc_back1.pcd",
                    cloud_blob);
                pcl::io::loadPCDFile (cloud_path_front, cloud_blob);
                pcl::fromPCLPointCloud2 (cloud_blob, *cloud);
                //* the data should be available in cloud

                // Eliminate a plane from the cloud
                cloud = FilterPlane(cloud);
                // Remove any remaining outliers
                cloud = RemoveOutliers(cloud);
                // Cluster extraction to find biggest object
                cloud = ExtractLargestCluster(cloud);

                //Smooth the cloud
                cloud = MLSSmooth(cloud);
```

```
/*
        FOR BACK CLOUD ONLY
        Take the front cloud, invert its coordinates
           , and reposition it to align
        with the original cloud
*/
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_back (new
   pcl::PointCloud<pcl::PointXYZ>);

cloud_back = GenerateSecondCloud(cloud,
   cloud_path_back, std_dev, single_cloud);

// Combine the front and back clouds in one cloud
*cloud = ConcatenateClouds(*cloud, *cloud_back);

// Use poisson meshing if applicable
if(poisson){
        Poisson(cloud, mesh_path);
        return;
}

cloud = RemoveOutliers(cloud);

// Rotate the cloud a number of degrees about the x
   axis to account for varying camera angle
// Dont transform, only rotate (use identity
   transformation matrix)
// Eigen::Affine3f transform (Eigen::Affine3f::
   Identity());
// // Rotate 20 degrees for test
// Eigen::Matrix3f rotation (Eigen::AngleAxisf
   ((20.0*M_PI) / 180, Eigen::Vector3f::UnitX()));
// transform.rotate(rotation);
// pcl::transformPointCloud(*cloud, *cloud,
   transform);
// std::cout << transform.matrix() << std::endl <<
   std::endl;


// Normal estimation*
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> n;
pcl::PointCloud<pcl::Normal>::Ptr normals (new pcl::
   PointCloud<pcl::Normal>);
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new
   pcl::search::KdTree<pcl::PointXYZ>);
tree->setInputCloud (cloud);
n.setInputCloud (cloud);
n.setSearchMethod (tree);
n.setKSearch (20);
n.compute (*normals);
//* normals should not contain the point normals +
   surface curvatures
// Concatenate the XYZ and normal fields*
```

```
                pcl::PointCloud<pcl::PointNormal>::Ptr
                    cloud_with_normals (new pcl::PointCloud<pcl::
                    PointNormal>);
                pcl::concatenateFields (*cloud, *normals, *
                    cloud_with_normals);
                //* cloud_with_normals = cloud + normals

                // Create search tree*
                pcl::search::KdTree<pcl::PointNormal>::Ptr tree2 (
                    new pcl::search::KdTree<pcl::PointNormal>);
                tree2->setInputCloud (cloud_with_normals);

                // Initialize objects
                pcl::GreedyProjectionTriangulation<pcl::PointNormal>
                     gp3;
                pcl::PolygonMesh triangles;

                // Set the maximum distance between connected points
                     (maximum edge length)
                gp3.setSearchRadius (1.0); // Increased from .025

                // Set typical values for the parameters
                gp3.setMu (2.5);
                gp3.setMaximumNearestNeighbors (100); // Decreased
                    from 500
                gp3.setMaximumSurfaceAngle(M_PI/2); // 90 degrees
                    increased from default 45
                gp3.setMinimumAngle(M_PI/18); // 10 degrees
                gp3.setMaximumAngle(2*M_PI/3); // 120 degrees
                gp3.setNormalConsistency(false);

                // Get result
                gp3.setInputCloud (cloud_with_normals);
                gp3.setSearchMethod (tree2);
                gp3.reconstruct (triangles);

                // Additional vertex information
                std::vector<int> parts = gp3.getPartIDs();
                std::vector<int> states = gp3.getPointStates();

    //pcl::io::saveVTKFile ("meshes/flat_mesh_back1.vtk", triangles)
        ;
    pcl::io::saveOBJFile (mesh_path + ".obj", triangles);
    //pcl::io::saveVTKFile (mesh_path + ".vtk", triangles);
                // Finish
                return;
}
```

A.2 Repository

All source code is available at `https://github.com/jcallin/Senior-Design-2017-Backend` and `https://github.com/max2dn/PointCloudBuilder`