18551 Spring 2017
3D Object Reconstruction

Group 3: David Buzzell, Pragna Mannam,
David Zhou

Introduction

Background

In researching this project idea, we came across a current Project Tango app *Wayfair View*, which does object reconstruction based on current furniture models, but then inserts them into the current environment (augmented reality). We wish to focus on generalized models that can be used for any application, eliminating the use for the physical object. In addition, there are a few previous 551 projects that have similar aspects to our proposed project: Big Brother Ain't No Cyclops (Group #11, Fall 2006 - creates full face models using only partial information), Get in my Belly! (Group #6, Fall 2008 - creates a 3D model of an organ from medical images).

Problem

2D images or textual descriptions of objects provide a very limiting description of the object. 3D object reconstruction is so detailed that it could eliminate the need to have the physical object, which has many different applications. The model would provide sufficient information such as scale, color, texture, etc. The most practical application of having a model of a physical object is for measurement. Knowing the real-life scale of an object can help you place it into new environments without guessing. Specifically, one possible application for 3D object reconstruction is to create a model for a piece of furniture you are considering purchasing. Inserting the furniture into an image of your home can give you an idea of how well it fits in your home, aesthetically and literally. Other applications include interior design, modeling larger

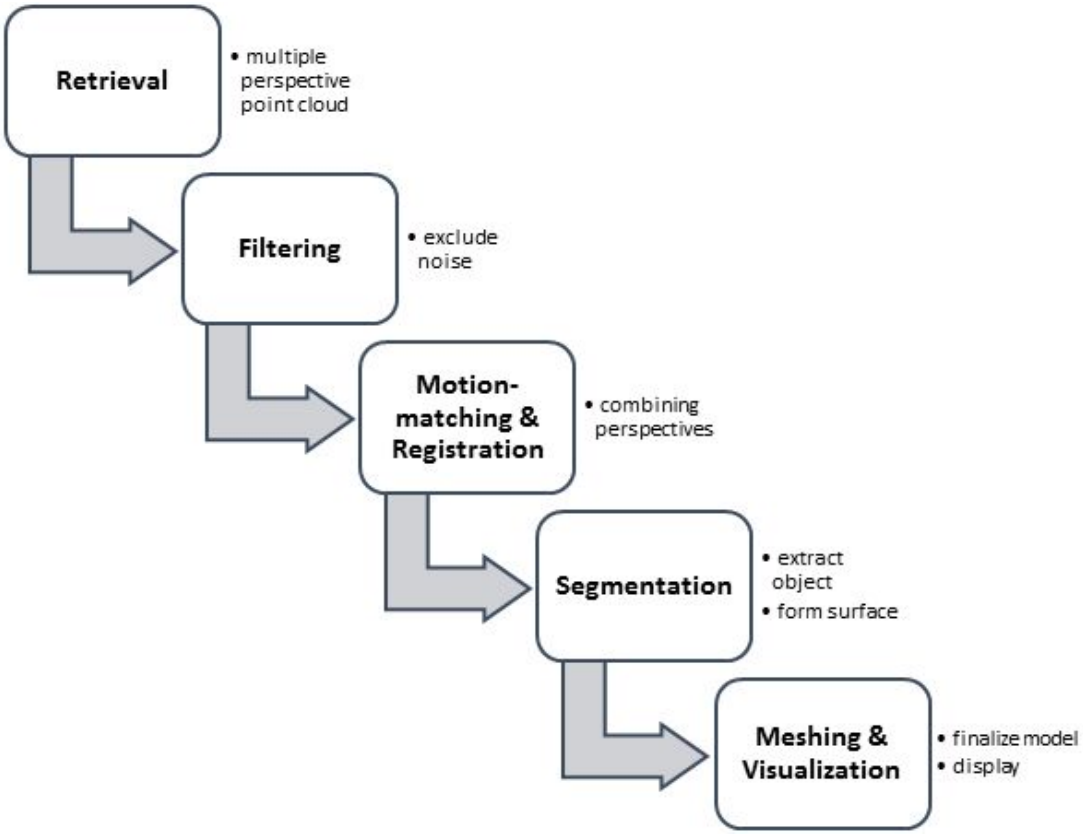
scale systems, and providing a CAD-style overview for designers. Our project involves creating 3D models from real-life objects.

Solution

We will be taking point cloud data of objects and building models from them. This creates an easy and intuitive way to create 3D models, which will be more efficient than tediously modeling an object using softwares like CAD. Project Tango provides guides and toolkits for motion tracking, area learning, and depth perception, as well as advanced topics. On top of creating a point cloud, we will use information to recreate the objects.

Algorithms

Signal Flow



Point Cloud Retrieval

To begin this project, we first collected the 3D depth information of our whole environment. We used a Project Tango tablet with 3D camera capabilities to record this depth information in a point cloud format. We set up our room environment so that it is as uncluttered as possible, setting up the object exactly 1.5 meters away from the tablet and directly in the center of the Lazy Susan, both on a level table in the room (see Figure 1).

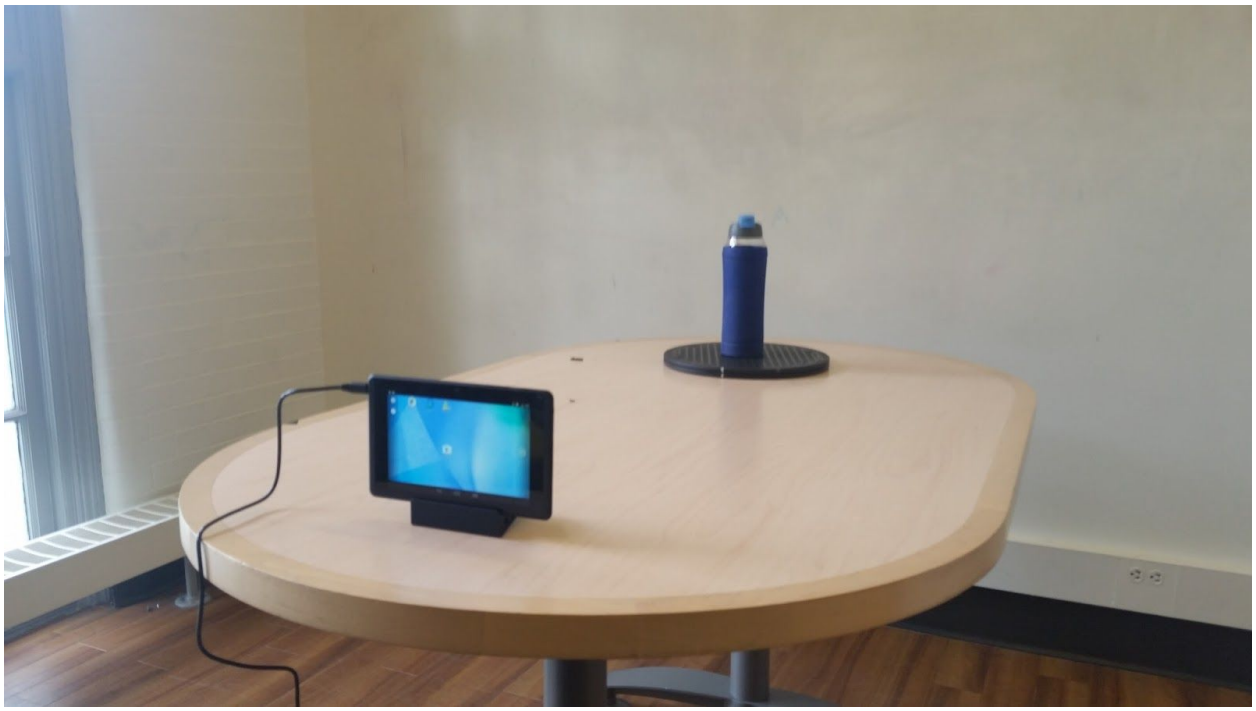


Figure 1: Hardware setup for PC extraction

While Tango API can perform depth perception calculations by using its 3D depth technologies from different views in order to create a thorough 3D model, we used the wrapper app, Voxxlr, as our method of collection (due to the abstractions away from Android File I/O issues, which are explained later). Recording at 0.5 voxel resolution (where voxel is the 3D space equivalent to pixel resolution) and at 2 seconds per voxel recording rate, the app is able to record 3D floating-point coordinates (x, y, z) along with RGB color data relating to each visible point from the Project Tango tablet's cameras. This point cloud is then uploaded to Voxxlr's servers, which we can then download the point cloud as a .PLY file (a common file format for point clouds as well as .PCD file) on a laptop running the appropriate algorithms detailed below.

Filtering

The 3D point cloud data collected by the Tango device could have significant noise. We tried to limit the amount of noise and background by collecting the data in a room with white walls and cleared all other objects out of the room. The test object was put on a table. However, measurement errors from the Tango device are relatively large. Those noise and errors could lead to erroneous results in our future steps. So we implemented filtering to remove outliers from our point cloud data. Algorithmically, we used Point Cloud Library (PCL) to compute the distribution of point-to-neighbor distances in the input dataset. We iterated over all the points and computed the mean of distance to neighbor for each point. Assuming the distribution of mean distances is Gaussian, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset.

Registration

There are two methods of doing registration using the Tango device. The first one is to take snapshots of a test object at different angles and try to stitch all the views together. The other method is to execute registration during data collection. Due to time constraints and complexity, we used both methods in our project.

For the first method of stitching snapshots from different angles of an object, we set up the data collection phase by putting the test object on a Lazy Susan (with various angles being marked). The Tango device is located on a fixed point, and the distance from the Tango device to the center of the the tray being fixed. The next step is to take snapshots of point cloud data using

the tango device. In this step, the object is examined usually at 0°, 90°, 180°, and 270°. At each angle, multiple snapshots are taken because the Tango device can only collect a relatively sparse point cloud using this snapshot method. Registration is then done on all the point clouds that are taken at the same angle. This enriches the point clouds. Consequently, we have 4 relatively high quality point clouds at 4 different angles. Then, we inspect the data and shift the center of all the point clouds to the origin of the coordinate system. A rotational transformation is applied to all the points in the point clouds. The last step is then to perform the Iterative Closest Point algorithm to achieve the final result. The algorithm takes two point clouds as input. One of the point cloud is considered the target point cloud, which is fixed. The other point cloud is considered the source. The goal of the algorithm is to create a matrix transformation so that the source point cloud can match closer to the target point cloud. From each point in the source point cloud, a mapping will be created between that point and the closest point in the target point cloud. If a point in the source point cloud does not have a closest point in the target point cloud within certain distance, the point will be neglected for mapping purpose. Next, find the optimized rotation transformation for the source point cloud by minimizing root mean square point to point distance after the transformation. Lastly, the optimal transformation is applied to the source point cloud. These steps are repeated until two clouds are close enough to each other.

We also utilized the registration capabilities of Voxxlr app as fallback option for our registration operation. The Voxxlr app collects point cloud data in a non-real-time fashion. On startup, Voxxlr creates a point cloud snapshot, and using the motion tracking capabilities of the Tango device, it collects information about the device's movement and applies a transformation to the point cloud data collected afterwards so that all point cloud data is placed in a coordinate

system based on Tango's camera's perspective. By doing so, the Voxxlr app allows us to create point clouds of 360 ° of any object in a scene. When collecting data utilizing this innate registration capabilities, we no longer need to set up the Lazy Susan and measure the distance manually for registration.

Segmentation

After experimentation with segmentation methods like edge detection and plane model segmentation, we settled on Euclidean cluster extraction. The algorithm for Euclidean clustering organizes a point cloud as a Kd-tree. For each point, we add points close by to a queue. Once all the appropriate points are added to the queue, we deem that the points in the queue form a cluster. Pivotal parameters to choose the points considered to be a part of a cluster are distance threshold, minimum and maximum cluster size, and maximum iterations [1].

For some objects the clustering extraction was not good enough to distinguish planes from objects. However, the water bottle, shoe box, and the shoebox segmented tremendously well using Euclidean clustering extraction. Therefore we finalized on this method of segmentation.

Meshing and Visualization

The fast triangulation method was the only surface method we tested due to time constraints. We tested this method on a variety of shapes of objects. For a cylindrical object like a water bottle, triangles are sub-optimal at recreating the rounded surface. However, it performs slightly better for a rectangular object like a shoe box. Furthermore, a more complex shape like a

backpack surface is recreated relatively well with fast triangulation. Across all object fast triangulation surface recreation worked decently. Without an interface to Tango through Android Studio, we had to visualize the models on a laptop through Visual Studio.

Parameters used to adjust the triangulation to specific objects include maximum nearest neighbors, search radius, and minimum and maximum angles per triangle. For small objects like the water bottle, we set the maximum nearest neighbors to smaller values like 100 points. Large objects like the backpack required 500, which caused latency issues (12 minutes) in mesh model rendering, which took the longest time in the data flow.

Code Hierarchy

Data Flow

Using the Tango Tablet, we collected depth and color information using Voxxlr. Voxxlr is a point cloud data collection app that uses Tango Core. Then Voxxlr uploads the point cloud data to its server, which we can access on a laptop. Using Point Cloud Library resources, we implement the object reconstruction algorithms on a PC in Visual Studio.

Android Environment (Java / C++ for NDK)

Originally, we were using Tango API directly in the Android environment to extract the point cloud. We used the developer's project *cpp_point_cloud_example* as a starting point for being able to view the point cloud in real-time on the Tango Tablet screen. Modifying some functions in the Java app and C++ NDK code, we then built a "snapshot" capability that allowed us to capture the current point cloud and save it to a file. Unfortunately, the file I/O process did

not perform accurately enough for our application, driving us to switch to the Voxxlr app for point cloud data collection and retrieval.

Once we got to the demo, we did end up building an Android app (purely in Java) that allowed viewers to guess which models matched the point clouds we extracted. Due to time and the complexity of displaying 3D mesh models inside an Android app, we left these models has picture snapshots that we took after fulling building the model inside a laptop running PCL and the fast triangular meshing algorithm.

PCL (C++ in Visual Studio)

Once collected, the point cloud will be processed using PCL. Based on the methods used for registration (mentioned in the *Registration* subsection of *Algorithm*), we go through two slightly different procedures. If we are doing registration from scratch, the first step is to visualize and inspect the raw point cloud. However, if data is being collected using the registration technology of Voxxlr, the preprocessing steps will not be necessary.

Preprocessing:

The point cloud will likely have some offset away from the origin. Thus, it is necessary to manually adjust the point clouds to the origin of the coordinate system. (Note that in point cloud collected by Tango, x is the horizontal direction, y is the vertical direction, and z is the direction from the camera towards the object.) We then use a cloud moving module to shift the point cloud so that it stays roughly around the origin point of the coordinate system. Afterwards, we eliminate any points outside of a certain radius (usually predefined based on the dimensions of

the test object). As described in the *Registration* section, at this point, we should have several point clouds of the object taken from different perspectives. The next step is to rotate the point clouds taken from multiple views to the proper angle, and run the registration algorithm. At the end, we filtered out any outliers in the resulting point cloud.

Modeling:

When 360° data is collected by Voxxlr directly, we need to apply the Euclidean cluster extraction algorithm to segment the actual test object from the table, walls or any other background object. Once this is done, depending on the nature of the object, we sometimes choose to do another round of filtering using statistical outlier removal (in some cases this could result in worse models). The last step is to run the meshing algorithm to ensure the model is generated.

Demo

Interactive Data Collection

The first part of our demo consisted of a live data collection session. We had a setup of an object on a Lazy Susan exactly a certain distance away from the Tango running the Voxxlr app. People were able to collect 3D raw point cloud data of anything (as Voxxlr is better at collecting landscape data). Then we collected data from all 360-degree perspectives of an object throughout the demo session to simulate our data collection process to passer-byers.

Demo App

Our demo was two-fold. We wanted to let people see how we recorded the point clouds with our setup and interact with the models in a game format. First, we had viewers try out the Voxxlr app, get accustomed to how it records data, and try it on our table setup with the object on a Lazy Susan. Once the recording was complete and they uploaded the file to Voxxlr's servers, we then downloaded the file and let them explore their recording in Voxxlr's online viewer. This way, people could actually visualize the amount of 3D work we collected when making these models and how visually accurate the information was to the actual environment they recorded.

For the second part of our demo, we wanted people to understand the mesh models that we created and how we got them from the original point cloud file. Thus, we built a quick app purely in Java to be used on our other Android device that allows users to guess with model was made from which point cloud. For the models best designed using our meshing algorithm, most

viewers were able to correctly identify the mesh model (i.e. shoe box, backpack). However, the only one most viewers labelled incorrectly was the object not designed for this particular meshing algorithm (i.e. water bottle), thus defending the fast triangulation meshing algorithm's effectiveness.

Results

Successes

Despite our fallbacks, we were able to build relatively realistic 3D models on a laptop. During our demo game where people guessed what the models represent, most people were successful. This shows that our models were representative and achieved the basic requirement of 3D object reconstruction. The segmented mesh models of a water bottle, shoe box, and backpack are shown below.

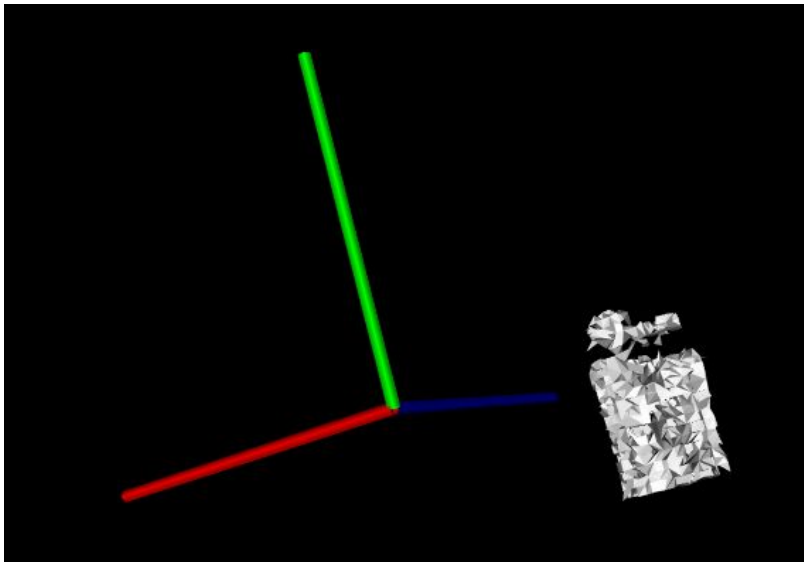


Figure 2: Mesh Model of the Water Bottle

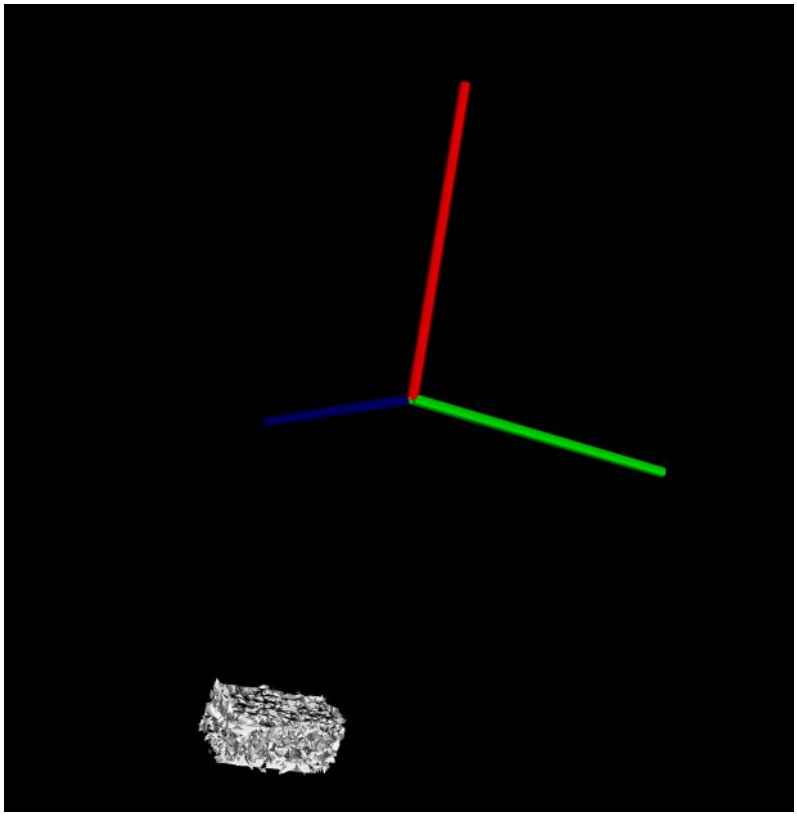


Figure 3: Mesh Model of the Shoe Box

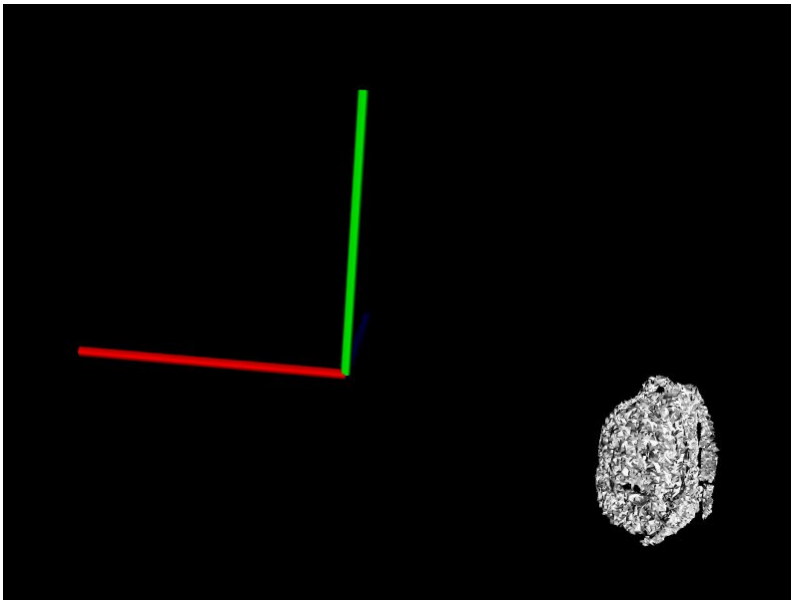


Figure 4: Mesh Model of the Backpack

Limitations and Fallbacks

Our largest limitation in implementing this project and by far the largest time-drain was ironically the conceptually least difficult part of this project. For some reason (as seen and reported by other fellow 551 groups as well), performing file output operations on an Android device is near impossible to perform with 100% accuracy and no bugs.

Firstly, we were using the C++ library to extract the point cloud from the Tango API (originally we had plans to compile PCL on our Tango tablet as well, thus perform all of the necessary calculations on the tablet, all in C++). However, none of the file operations in Android NDK (the framework for supporting C++ in Android environments) are fully supported, and nearly all documentation suggests using Java for file output functions. This required us to now send the point cloud struct through the NDK/Java boundary, which is not designed for large structures of data (like a 17000-point point cloud) and thus has inaccurate performance. Furthermore, simply writing a text file to the Android device is not as simple as with other file systems. Writing to an external storage device (i.e. an SD card) is manufacturer specific, thus cannot be written as a general purpose function in Android. Even writing to the internal storage is difficult, because of the USB indexing that happens. In order to view a file on the Android device, you must reset the USB indexing function that accesses the Android device, which is not supported in Android Studio (other than reconnecting the device every time you write a file). And even after all of that, the file would only write to the internal storage about 30% of the time, even after Android Studio reported a “successful file-write operation”. The whole process was very frustrating, and with very little known solutions available online, moving the point cloud

retrieval process to the Voxxlr service was one of the smartest and time-saving moves our team made.

In the Registration step of our project, we were able to achieve partial registration using point clouds taken at the same angle to enrich snapshots of point clouds as shown in Figure 5 and 6. However, we were not able to create a fully registered model from all angles. The tricky part of registration is that it requires all the points to be roughly taken at the same view point. Meaning that all point clouds should share one coordinate system. Since the measurement noise created by using Tango, it is very difficult to align all the clouds without the precise motion tracking data. Since we failed to implement PCL on Android, it is extremely difficult to refine the errors and noise in the point cloud data so that they can align perfectly after registration. In addition, in our experimental setting, the object is located on a Lazy Susan from a fixed distance away from the Tango device. This means that we have to eliminate the background before rotating and aligning the point clouds. Otherwise, after rotation on the Lazy Susan, the resulting point cloud would consist of overlaps of point clouds that didn't appear in real life. A possible solution is to only fix the position of the test object and rotate the Tango device around the test object at fixed distance. However, that will also require rotation and alignments of the cloud since each snapshot of point cloud will be taken with a new coordinate.

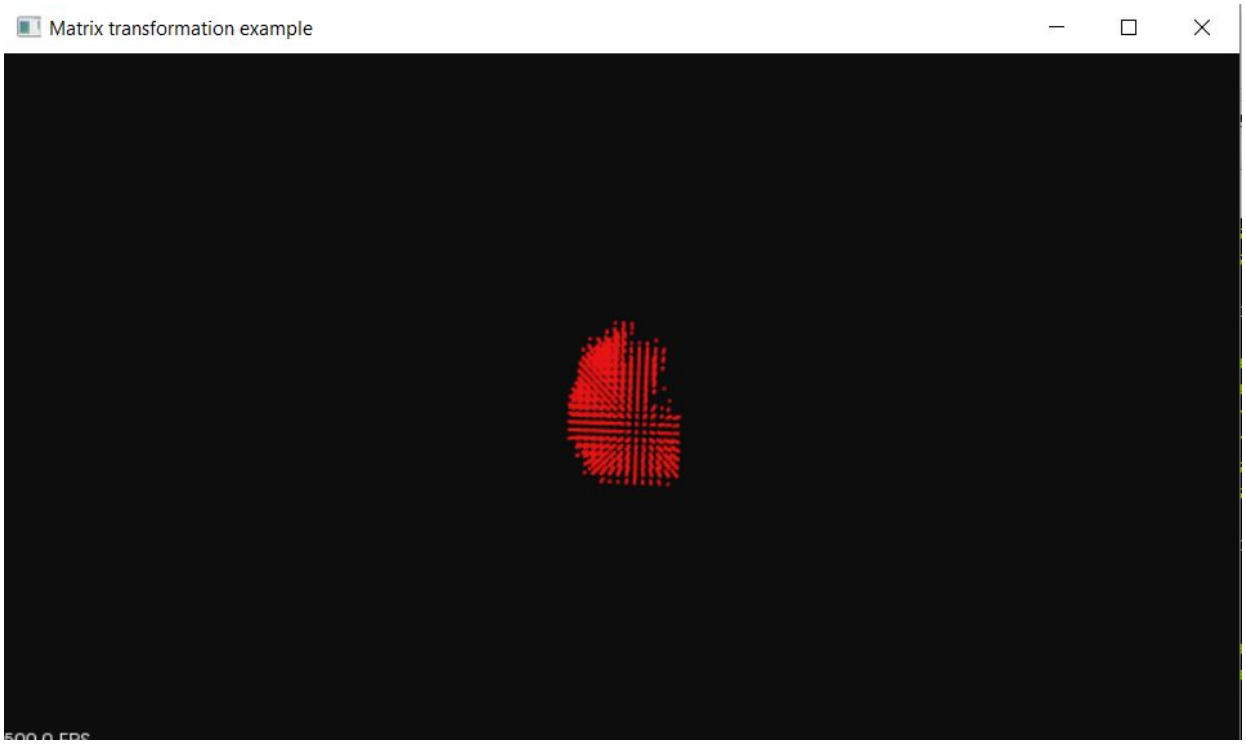


Figure 5: Top-down view of point cloud achieved by registering snapshots of point clouds from the left side of the water bottle.

Figure 6: Horizontal view of point cloud achieved by registering snapshots of point clouds from the left side of the water bottle.

The filtering method also has limitations. For more complex objects, like the backpack, the statistical outlier removal algorithm removes plenty of the meaningful data points, which results in a lesser quality mesh model. Another minor drawback of our project is the modeling algorithm. The fast triangularization method that we chose performed well on shapes with sharp edges (such as the shoe box) and complex-shaped objects (backpack) but not objects with curves (like the water bottle). The reason for this trend is that triangles do not closely represent circles to form curves surfaces.

Feedback

Mid-Project Oral Report

At the end of the mid-project oral report, we received several concerns about how we were going to capture our point cloud data and stitch it together properly and easily. Upon suggestion, we purchased a Lazy Susan, a turntable device that allows us to rotate the object accurately and easily by marking the desired rotations on the Lazy Susan itself (see picture below). We only needed the Lazy Susan for the recording methods involving our own registration functions, so when we used Voxxlr's registration method as a backup the Lazy Susan was cut from the recording setup.

Since our app was recording in real-time originally, it was also suggested to use a motor that would automatically rotate the Lazy Susan at a desired speed for recording. Once we switched to a non real-time recording solution (Voxxlr), the need for the motor was no longer necessary.

Figure 5: Our Lazy Susan with appropriate markings for rotation (0°, 90°, 180°, 270°)

Final Oral Presentation

During our final oral presentation, we received concerns about our demo format and our progress. Suggestions were made that we should display partial models of registration, segmentation and meshing. We were able to successfully display how we can segment models from a 360-degree-view point cloud as well as our capability to create rough models of the objects. During demo session we had models of a water bottle, a shoe box, and a backpack being displayed in our demo app.

Demo Day

The feedback we received on demo day was concerning our registration efforts. While Voxxlr performed its own filtering and registration, we were making efforts on our own that were not ready by demo day. We made sure that we would be finished by the final report to remedy this.

Schedule

Tasks	**Assigned to**	**Deadline**
Getting a Point Cloud	Buzzell	2/10 - 2/26
Importing/Compiling PCL	David Z.	2/25 - 3/5
Filtering Point Cloud	David Z.	3/6 - 3/12
Motion-matching - On sample data - Real objects	David Z. & Buzzell & Pragna	3/13 - 3/19
Registration	David Z. & Buzzell & Pragna	3/20 - 3/26
3D Object Segmentation - Euclidean clustering	Pragna	4/9 - 4/15
Object Model Rendering	Pragna	4/16 - 4/23
Visualization	Buzzell	4/23 - 4/29
Interacting with Model/ Prepare Demo	David Z. & Buzzell & Pragna	4/30 - 5/4

Future Work

As stated before, having our largest time hangup in Android File I/O was something very unexpected and difficult to remedy so late in the project. It's possible that with more time that we could have figured out all of the issues with Android File I/O, but it's more reasonable to say that we could have had more time for other work had we used an app like Voxxlr earlier on in our project phase. Also, integrating PCL with Android would have been a possibility if we got File I/O working better at the end.

According to documentations by PCL, PCL registration is aimed to align two point clouds that are relatively close to each other. Since it essentially tries to find the optimized rotational transformation of the original object that minimizes points with smallest distance. There are couple ways to improve the quality of registration. The first of all is to include RGB data in the point cloud. By doing so, each point in the point cloud will essentially be points in a 6 dimensional space. In the Iterative Closest Point algorithm will have more accurate point matching between two clouds. Another way of improvement is to include more background data in each shot, and match data points using key point detection algorithms such as SIFT to create point mapping between two point clouds.

While the fast triangulation meshing technique was more effective on complex objects like the backpack, we could experiment with more meshing techniques that could seamlessly create surfaces for different types of objects or try shapes other than triangles.

Acknowledgements

We would like to thank Professor Sullivan and Savvides for their support and guidance. Furthermore, Markus helped us overcome hurdles in our project.

References

Bibliography

[1] "Euclidean Cluster Extraction." *Documentation - Point Cloud Library (PCL)*. Point Cloud Library, n.d. Web. 10 May 2017. <http://pointclouds.org/documentation/tutorials/cluster_extraction.php#cluster-extraction>.

[2] "Fast triangulation of unordered point clouds." *Documentation - Point Cloud Library (PCL)*. Point Cloud Library, n.d. Web. 10 May 2017. <http://pointclouds.org/documentation/tutorials/greedy_projection.php>.

[3] “Getting Started with the Tango API.” *Tango C API*. Google Developers, n.d. Web. 10 May 2017. <https://developers.google.com/tango/apis/c/>.

Appendix

Appendix A: Project Tango API

As discussed earlier in the report, our original project conception directly used the Tango API in order to capture point clouds and output to the file for PCL to use. This is a cursory guide to how to setup this section of the project, with a lot of hours of documentation-reading and long hours of debugging going into this expertise.

First configure the Tango API code package for your device. The guide we used is listed here, as created by the Google Developers page: <https://developers.google.com/tango/apis/c/>. We used the C++ examples for our project, but there is also listed support for the Java and Unity versions on this site as well.

Once configured, follow the instructions on that site to load up the *cpp_point_cloud_example* (should require an import from VCS in Android Studio). There's a large amount of files in this project, but here is the main files that you will need to edit:

- PointcloudActivity.java - the Java file for your GUI (tells the button/view what to display/activate)
- TangoJNINative.java - the Java wrapper file that allows you to call C++ functions in your Java activities

- jni_interface.cc - the C++ wrapper file that defines the parameters/return values of your Java function call and what C++ functions to call to determine the return values
- point_cloud_app.cc - the C++ file for all of the Tango functions (such as TangoPointCloud types)
- AndroidManifest.xml - the definitions file that you will need to check and update in order to configure Tango and File I/O properly for your app

I recommend reading through the other files to get a general idea of how information is passed between the two interfaces (Java and NDK). A lot of documentation and online forums (i.e. Stack Overflow) will be necessary to provide correct and helpful solutions (as many solutions tended to contradict each other). Oh and when in doubt, restart Android Studio (seriously, this software is so buggy that sometimes that's the only thing that was wrong with your project).

Appendix B: Voxxlr

To contrast, Voxxlr is super easy to work with and requires very little explanation. At the time of writing this report, Voxxlr is a beta app released for certain Project Tango devices (i.e. designed for our Project Tango tablet but could not run on our newer Lenovo Phab) that works on top of the Tango API to record point clouds and save them in the convenient .PLY extension.

First, download the app from the Google Play Store (requires an active Google account). Then, register for a Voxxlr account (allows for Google account signup, for your convenience). Once registered, you can now record and save your point clouds to your account on Voxxlr, viewable

on any computer device with a decent internet browser. Under their free plan, you can upload a limited amount of point clouds and can have only so many people viewing it at one given time due to bandwidth caps; however, keep in mind that Voxxlr is designed for very large scale projects (i.e. full landscaping, room modelling), and so our smaller scale use of the app barely made a dent in our storage or bandwidth caps. After that, you can then download the point cloud files to a .PLY file on your computer, which you can then import to your PCL programs. There may be more features and systems available in the future, but the app is very simplistic currently and performs pretty much all that we needed for this project.

Appendix C: Cmake

Cmake configuration of PCL is extremely critical for this project. First of all, it is critical to stick with PCL 1.8.0, and we strongly recommend using PCL 1.8.0 all-in-one installer for Windows OS because it includes all the dependencies. In addition, Visual Studio 2013+ is recommended.

Make sure the project names and filenames are correct. More importantly, check PCLConfig.cmake in the cmake directory under your PCL 1.8.0 for cmake path variables and make sure cmake can find the proper libraries before compile. For further details please consult the README file in our drive.