Group Name Hackulus Thriftus

https://github.com/paigehinkle/hackulus

Group Members

Paige Hinkle (ph7457) paigehinkle@utexas.edu

Siena McFetridge (scm2539) scmcfetridge@utexas.edu

Jon Lee (cjl2443) chencjlee@gmail.com

Jaime Rivera (rjr2426) jaimerivera@utexas.edu

Rohan Ramchand (rsr898) rohan@ramchand.me

Project Topic

Reconstructing a 3D object or scene using Kinect and viewing that mesh in Google Cardboard

Objectives and Key Results

Objective: Create a 3D mesh by moving an object and not moving the Kinect

Key Results:

1. Create a mesh of the object using depth data from Kinect
2. Smooth the data using a filter so that the mesh is reduced in size so that we can render it on a phone for Google Cardboard

Objective: Create a 3D mesh of a scene using Kinect when the Kinect is moving in a circle

Key Results:

1. Stitch the images received from the moving Kinect to render a single scene
2. Create a 3D mesh of the scene using the depth data from Kinect

Objective: View a mesh in Google Cardboard

Key Results:

1. Mesh is loaded and viewable in a web application that works with Google cardboard
2. Mesh is able to be rotated or viewed at different angles

Meeting Schedule and Objectives

Progress Report 1: Have OKR 1 done and begin work on OKR 2

> Expected Effort: OKR 1 will be the most difficult to achieve. Thus, it may take until Progress Report 2 to be able to have something to show.
>
> **Updated 11/13/14:** Finished OKR 3.

Progress Report 2: Have OKR 2 done and be almost done with OKR 3

> Expected Effort: OKR 2 will be an extension of OKR 1, so assuming good progress with it, this will not be as difficult as OKR 1.

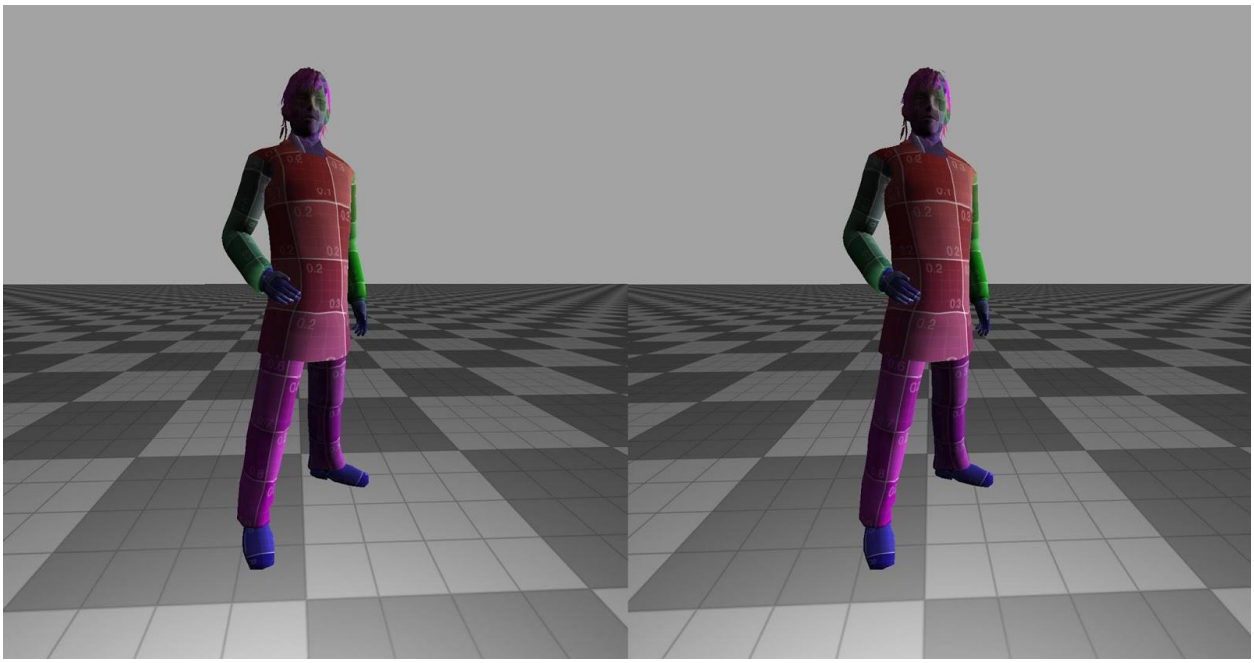
Final Project Due Date: Finishing touches on OKR 3 and possible reach goals

> Expected Effort: Depending on the complexity of the files given from OKR 1-2, OKR 3 will be somewhat difficult to render in a VR environment.

Progress Report 1

We chose to use a web app for Cardboard because of the simplicity in setup, since it uses Threejs and WebGL. We were able to complete most of OKR 3, which encompasses getting an .obj file loaded into the web app and projected into 3D space. Using the Cardboard setup, this object is projected automatically into a VR space with two views that are slightly skewed.

We are currently supporting .obj files since as a group, we decided that would be a simple file type to support. However, if we found that another file type worked better, we would be able to support that.



Next, we want to complete either being able to walk around the object or rotating the mesh along an axis so that you can see it from different angles.

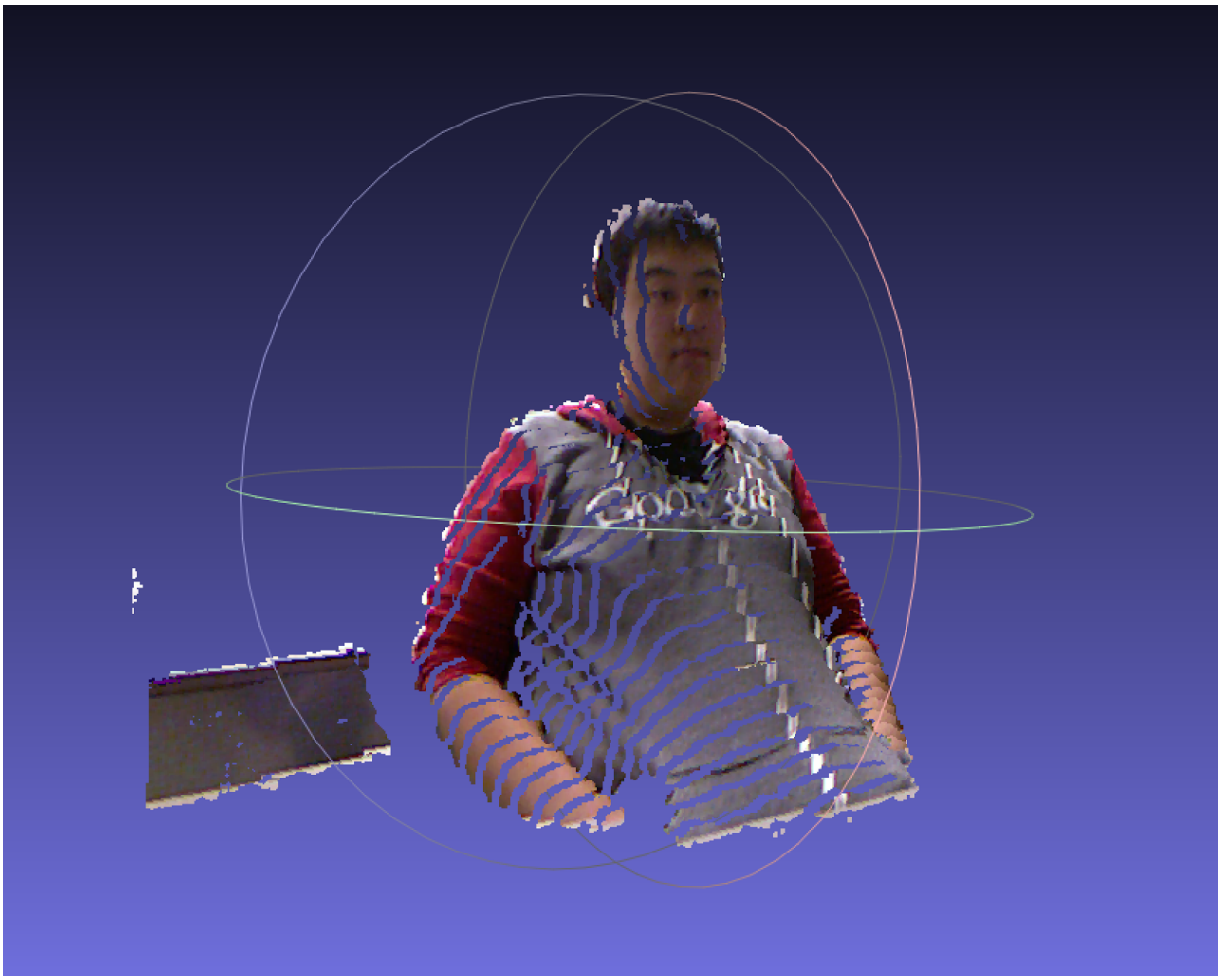
Unfortunately, we were unable to make much progress on the Kinect section of our project. The most significant problem we faced is that we were planning on using one of our team members' Kinect, but due to the device being from the newer Xbox Ones, we were unable to get a dongle to connect to our computers for development. As a result, we are more than likely going to request to use Dr. Klingner's Kinect setup instead.

As we noted in our status report above, we were going to finish OKR1 by the first check-in; however, we didn't anticipate having this kind of significant setback once we finished setting up our development environment. Since we had the development environment and setup for OKR3, it was easier to get started, which is why we decided to ahead with the Cardboard portion, since it wasn't an extension of anything else. On the other hand, we were unable to get started with either OKR1 for the reason mentioned above or OKR2 since it's an extension of OKR1. We plan on finishing OKR1 and hopefully being able to integrate our work with the results of OKR3 by progress 2.

Progress Report 2

We are now able to retrieve the depth map information from the Xbox 360 Kinect and in turn have generated several point clouds representing the underlying information. Using the OpenNI, NITE, OpenKinect, and SensorKinect libraries, we were able to connect the Kinect up to one of our Macbooks through OpenCV (compiled with OpenNI support) and

retrieve disparity maps that we were able to convert to point clouds like the following:



The next step is to stitch together several of these point clouds to reconstruct a 3D object. We have brainstormed several ways in which we could stitch the depth map information and below we discuss in more detail the ideas that we have come up with:

- We shall use an object that can swivel, such as a Lazy Susan, and place an object upon it. We will use this to rotate the object 360 degrees to retrieve depth map information from all angles.
- In order to make the point clouds useful, we have to determine the center of rotation for the swivel. We hope to do this by placing a vertical identifier at the center of the swivel and determining the depth of that identifier from the kinect. Since the kinect won't move, we can use this center of rotation as our origin
- Considering that we now have a center of rotation, we must then find a rotation matrix along the xz-plane that can transform all of the points in the point cloud onto the correct positions of another point cloud.

- Considering we're solving for a primitive rotation matrix, all we need to know if the angle of rotation in terms of one of the rotation around the y-axis.
- Before we move forward, we need to first identify feature points between two images. We will always use two grayscale 2D images to detect feature points and determine the all pairs of points that should correspond to one another. We will then take these points and convert them into 3 dimensions using their depth as the extra z-coordinate.
- Given the pairs of 3D coordinates that must be mapped to one another, we essentially have the equation: P * M = Q, where P is a 3-vector, M is a 3x3 rotation matrix, and Q is a 3-vector. We have P and Q, but we must determine what M is. We know it must be in the form of a rotation along the xz-plane, so the only missing parameter is theta.
- In order to determine what theta is, we shall take advantage of a wonderful characteristic of our swivelling object. Consider a two points A and B on a circle. Say that A is static and B moves. Say we draw a diameter across the circle starting at A to create point C on the circle. If we move B left of A until it hits C, the distance to B increases. The same applies if we move B to the right of A until it hits C. If we were to graph these distances starting from the leftmost point on the circle until C was reached to the rightmost point on the circle until C was reached, then we will have a concave up parabola. This is wonderful, because we can ternary search a function/graph that has a single inflection point. So from this, we will simply have a cost function that takes in a particular theta. It will rotate all of the feature points in image A by the rotation matrix M using theta. It then takes all of the points represented by A{i} * M and determine the sum squared differences in terms of distance from the expected points. The closer this value is to 0, the more accurate our transformation matrix becomes. Since we're using ternary search, this will converge extremely quickly and produce a nice stitching between the point clouds.
- From this, we are thinking of ideas of how to take the point clouds and turn them into a mesh-like representation. The idea of generating a bounding convex hull was nice, but it doesn't work for objects that have concavity. Because of this, we've considered a couple of other approaches. One of them is to use a O(N^2 * k) algorithm to determine the k closest points to all particular points and use those to create triangles for an obj file. Another goal is to use some sort of spatial partitioning, in the form of a KD tree or Octree. This will increase the runtime of our algorithm significantly, but we don't know how accurate or difficult this approach will be.
- Finally, given that we have a mesh, the mesh may not be smooth. To accommodate this, we are considering using catmull subdivisions to increase the size of our obj file, but to interpolate and increase the smoothness of our mesh.

The one worry is that either the .ply or .obj file will be too large to accomplish these tasks.

Viewing in Google Cardboard

- The object being viewed in cardboard should be able to be seen from different angles. Since cardboard does not have very many inputs to use we decided to use the pan and tilt controls to rotate the object instead of the camera.
- We are still working on fixing a bug where the camera gets stuck viewing the ground on mobile.