LEAP VIEW DESIGN

ANKIT GUPTA, LUCY NAM, AND KATHLEEN ZHOU

The Leap View image viewing tool require several steps, each of which involved a multitude of design decisions. As an overview to the software, the tool allows people to manipulate images using intuitive, simple gestures. See the full documentation for thorough details. These gestures required interfacing with the Leap Motion API for Javascript. So, we first spent around a week learning the multiplicity of functions that are available through the Leap Motion software. Then, we had to find a framework for panning and zooming, since those are the two major functions of our image viewer.

In order to do that, we searched the web and found a library called the PanZoom library. This was a very helpful library as it gave us access to variety of functions that made our implementation easier. For example. the library included functions like pan left, pan right, zoom in, zoom out, and fit. So, what we had to do was the more arduous task - taking data from the Leap Motion sensor and call the right functions depending on what the input was.

The first gestures we built were zooming and panning left and right. To do these, we harnessed the Leap Motion's ability to identify swiping gestures. However, we had to then interpret the gestures to understand the direction in which they were occurring. Namely, we had to be able to identify left from right, and up from down. This was relatively easy. However, there was a catch. When we made a, say, left to right gesture, there was a slight vertical change as well (no one's hand is perfectly horizontal in such a gesture). Thus, the program detected that there was both vertical and horizontal gestures, and therefore both panned and zoomed, instead of just one. So, we had to develop a way of checking which has greater magnitude of difference, and then use that to determine which gestures to perform.

Then, we had to develop a reset function. We felt that the most appropriate gesture for this was the circle gesture. However, we ran into some problems in implementing this. The issue was that midway through the circling gesture, the Leap Motion detects that a circle is being drawn, and then tells the program that there is a circle for the next 30 frames or so. As a result, the "fit" function was being called around 30 times. So, we had to develop a system that stored timestamps in an array, and only called the "fi" function if the fit function had not been called in the previous one second. This solved the problem we were facing.

Next, we developed a way to go to next and previous images. In order to do this, we had to make an array that stored the links to images. Then, we had a javascript loop that iterates over the array and adds the syntax for another image to the html. However, we used

another set of code from "tympanus.net/Tutorials/FullPageImageGallery/", that puts the images in a horizontal, scrollable format, among other things. So, upon adding the html for the additional images, we had to include the code to make the horizontal, scrollable format. Then, to switch images, I simply used a Javascript function that detects for a KeyTap or ScreenTap gesture, and then upon doing so, changes the picture to the next one in the array or the previous one in the array respectively. We also included an upload feature. On the top-left of the page, there is a text field in which the user can insert the url to any image hosted on the web (jpg works best) and click upload to get it added to the image gallery. Type in "http://wp.streetwise.co/wp-content/uploads//2013/08/1.-Harvard.jpg" for an example! We did this by simply adding the inserted link to our array of links, rerunning the function that iterates over the array, and then rerunning the function that makes the images horizontally aligned in a slider.

Lastly, we included a pullout menu that allows the user to see the controls. The requires a three-finger-down gesture to open and close the sidebar. Upon doing so, the user can click one of the options to be redirected to the user manual to see how to properly execute that gesture. This allows the user quick access to the user manual.

This project was complex because we were interfacing with technologically-advanced hardware, and yet we enjoyed working with it very much. Many of the design decisions we made took hours to think of and many more to implement. As a result of this, we benefited greatly as we got great experience working with APIs, especially in JavaScript.