Team Name: SHAPE
Team Members: Patrick Bowling, Andrew Lewis, Nick Strazis, Aaron Kwan, Jonah Simon
Instructor: Behrooz Shirazi
Mentor: Hassan Ghasemzadeh
Sponsor: Mona Ghandi
Course Number: CptS 421
Date Submitted: 4/28/17

Alpha Prototype Description

Android app:
Our sponsor, Mona, wanted a way to interact with the wall through touch. To accomplish this, we created an Android App using Java and Android Studio. The app allows users to touch a square the represents the wall and place a window in the wall. The window can be moved based on where the user taps in the square. To adjust the size of the window, users can moves sliders that represent the width and the height of the wall. Finally, users can submit the wall to the preview and to the final version of the wall. The application works by sending HTTP requests to the server. The requests contain the new coordinates for the window. This is currently the only peripheral that does not currently support or communicate with the server's preview. This is because the ap has its own local 'preview' and was designed before we created the server preview.

Alexa skill:
To provide voice interaction for the wall, we utilized Amazon's Alexa voice service to interpret voice commands. Our Alexa skill works by sending speech converted to text to an application we created on AWS Lambda using Node.js. The Node.js app works by first retrieving the current coordinates of the window via HTTP get request. Then based on the speech text it received, the Node.js app modifies the coordinates and sends them back to the server. Additionally users can create a new window instead of modifying and can submit the preview window to grasshopper.

With Alexa, we are able to create a window in the wall and move it around by using certain "utterances", or spoken phrases, which are mapped to "intents" (actions executed by Alexa). Some example intents include "move", "moveALittle", "makeAWindow", and "submit". The utterance mapped to "makeAWindow" is, appropriately, "Make a window." "Move it [up/down/left/right]" is mapped to the "move" intent which takes the direction given by the user and moves the window in that direction. Other utterances include "Make it [bigger/smaller]" and "Make it [wider/thinner]".

Kinect:
To allow the user to input data points via gestures, the team decided that using the Kinect would be the most efficient way to track human movement. The Kinect was the cheapest device that could track entire bodies and there was a vast amount of documentation for its code. In order to recognize the gestures created by the user, we created a program that draw a skeleton and attach it to the user. The program then logs 25 joints across the entire skeleton and uses these joints to log the gestures. As for the gestures, we have the program recognize three different hand shapes which are the closed fist, an open palm, and the lasso (which is two fingers pointing up). We combined these hand shapes with basic movements (swipe up, swipe down, swipe left, swipe right) to create gesture commands. Moving the right hand with a closed fist moves the box, swiping left with the right hand as an open palm makes the box thinner, swiping right with the left hand as an open palm makes the box wider, swiping up with the right hand as

a lasso makes the box taller and swiping down with the right hand as a lasso makes the box smaller. Once the user is satisfied with the changes and wants to submit, they just have to wait five seconds and after five seconds of inactivity, the program will send the data to the server.

Server design:
Our sponsor, Mona wanted multiple forms of input to be able to modify the wall. To accomplish this our design centers around having input from all peripherals sent to the server. The server updates to separate sets of data. The first is the ‘master’ set of data, which is what Grasshopper uses to generate the virtual wall. The second set of data is the ‘preview’ set. The preview set is where changes are stored before the ‘master’ dataset is updated. This is because we don’t want the physical wall (once it exists) to reposition itself after every tiny change, but to instead allow the user to make sure the wall is positioned how they want and then update the wall so as to reduce wear and tear on the walls physically moving parts.

Additionally, the server’s ‘preview’ dataset is used to display a live preview of how the wall should look. The server hosts a url which uses javascript and html5 to display a visual representation of the preview data and a very simplified version of what grasshopper would display. This can be accessed by most web browsers and so users can see the effect that their changes would have on the wall before submitting to the ‘master’ set and ultimately having those changes reflected in grasshopper.

Grasshopper backend:
The Rhino software had a small problem where the base software would not accept data sent from the server that holds the sensors data. To rectify this, the team used the extensions Grasshopper and Ghowl to allow Rhino to accept data points via code. To allow Rhino to accept information from the server, the team needed to send the data in a single file and parse it in grasshopper. To accomplish this, the backend uses Ghowl to ping the server every 5 seconds and request for the data in the XML format. Once the server sends the data, we check verify that the information was properly sent. Once we verify that the information was sent properly, we parse the data and plug them into the proper data points.

Design Modifications Resulting from Alpha Prototype Testing

Android app:
We have many future plans for the Android app. We found that it was tedious to view the preview canvas on a different screen. To make our app more user friendly, we would like to incorporate the preview canvas into our app. Doing this should make viewing the preview while using the app less tedious.

Another improvement we would like to add is support for gestures. We found it was very unintuitive to place the window and resize it quickly. In order to make our app more intuitive we plan to incorporate swipe gestures to place and resize the window. This should make it easier to place and resize the window.

Lastly, we eventually want to have the app support the server's preview system, so that when changes are made by other peripherals, the apps display will update, and similarly, any changes made in the app will be reflected in the servers preview, and thus changes will be reflected in any other peripherals which have a local preview.

iOS app:
Currently there is no existing iOS app. In the future, we plan to create an iOS app which functions the same as the android app. This will extend usability of our system to users of apple products, and is something which Mona has been clear she wants to eventually be implemented. Additionally we have talked about adding Siri Support to the iOS app. We will explore adding this feature in the future.

Alexa skill:
Right now, we have a limited number of utterances available for the user. Without prior knowledge of these commands, the user is unable to manipulate the wall. Ideally, the user will be able to move the window around in an intuitive way according to whatever word they feel is appropriate at the time. We can accommodate these needs by adding additional utterances for each intent. For the intent "move", you can only say "Move it [up/down/left/right]" right now, but we could make this feel more natural by adding support to more phrases that do the same as "Move". Some possibilities are ("Slide it…"/ "Bring it…") or by adding words in front ("Let's move it…"/"Why don't we move it…"). These utterances would make more sense in situations where multiple users are collaborating and both want to manipulate the wall.

Kinect:
At this time the Kinect is fully functional and the backend is well integrated with our server, thus allowing all inputs to be directly sent to GrassHopper. We plan to improve the user experience by improving gestures, the user interface, and the way in which the Kinect interacts with the actual wall.

To improve the gestures, we will be working closely with Mona and Hamid and create gestures that follow their specifications. Currently, the gestures that we have developed are at times challenging to complete, and have high risk of user error. What we plan to do to resolve these issues is to create gestures for resizing the wall that allow the user to grab the top and bottom sides of the wall and drag them further open or drag them together, effectively opening or closing the wall. To complement this, we can create a similar gesture for grabbing the left and right side of the wall and making it wider or skinnier. This is a more intuitive approach that we think will allow users to use the full functionality of the Kinect with relative ease.

At this stage in the Kinect development process, we have created a simple User Interface that shows the Kinect's camera image, as well as a rough outline for how the user is transforming the wall, and a rough position of the wall. We plan to change this UI so that it offers the user a more descriptive view of the wall and the changes that they are making to both the window's position on the wall, as well as the width and height of the window. We plan to create more of a visualization of what is happening that can be simplified for the user's convenience.

Lastly, the change the user makes to the current Kinect's representation of the window is only roughly translated to changes to the window on the actual wall. This is due to the information received from the Kinect being very different than the sizes and dimensions of the actual wall. For the most part, this will require fine tuning of the dimensions, and the translation to the actual wall. This is an issue that we are confident we will be able to fully resolve.

Server design:
Currently the server does not send a message to grasshopper or the preview when the window is changed. Instead we use timers so updates happen at set intervals. This can cause a significant delay with changes to the window reflecting on our models (up to 5 seconds). Due to server limitations, we cannot send messages when values change. A fix for this in the future is to acquire a server that allows us to run Node.js on it and open up ports. Our current server only supports PHP and does not allow Node.js to run on it. Node.js is always running unlike PHP. Currently our server scripts are running only when called. Creating a server based on Node.js would allow our scripts to always be running. This would allow for more real time updates to the canvas, grasshopper, and ultimately the wall.

Additionally, we would like to add a front end to the server. At the moment when users go to the server, there is no html front end and instead just a list of files. While the user can still access the preview by using the correct url, they have to know the url beforehand instead of being able to navigate to it.

The preview page is not currently compatible with all browsers (Internet Explorer) so we would like to add support for other browsers which are not currently supported. This would expand the usability of our system to be accessible in more circumstances.

Lastly, the server we are currently using is a shared server. This means that when our server receives a large amount of requests then the communications are throttled, and in some cases never recieved at all. This problem could be mitigated through the updated messaging system described above, however that is more of a bandaid fix, and in certain cases may even cause an increase in communications if the update messages are all sent at once.

Grasshopper backend:
The only plans toward grasshopper backend improvements are the implementation of a messaging system, to override our current timer system. As stated above this is a limitation of the server we are currently using so this improvement is dependant upon changes to the server. It is important to note, that in the coming semester Mona and Hamid may have us move away from grasshopper altogether in favor of a new platform such as autocad. In this case we would be building a brand new backend system for whatever rendering platform our sponsor decides upon.

Summary of Alpha Prototype Session with Mentor

Server design:
Overall Mona and Hamid were happy with our server. The lag/delay caused by the lack of messaging did not cause any issues in our demo. However, we discussed getting our own server instead of a shared server with Mona and Hamid. We agreed that next semester getting our own server would be very beneficial to the project. Multiple server options were discussed and in the end we decided on asking the EECS department for a simple apache linux server.

Android app:
Our Android app demo went very smoothly. The app was correctly updating the window in real time. The only concerns were accuracy of the finger tap and the lack of preview canvas. Mona and Hamid suggested that we add gesture support to make placing the window more intuitive. As discussed above, we have plans to alleviate all of these concerns next semester by adding the preview canvas to the app and adding gesture support.

Alexa skill:
The demo of our Alexa skill prototype went relatively smoothly. When we gave commands to Alexa, the software responded swiftly. Alexa was also able to recognize commands given by multiple users. We encountered problems when people had conversations after commands were given because Alexa was still waiting for an utterance and would interrupt for clarification. A stable Internet connection is also required for Alexa to respond in a prompt manner, but we have been operating on the assumption that the end user will have such connection. Mona and Hamid's feedback on the skill was overall very positive. There were two issues that Mona would like addressed. One issue is that the app does not allow the user to move the window diagonally. We plan to address this by adding support for diagonal movements. The next issue was that the app did not move the window the correct amount. This will be addressed when the wall dimensions are finalized over the summer.

Kinect:
Overall the demo for the Kinect Application was a success. Our team presented a video demonstration to Mona and Hamid and received mostly positive feedback from both. Approximately one week later, our team presented a live demo to Hamid, showing off the Kinect's ability to reshape the window, move the window, and have these changes take effect in both the server preview as well as in GrassHopper. Hamid gave us two thumbs up for the functionality of the Kinect application, but told us that in the future he wanted to see better gestures. As mentioned in the previous section, one of our main targets for the future plans of the Kinect is to work with Mona and Hamid to develop more intuitive gestures for users to use. Hamid agreed with our team that from a functional standpoint our progress on the Kinect application met and exceeded the aim of this semester's project.

Grasshopper backend:
There was not much to show Mona and Hamid for the grasshopper backend. It worked as intended in all of our demos. The only feedback we got on it was that it is not updating in real time and is updating on a timer. In the server section, we discussed how having a server with ports open would alleviate this issue.

Preview Canvas:
The preview canvas was created at the request of Mona and Hamid so that we would be able to see the changes in the wall before they are confirmed. When we demoed the submission of new data to our server, the canvas accurately updated according to the requested changes. Following that, we successfully submitted the changes to Grasshopper. We were able to do this while switching between platforms i.e. creating the window with Alexa then setting a new width/height using the Android app.