CS 6456 - Project Implementation

Using Gesture Recognition to Manipulate Graph-Based Visualizations for Data Analysis

Siddharth Raja
Georgia Institute of Technology
Atlanta, GA.
siddharth.raja@gatech.edu

Scott Wise
Georgia Institute of Technology
Atlanta, GA.
swise8@gatech.edu

Overview and Background:

In the project proposal, we described a system that enables users to arrange snippets of information into a Graph-Based Visualization by hand-gesture based manipulation of these snippets on a large display. We set out to build this system using a Microsoft Kinect v2, a projector (to project the screen on a large surface), and wrote the code in C# and XAML, making use of the WPF graphical subsystem for rendering on Windows 8. The Kinect V2 enabled us to detect hand-based gestures and translate them to manipulation of on-screen objects.

To begin, we recognized that the Kinect V2 could only be used on the Windows 8 operating system. There were thus three environment options for the implementation of our program: Unity3D, Visual Studio 2013 for Desktop, and Visual Studio 2013 for Windows. We chose to implement our application in Visual Studio for Desktop, which surprisingly has sharp differences with the "for Windows" Visual Studio in terms of Kinect support and application look and feel. Also, as far as data representation is concerned, we found that capturing snippets from the web browser pages would not be practical for our "for desktop" application. Hence, we assume that the snippets are already present in the workspace that we've built as a part of this application. We decided to focus most on the gestures, design, and user interaction experience.

Key Features:

- The features we mentioned in the project proposal included pointing/cursor movement, selection, un-selection, multi-selection, drag-n-drop, right clicking, scrolling, stretching/shrinking of objects, cut/copy/paste, edge creation, grouping, and zoom.
- Apart from right clicking, we implemented all the features in the list above. Instead of creating a gesture analogous to a right click we implemented a menu with key functionalities, which we think is a better option. This visibility of this menu is controlled by a vertical sweeping hand-gesture.
- We replaced the cut/copy/paste feature with a 'Duplicate node' feature which simply creates a copy of the selected node and adds it to the canvas. This seemed a much smoother way to create copies of nodes quickly.
- In addition to what we proposed, we created a gallery of snippets from where these individual snippets can be quickly put onto the canvas and also dragged back to the gallery from the canvas. This gallery is a scrollable vertical list that appears along the right edge of the screen. Although the main canvas in not scrollable, we allow the user to zoom in / out thereby adjusting the available space when needed.
- We've implemented a functionality to allow the users to take a screenshot of the canvas using a hand-gesture.
- We've also implemented a 'trash can' which allows the user to delete any nodes.
- Grouping is achieved by connecting the nodes with colored links, with each color being unique to a group. The 'Group Mode' button in the horizontal top menu bar enables the user to define such groups of nodes.
- We've also created a 'Shift Select' option that enables user to select all the nodes in a given group and move all the nodes in that group simultaneously across the canvas. This shift select mode is enabled by selecting any node in a group and then closing the left hand's fist. Opening the left fist would get rid of this group selection.

In the section below, we have compiled a list of gestures we incorporated in the system to enable users to leverage these functionalities.

Implementation Process:

Node Interface Design:

The first step in creating our Kinect whiteboard was to find a way to best represent nodes. We decided to represent them as thumbs (from the C# Thumb class). These thumbs are simply manipulatable objects that we made to be represented by a single image file. The next step was deciding how the connections between each of the nodes would be represented. We decided that grouped nodes would be best represented by lines. Thus, each thumb had either a set of start lines and end lines emanating from their locations. Once we figured out how connections would be represented, we next had to set up how these lines would be created by the user. Our first implementation was to have a button labeled “create links”. When we click that button, our cursor would change its shape and we would then ctrl-click on two nodes, which created a link between them. This was clunky, but we chose that style to begin the project.

Kinect Support:

At that point we had to figure out how to add support for the Kinect V2. Here are some things we had to think about: how does a user select nodes? How does a user move nodes? How does a user create links between nodes without a mouse?

In order to be able to manipulate objects on the canvas, we made our canvas a Kinect Region. Once this region was defined, we had to track the body of each user in the canvas. Once the body data is tracked, both of the user’s hands are then be tracked. However, the Kinect V2 SDK assigns a primary hand to the user based on whichever hand it tracks first. This is the hand that shows up on the screen. It enables a manipulatable object to be created on the screen, and as a result, allows users to manipulate the canvas itself, i.e. change the size of the canvas. This was problematic for our desire to implement a two-handed manipulation system. Therefore, one proposed alternative to using the Kinect SDK primary hand feature was to generate our own images for each hand from the skeleton data feed and map the absolute positions of the hands from the Kinect camera to their relative positions on the actual canvas. However, this did not seem practical and would become more of problem than implementing the actual interface. We decided to stick with what was offered in the Kinect SDK. Either way, the Kinect Primary hand system does not hinder the use of two-handed gestures (only two-handed manipulation of objects).

New Link Creation:

Once able to firmly manipulate the canvas, we had to manipulate the thumbs themselves. In order to select thumbs, a user simply closes his fist over the location of the thumb. This allows the user to grab objects and drag them on the screen. When a user closes his/her fist over an object, the thumb is considered selected and has a bounding box around it. We will call the most recently selected thumb the primary thumb. In order to drag the primary thumb in the Kinect Region, we had to continually update its position, as well as maintain the data structure that holds the lines emanating from it and its connecting thumbs. Once a primary thumb is selected and draggable, another thumb may be selected after it. This newly selected thumb becomes the new primary thumb. We then call the thumb selected before the newly selected thumb the secondary thumb.

This system allows users to understand how to draw and delete links between nodes. Essentially, when a primary and secondary thumb is selected the user can draw a line between them by closing both his/her hands together, almost as if the user is clapping. In order to delete the link between a primary and secondary node, a user simply makes a slice gesture around his/her neck. The details of how these gestures were created are in the Gesture Creation section below.

The next problem we wanted to tackle was the importing of nodes. One major problem with importing new thumbs was the fact that the Kinect V2 SDK cannot communicate with the underlying Windows System (at least not easily). In other words, if we opened a dialog box in the application, we would have to use our mouse to navigate through folders to select an image we

wanted to import. In order to avoid that problem, we allow users to maintain a folder of all the thumbs they want in a given project. They can then open up a right menu (that stays fixed to the right to decrease time of access – Fitts' Law) we will call a "Snippet Gallery" with all their desired thumbs listed as buttons. They can click (by click, we mean push their primary hand forward) on a thumb button in this Snippet Gallery and it is added to the main canvas. If a user wants to put that node back into the image bank, the user can simply drag and release that thumb over the Gallery and it is added back as a button.

As mentioned above, we also set up a trashcan in the left corner of the canvas for users to drag any elements they wanted to permanently delete. The trashcan provides visual feedback for deletion in the form of puff of smoke.

Gestures Created:

Throughout development process, we had to come up with different gestures that would work best with the Kinect, as well as with our application. These gestures were created in the Microsoft's Visual Gesture Builder and were created from a range of 3 to 9 training clips, each about a minute long (1 GB of IR, Depth, and body frame data). In essence, the Visual Gesture Builder allows users to import videos of proposed gestures that can then be marked on a timeline. The system then creates a database of these gestures using a machine learning approach to gesture building, as opposed to a heuristic/algorithmic-based approach, which significantly saved time and lines of code and allowed us to implement more features in the long run of the project. Here is a list of the gestures we were able to implement:

- *Swipe right hand toward body* - opens the right Snippet Gallery menu.
- *Swipe right hand away from body* - closes the right Snippet Gallery menu.
- *Swipe down* - opens up the top menu.
- *Swipe up* - closes the top menu.
- *Clap Hands Together* – Draw link between primary and secondary thumbs
- *Slice Neck* – Delete links between primary and secondary thumbs
- *Hold hands up like a picture pose* (similar to minority report) – This allows for a snapshot of the current state to be taken in order to get a picture of the user's current canvas. This can also be taken through a snapshot button. This gesture also provides visual feedback in the form of a yellow flash to tell the user that snapshot has been taken.

The Visual Gesture Builder is by far the sleekest of the new additions to the Kinect SDK, although it does come with its own drawbacks. One major problem with training gestures with the visual gesture builder is that gestures that are similar must be trained rigorously, which leads to several hours of training. It thus becomes just as important to make sure that users train the gestures the ML algorithm should NOT recognize. For example, our link combining gesture brings both hands together, like a clap. However, the snapshot gesture brings both hands relatively close together, as well. Because the skeleton tracking is still not incredibly robust, the body's hands are considered blobs, and when the blobs are near each other, they are still read as closed together.

Finger Tracking:

In our original proposal, we wanted to explore the ideas of finger tracking with the Kinect V2. A very new library made by Metrillus was released in Beta in August and is called Aiolos. Aiolos makes use of the IR data, depth data, and skeleton data. We decided we wanted to make use of this finger-tracking library. After contacting the creators and understanding we could not use the .dll in Unity, we moved on to implement finger gesture detection in C# (this is why we ini-

tially chose to work in Visual Studio for Desktop). Unfortunately, this came with a few setbacks. Finger tracking in Aiolos stops when one finger is down/null. We wrote an algorithm to detect a "rock on" symbol (to detect distance between the index finger and intermediate fingers between the pinky). Unfortunately, the way detection was implemented would not allow us to successfully detect a pose, and because the additional use of both IR and depth data significantly slowed down our application anyway, we decided to stick with full body gestures.

Downsides/Improvements to be made:

As a result of making use of full-body gestures, we were able experience gorilla-arm and arm fatigue first-hand, one of the major complaints of such a system. However, depending on a user's level of engagement, this does not seem to be much of an issue, although it may become fatiguing after a minute or two.

Conclusion:

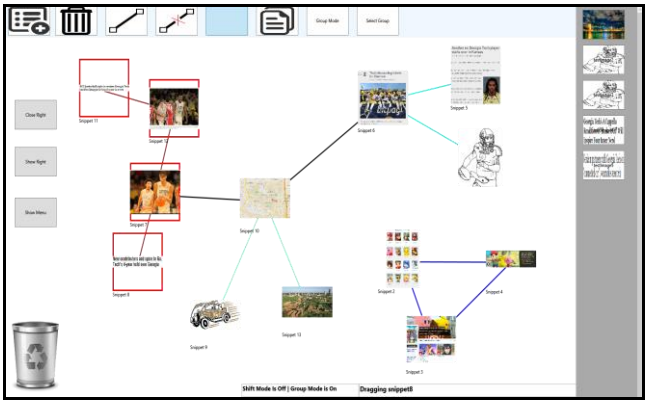
We believe we were able to meet and even exceed many of our implementation goals. The compromises we made were actually made better than originally proposed – i.e. the Snippet gallery, menu system, and group mode.

We found that this was a great opportunity to learn about a brand new technology that spans across the realm of computer vision and expands upon its own previous implementation—the Kinect V1. We felt we were able to leverage the Kinect's ability to recognize full body gestures. We are also now much more aware of the positives and negatives in Microsoft's documentation, the WPF work environment, as well as node-based code structures.

Base Code and References:

The base for building a Kinect Region was found in Microsoft's official Kinect SDK WPF examples. The base for adding the gesture databases from the Visual Gesture Builder was also found in Microsoft's official Discrete Gesture Basics example. This code was modified significantly to support our custom-made gestures. We essentially split Microsoft's code-base into GestureDetector and GestureHandler classes. This separation allowed us to seamlessly integrate new gestures to be detected and then allowed us to handle the gestures once they were detected.

In addition, an example called "shape connectors" written by Dennis Vuyka was a good starting point for understanding how to use Xaml in manipulating objects on a Xaml canvas. Once we better understood our problem and had a firm grasp on how we would implement our own system, we were able to make a clear deviation from his original examples. This example gave us a basic Xaml canvas base to work from.



This figure shows the layout of our Kinect region. The image shows grouping of snippets via colored links, one entire group of nodes selected (shown by red rectangle border on each snippet), the horizontal menu bar at the top, the vertical snippet gallery on the right corner and the trash can at the bottom.

We have a basic landing page for this project online. It will include a video and detailed screenshots for the project: siddharthraja.com/uiproject/