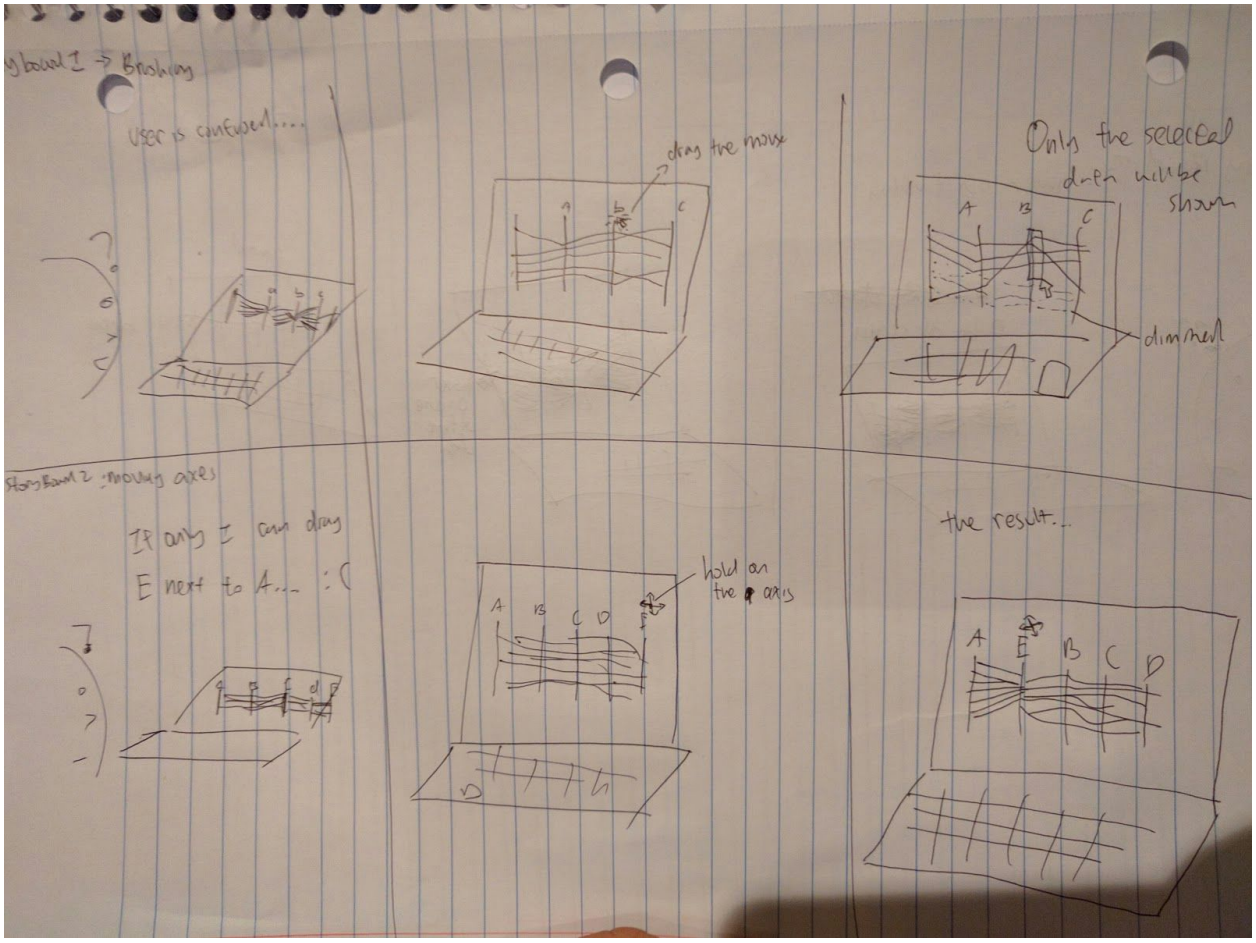
Vincent Jonany
INFO 474
Assignment 3 Write Up
02/16/16

Introduction

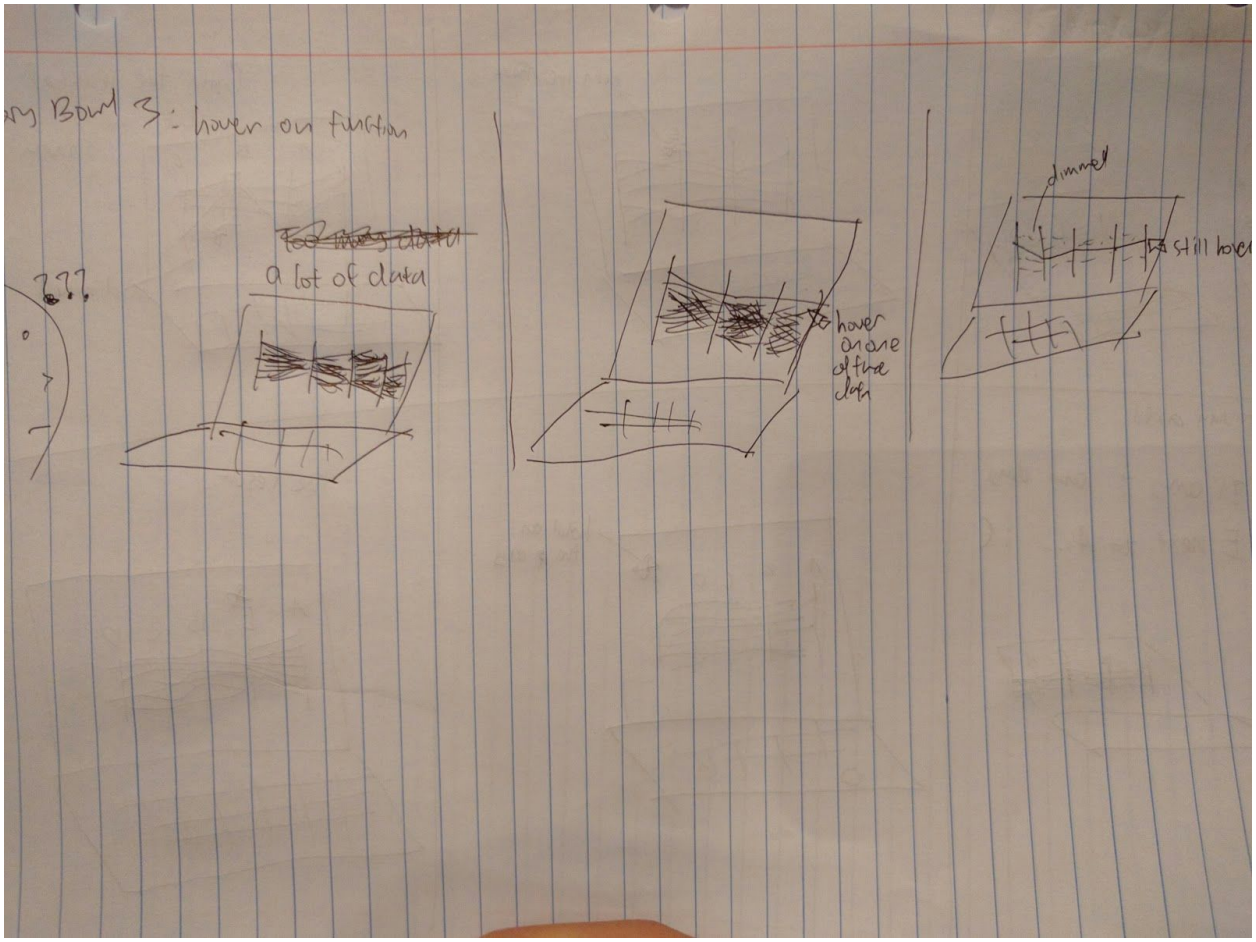
I first did research on the different interactive visualization techniques that first, is implementable in the scope of the assignment. Secondly, it has to work great in parallel coordinate visualization. I have always wanted to implement interactivity in a parallel coordinate visualization, because there are so many hidden messages and results that can be only be conveyed through interacting with the visualization. Traditionally, a static parallel coordinate visualization issues a huge amount of data, with many correlations through the parallel coordinates. Though one can grasp few results from it, one may have missed more crucial points from the visualization due to the absence of interactivity. Based on my research on the different helpful visualization techniques for parallel coordinates visualization, I will try to implement brushing, dragging of the axis, and a basic hover functions.

The data domain that I chose revolves around the different models of cameras with their specifications, and price. Hence, I will develop an interactive parallel coordinate visualization so that people can see the different properties of the different models of cameras, and more importantly, users should be able to see the correlations between one specification/attributes to another. The method brushing is very helpful in filtering out the specific range inside one specification, so that one can focus only on the correlations that are created just from the specified range in that attribute. A specific scenario in which the user will be using this feature is when a user wants to compare the specifications of all the cameras out there that are in the price range of $100 to $150. Hence, the user will brush the data that are on the price axis that are in the range of the user's preference. Then these data will trace and will be highlighted across all the coordinate, whereas all the other unselected data will be dimmed (Storyboard 1). Dragging the axis is very important and useful for this domain as well. Different users want to see different kinds of correlations. For example, user A might want to see the direct correlation between price and the zoom wide value. On the other hand, user B might want to see the direct correlation between price and storage. These axes may not be close to each other in the visualization, but the user will have the ability to place the axes they want next to each other (Storyboard 2). Lastly, since my dataset involves more than a thousand rows of different camera models, it will be good for the users to be able to select one row of data and trace the data across the coordinates very easily. A hover function which will highlight a particular data and dim the rest of the data will definitely help the users. A use case for this includes when a user wants to see the data for Sony 625 model. However, the rest of the data is distracting the user to focus on this particular data only. Hence, when the user hovers this data only, the rest of the data will be dimmed, and hopefully, it will help the user to focus on this particular data only (Storyboard 3).

Storyboard 1 & 2



Storyboard 3



Process:

I first just layout everything without interaction. What is hard is that for the axes or domains, not all of them are the same type. For example, the Model axis is an ordinal, rather than a linear scale. And I had to do a little bit of research on why .ordinal() does not really work with .range() than rangePoints(). The reason being is that .range() does not work with ordinal in terms of automatically insert the “in between” values from min and max. Whereas rangePoints(), read the model name from the Map object returned from the d3.map() function of the single row from the csv file.

With the help of online tutorial, I was able to implement axes drags. It took me some time to understand every lines of code in the function, but I managed to convey my understanding of it through the comments.

Same goes with brushing. It was not as hard as I thought it would be, as the d3 has brush function which makes it really easy and straightforward. It was very difficult for me to be able to brush over ordinal values. I did not implement it due to the scope of the assignment. But the idea is that I had to map the pixels drawn by the extent() method, and find what the ordinal value or the model name of it in that extent of pixels. And when mapped, return to the extents object the min and the max ordinal value of from the rectangle brush that was created by the users. This extent object will be used to check on every data that pass through the axis. If it is “smaller” or “bigger” than the values in the extent object, then it will make it a ghost line.

After brushing, I added the stroking implementation so that user can focus on one data only. I was scared that it will conflict with the brushing implementation. Fortunately, it did not conflict, and went quite well. For easier stroking, I added another superGhost layer with wider stroke-width. I also must not forget to include this layer when users move axes around. I wanted the users to be able to stroke the model’s name on the side of the axes. However, this will allow the users to be able to stroke all of the ticks on the axes, and it is not a good thing, because some ticks do not have any data on it. This is one of the other problems that I could not solve. I believe that this function outweighs the problem, and hence, I will keep it that way.

The last problem I had with the visualization is that the data from Cameras.csv is too abundant. It makes the visualization lags really badly. Hence, I scoped the Cameras to Epson and Canon EOS cameras only. The scoped data can be found in Cameras1.csv, and what is used to feed the visualization.

Instructions:

1) Go to the directory
2) run a simple python server
3) go to the browser open localhost on whichever port the server is running on
4) To apply the brush, go to one of the axes, and drag an area on the axes. One can also move the created brush around the axes. One can also create more brushes on other axes to filter more data.
5) To move the axes around, hold one of the title of the axes, and move it around.
6) To stroke one of the data, one can simply hover on one of the lines.

Conclusion:

All in all, the visualization conveys the storyboard pretty well, except for the problems with the strokes. If only I disable the strokes on the titles and the ticks, but not the title for the camera's model names, then it will be just the way I want it to be.

Roughly, it took me about 10 hours to develop this application. The aspect that took the most time include when I tried to implement the dragging of the axes. There are so many things that are needed to be included. Unlike the built in brush function from d3, I had to implement it by myself. I also found it very hard, because the hints I got from other people, due to my lack of knowledge and experience in d3. On the other hand, I gained so much knowledge from understanding other people's code line by line, and apply it to my application.