Visualizing Eye Tracking Data on the Web

Kevin Krpicak

**Intro/Purpose**:

For my capstone, I wanted to do some project using a few web technologies that I wasn’t familiar with yet. These technologies include Node.js and Canvas. I always wanted to do a project using these services but never had the motivation or opportunity. So after talking to a few professors on capstone ideas, I managed to ask Bonita if I could do a capstone dealing with client-side web technologies. She came back to me with the idea of using a new Hosting platform, Windows Azure, to host a projects data she had been working on in the cloud. The current project dealt with tracking a users eye movement while they were looking at the computer screen. That is how the project idea started. We needed a place to host the eye tacking web application, a place to store the eye tracking data, and a client side representation of the data and video.

**Background on Eye Tracking**:

Eye Tracking is simply tracking a user’s eye movements on a screen. For this project we used the Mirametrix S2 eye tracker, which is a small piece of hardware that sits in front of the screen when a user is undergoing an eye tracking session.

Some definitions relevant to eye tracking are fixations, saccades, and scan paths. Fixations are points on the screen that a user is looking at. These are the focus of my project, such that I need to sync of these fixations from the XML data with the AVI video file that the eye tracker produces. Saccades are very small, almost unnoticeable eye movements between fixations. These saccades are not represented in my capstone project. Scan paths are paths produced between fixations.

**Approach**:

In order to actually begin the eye tracker project, an approach or plan of attack was necessary. The project was obviously going to be broken up into two main components, the server side part and the client side part.

**Server Side**:

For the server side, which was used to handle storing the data and fetching up the data, I decided to use Node.js as my server side language. I am very comfortable with client side JavaScript, I figured it would be a good time to use Node.js and become familiar with it. Besides just using Node.js, I also wanted to use the Express.js. Express.js is a minimal framework for Node.js, providing some very useful web functionality. Some of the Node.js libraries I planned to use were XML2JS, Async, and MySQL. XML2JS is a Node library used to convert XML data over to a more useful web data structure, a Javascript Object. I used this to convert the XML data filled file into a JSON file stored into a public folder. The other Node library I used was called Async. It provides a simple way to handle JavaScript’s asynchronous methods. I used this during the XML and AVI upload process. The last Node library used was MySQL. This library was used as a driver to access a MySQL database and do the necessary queries.

To go along with Node.js as a server side language, I needed a place to store data from the Eye Tracker. I decided to use a MySQL database mostly because that is was I’m most familiar with.

After deciding what server side language to use, I needed a place to host the web application. The plan was to use Windows Azure as a hosting solution. We thought it would be interesting to play with Microsoft’s new talked about Cloud Platform.

**Client Side**:

Besides the server side aspect of the project, a client side representation was needed. Because of my web background, I was more familiar with this part of the project. I knew I was going to be using jQuery, Backbone.js, and Underscore.js as the main structure for the front end. I was also going to be using Mustache templates to render out dom, and Twitter Bootstrap as the actual styling of the web application.

Some of the other font end services that were needed for this project were HTML5 Video, Canvas, and Heatmap.js. HTML5 Video is a newer way of displaying videos in web pages. The goal was to use newer web technologies, and being that HTML5 is the newest way for rending videos on a website, that was the route we were going to take. Canvas is another newer web technology that was going to be used to render out the fixations from the Eye Tracking data on top of the video. And lastly, Heatmap.js was going to be used as a different visual representation for the fixations from the Eye Tracker.

**Problems Encountered**:

With every project there tends to be problems when trying to complete the goal. Most of my problems came from the choosing Windows Azure as my hosting provider. When first choosing the site, it seemed to have everything I needed to build the back end and store all the data, but I soon found out it didn’t easily support MySQL databases. Windows Azure only supported SQL databases, which are very similar, but wouldn’t work with how I wanted to make the application. After researching for a possible solution to get MySQL databases on Windows Azure, I found out that I needed to run a Linux based Virtual Machine. This wasn’t something I was very familiar with so it did take some time installing MySQL, installing node, installing express, building the database, and SSHing into the VM. After the initial setup things seemed to be going smooth, but then one month I got an email saying I exceeded the monthly download rate, and my account was disabled. I wasn’t too worried at this point, because I expected my account to be enabled again when the new billing period began. When my account was enabled again as I expected, I discovered that my VM was removed. I called support and they actually said they get a lot of calls about that event happening. With no other options, I decided to rebuild the VM. I recorded a step-by-step list of what I did to set up the VM, but after creating my new VM, I was unable to actually SSH into the VM to get anything done. I kept getting timeouts, and after looking around it was a common problem other people were having. As a result, I decided to just host the project locally for the time being mostly because of time.

Besides just the hosting problem that I faced, some other issues came from the way JavaScript generally runs. JavaScript is an asynchronous language, so when uploading the XML file and the AVI file, I needed a way to know when the uploaded files were completed. Using the Node library, async, easily solved the asynchronous problem. Basically you set functions that are suppose to happen as promises, where they wait to execute the final callback till all the promises have finished.

Another problem I ran into was actually syncing the fixation data with the Eye Tracker video file. After I discovered the event that was continuously being fired called “timeupdate”, I knew I needed to grab the fixations according to the time closest to the current time of the video. One of the problems with the “timeupdate” event that the HTML5 video was sending me was the difference in frequency of the event per browser. For example Firefox triggers the update event every frame, whereas chrome triggers the update every so many milliseconds.

**Lessons Learned**:

The main lesson I learned from this project was that I needed to pick the correct hosting to go with the correct project. It was a bad idea to pick a Microsoft based hosting provider to use with more Linux based services. It just became more of a hassle then it was worth, and I was not utilizing the services provided. Everything turns into a work around when you go down that route.

**Future Work**:

There’s still a few pieces of this project that need to be completed in order to have a finished product that can work well with the Eye Tracker. For one, the AVI file that is uploaded to the server needs to be converted on the fly to a Webm, MP4, or OGG file type, preferably all three for the best compatibility across browsers. This can be completed by using the ffmpeg Node library. I just didn’t get around to implementing that part of the project. Also expanding on the ffmpeg library, a frame or thumbnail can be grabbed from the uploaded video, which can be used to render a thumbnail on the front-end. It will make the user sure of which video is being selected.

Other than the remaining functionality, the project needs to be hosted somewhere. I haven’t really thought much about a new place to host it, but Rackspace might be a good option. After hosting the application, that will provide the ability for who ever is working on the Eye Tracker API to hit a URL on this project to upload the XML file and the Video file. I do have a URL set up already for the files to be sent to. Currently if you hit “/add/” and post the two files it will put the files in the correct location. This can’t work though till its hosted some where.