Apex Prototype

Documentation

by

Tommy Fang

7/06/15 - 8/21/15

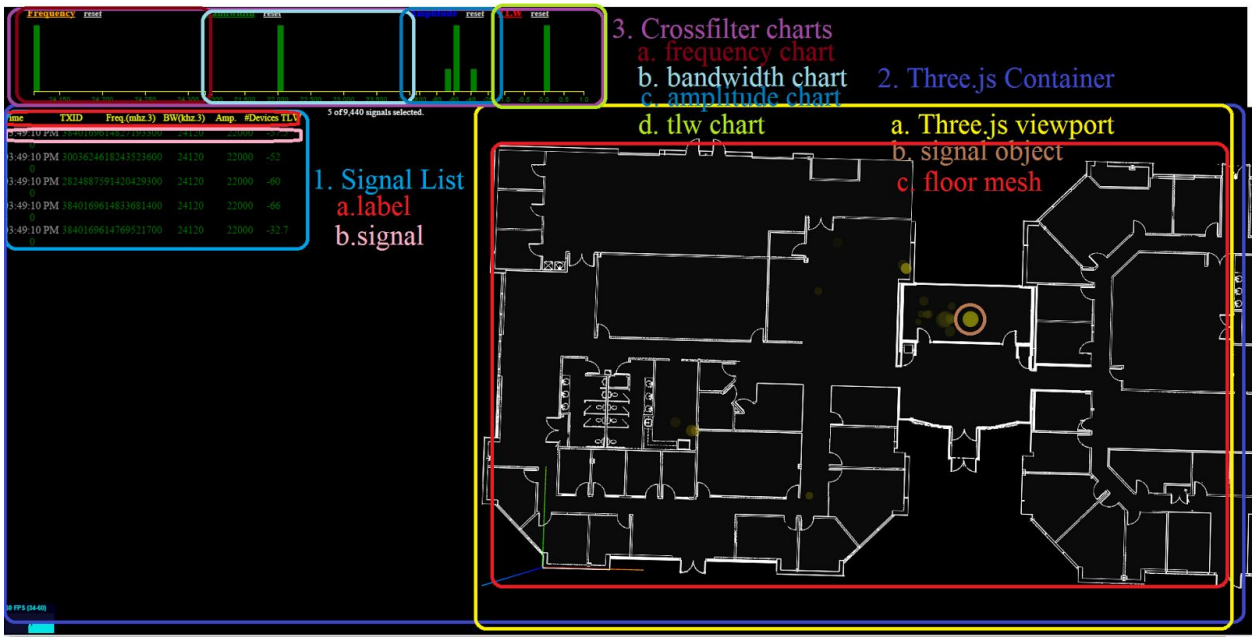
Transformational Security

Engineering Internship

dkfang7@gmail.com
Mobile: 646-257-0278
https://github.com/tfang7

Table of Contents

1. Introduction



I began this internship with some knowledge of web development that I gained by having a hobby of developing applications with various technologies. I was unfamiliar with all of the frameworks that were used in this prototype, however by the end of this internship, I felt that I had mastered many of the frameworks I had just been introduced to about 2 months or so ago. The purpose of this document is to provide the information necessary to pick up where I had left of and to prevent the next programmer from making the same mistakes I made.

The prototype utilizes HTML5/CSS and Javascript to provide a sleek interface that processes large amounts of data.

First, a server is set up using Node.js/Express.js, this allows us to send data from the server to client in order to decrease the workload of the client. In this prototype, two CSV files are sent(a large T2 Signal data file and a small floor plan CSV file). Node has a simple module

loading system, files and modules are in one-to-one correspondence. See **Section 2: Getting Started**, on how to install and use node modules. The files are sent using an http put request from the server and received using an http get on the client side.

These files are utilized through the frameworks optimized by data analysis experts (Crossfilter.js and D3.js). The file gets parsed by D3.js and is converted into a CSV file object. This is an array of CSV objects and each index represents a line of the file and the first line of the file are used as the indices of these objects. The floor CSV file is loaded and rendered by three.js to the THREE.scene which holds all of the visible objects. We can view the scene using a perspective camera and move around using orbit controls. see OrbitControls.js for more details. The camera movement is restricted to 180 degrees. Left click pans movement. Right click controls orbit movement. Mouse wheel zooms in and out.

Crossfilter takes this array and processes it so it can be usable in the data charts and Three.js visualizations. Once Crossfilter returns the filtered array that needs to be visualized, this array is sent to main.js, where the data pump resides. Currently, the data pump is triggered by the "L" key and renders the respective signals as circles in the current filtered array. The circles are drawn using the signal data. TLW, Frequency, and Bandwidth can be toggled to define the color. The size of the circle is defined by the Amplitude. These values are scaled using D3.js and an appropriate value is output depending on the range of the min and max values of these variables. The filtered array is organized by timecode and the "L" key will cause pump to start visualizing the first time code until it reaches the last time code. It will start over once it reaches the end. Code needs to be added to allow play, pause, and reset functionalities. For now, the user has to

reset the page to select a new selection.

---

2. Getting Started

You can either use a personal IDE that is compatible with Node.js or you can run the server through the command line. Ignore **2.a, if you don't want to use Intellij Idea IDE and go to 2.b.**

2.a. Install IntelliJ Idea 14.14 Ultimate Edition

i. Either buy it, use the free 30 day trial, or obtain the free student version.

ii. Clone the repository and drag project folder to desired location.

iii. Installation Guide

2.b. Install Cygwin Terminal

2.c. Node/Node Modules

2.c.i Download and Install Node.js

2.c.ii. Node modules such as d3.js, crossfilter.js can be imported using Cygwin.

2.c.iii. These modules allow us to use these frameworks on the server side and can be found within the project folder "Spiral/node_modules".

2.c.iv. To install modules:

```
1. cd into path/to/project_directory using Cygwin.
2. type npm install [module_name], it will be installed into node_modules
```

2.d. Final Step

2.d.i. If you downloaded your own IDE or used Intellij, just run bin/www, which will run a server at localhost:portnumber, which is 1337 by default.

2.d.ii If you chose to run the server through the command line, run Cygwin and cd to the project directory. cd into the bin folder and type

```
node www
```

This will start the server, wait a few moments, and go to http://localhost:1337. You can also safely delete the ".idea" project folder if you do not plan on using Intellij IDE.

---

3. Technologies and Frameworks

3.a. Languages

3.a.i. CSS/HTML5

index.html is the root of this prototype and it is heavily modified by charts.css. Much work still needs to be done to improve the overall design of the prototype, but the foundation of where everything should be has been laid out.

3.a.ii. Javascript

There are many javascript files used to create this prototype. I placed the frameworks I used in "public/vendors/" and all of the code I wrote has been placed in "public/resources/". Node.js is used to set up the server.

Folder Structure:

Public/

- Index.html

Public/css

- Charts.css, Three.css

Public/Vendors

- Crossfilter.js, D3.js, Detector.js, jQuery.js, Tween.js, Three.js

- OrbitControls.js, RequestAnimationFrame.js, Stats.js,

Public/Resources

- Animations.js, Charts.js, Data.js, Controller.js, Main.js, Scene.js

I will explain how I used the most important files in **Section 3: Source Code.**

3.b. Frameworks

3.b.i. Three.js, Tween.js

Three.js is a powerful visualization Javascript 3D library created by Ricardo Cabello. It utilizes WebGL to render complex shapes. You can easily draw various geometric and custom shapes without any prior knowledge of WebGL through this simplified API. There are many examples readily available. The github has many utilities that were developed for open source usage, please utilize this repository if you run into trouble.

Tween.js is a tweening engine used to easily create tweens for three.js There are two APIs with the name "tween.js", the one that I used can be found here.

3.b.ii. D3.js, Crossfilter.js

3.b.ii D3.js

D3.js allows us to select and bind data to DOM elements with very simple commands. Very elegant and efficient algorithms were developed by Mike Bostock to rapidly process thousands and thousands lines of data, which was necessary for this project.

3.b.ii Crossfilter.js

Crossfilter.js depends on D3.js and uses the data binding commands to combine data to create a filter that is able to take in multiple parameters. The current filter is able to filter signal

data based on TLW, Amplitude, Bandwidth, Timecode, and Frequency. The charts render the data live as well. It is extremely efficient and easily processes the 11000 lines found in the test signal data. This API has so much potential and I was amazed at how powerful it was in conjunction with D3.js.

3.b.ii. Node.js, Express.js, jQuery.js

3.b.iii Node.js

Node.js is used to set up the backbone of this prototype. I had the most difficulties trying to learn about the server-client relationship. I'll try my best to explain my thought process behind setting up the server. This code was generated using the Intellij Idea 14.14 template. It provides routes, which are used when a page is loaded. It takes in a user request on load. This request can be a variety of actions such as file uploads/downloads and sending data back and forth. Views can be defined by the programmer, they are an alternate form of page displays. In this prototype, I do not use views, I just serve a static html index, but it can easily be modified to use views, so if multiple pages need to be set up, we can easily do so. In order to properly use a router and a view, the link must be set up within "Spiral/views/Template/header.ejs". The extension ".ejs" refers to "embedded javascript", it is a form of HTML, in which we can embed javascript within the tags without <script>.

Example of embedded Javascript:

```
<p>Welcome to <%= title %></p>
```

```
In this example, title is a string variable, it gets printed with the <p> tags
```

The key files are "Spiral/routes/index.js", where the csv file gets loaded in and "Spiral/bin/www", where the server gets created and setup on localhost:portnumber where portnumber is currently equal to 1337. The current code includes other routes and views such as

users.js and about.js. These should be used as a template to show you how to correctly direct a page request to another page.

3.b.iii Express.js

Express.js is used as middleware for node.js in order to connect the components of this application. Usage of express can be found in "Spiral/app.js". In this file, express.js connects the application to the views, directories, cookie parser, bodyparser, and routes.

3.b.iii jQuery.js

jQuery.js is a feature rich library with a multitude of uses. In this project, it is used for getting AJAX requests to the server and receiving the data as a response.

---

4. Source Code

4.a. main.js, data.js

4.a. main.js

public/resources/main.js contains the data pump, which tells the page to retrieve signal data from charts.js and visualize it using three.js. This file requires Crossfilter.js, D3.js, and animations.js.

On document ready:

```
$(document).ready(function () {
    // initialization
    init(); see scene.js
    // animation loop
    animate();});
```

Animate: requestAnimationFrame was created by Paul Irish to optimize the animation loop. It prevents the page from animation while we are in a different tab or when we're not on the page.

```
function animate() {
    requestAnimationFrame(animate);
    render();
    update();
```

```
Render  is called 60 times a frame. It is the most memory consuming
function in this object. Look to prevent unnecessary render calls in
order to best optimize this program.
```

Variables:

```
_currentTimeIndex = 0, This keeps track of the current time selection
_floors = [], _dataSet = [],These are used to hold the data we receive
_loading = true, Loading is used as a flag variable
_signalDictionary = {},
SignalDictionary holds all the objects are currently being visualized.
_filteredSelection = []; The current filtered selection updates as a global,
So we constantly know what the filtered selection is.
```

4.a. data.js

public/resources/data.js contains the function that uses an AJAX put request to update the data that is sent from the server. Data sends the signal file to charts.js, where it processes and renders all charts and scales(color, size).

```
function LoadCSV(dataset, callback) {
 var result = [];
  $.ajax({
     type: 'PUT', define the request type
      contentType: 'text/csv',
      url: 'http://localhost:1337/config',
url: string containing the url to which  the request is sent
    success: function (CSV_ARRAY) {
We can only manipulate the data within this asynchronous success function
        if (dataset != null) {
              result[0] = CSV_ARRAY[0];
              result[1] = CSV_ARRAY[1];
Store the result using a callback function
Result is sent to charts.js (crossfilter/d3.js) and scene.js(three.js) upon
load
          }
          callback(result);
      }
  });
}
```

A Timecode helper function was also written and is still within the code, but I found out that crossfilter handles the same functionality easily. I left it in data.js in case it needs to be used later on.

4.b. scene.js, controller.js, animations.js

4.b. scene.js

public/resources/scene.js handles all of the visualization that the crossfilter will feed it. This file requires animations.js, OrbitControls.js, Detector.js, data.js, controller.js, and three.js.

Variables: see code for more details on parameters.

```
container: refers to the div element that holds all of the rendering we will do
scene: the background /3D plane, it holds the three.js objects that we add.
camera: A perspective camera that allows us to freely view the scene.
renderer: A detector class is used to see if the user's browser has webGL
enabled, otherwise this becomes a THREE.CanvasRenderer.
controls: THREE.OrbitControls are provided by the Three.js team. It allows to
move around scene using the mouse.
stats: used for development purposes, shows framerate and lag time.
Animator: instance of AnimationHandler, see animations.js.
FloorData: holds the array of CSV objects representing each line of floor data.
CrossFilter: instance of chart class, used to filter signal data. see charts.js
RawSignalData; holds csv signal data that gets parsed by d3 server side.
```

Functions:

```
function LoadData() {
   //Grab the data from the ajax call started in data.js
   LoadCSV(dataset, function (result) {
       This asynchronous anonymous callback function stores the data into
results.
       FloorData = result[0];
       RawSignalData = result[1];
       if (Loading) {
           LoadFloors(FloorData, 1);
           Draw the floors using the floor CSV.
           second parameter is number of floors.
           CrossFilter = new FilterCharts(result[1]);
           Create an instance of the charts class,
           allowing us to access the filters/charts and render them.
       }
       Loading = false;
       We are finished loading data
```

```
Signal Data CSV Columns:
TCODE,TIMESTAMP,TXID,FREQ,BW,AMP,DEVCNT,X,Y,Z,TLW
Floor Data CSV Columns:
floor_id,name,image,building_name,scale,origin_x,origin_y,altitude,building_off
set_x,building_offset_y,building_offset_z,last_updated,last_updated_by,
image_width,image_height
```

I think the origins are unclear. For example, when looking at the map produced by the original Apex map in ASP.net, I thought the origin of 0,0 began in the top left. More careful

work should be done when handling origin in this prototype. I will explain how the Three.js coordinate system works. I think the size of the image should also be included because there is not a built-in method or an easy way used to obtain the dimensions of an image. It is absolutely necessary to have these dimensions when scaling the objects using the origin. The base origin will be determined by the floor that is considered to be the base floor of the building.

```
floorMesh.position.x = BaseOrigin_X + Origin_X;
floorMesh.position.z = BaseOrigin_Y + Origin_Y;
```

The new floor origins are added to the base origin, so that each floor will reference the same origin point, but their positions have moved. The altitude (z-axis in csv)/(y-axis in three.js) should be adjusted in the CSV based on how they actually end up looking in the three.js scene. Currently I set them 1000 px apart in altitude, and it seems that they are too close together, so adjust this value accordingly.

Three.js takes in the image width and height as x and y values. So it will place the floor mesh vertically in 3D space without the programmer modifying the rotation of the floor mesh. The code below rotates the floor 180 degrees on the x axis, making it horizontal against the camera. and then adjusts the z axis so that the x axis lines up properly.

```
floorMesh.rotation.x = -Math.PI / 2;
floorMesh.rotation.z = Math.PI / 2;
```

Unfortunately, this causes the z axis to be inverted in the normal sense of a cartesian coordinate system. Where the bottom left quadrant should be considered, (-x,-z), it is now (-x, +z). The top right quadrant is considered (+x,-z) and the bottom right quadrant is (-x,+z).

```
_originAxis.position.x = -halfX + Origin_X;
_originAxis.position.y = FloorMesh.position.y;
_originAxis.position.z = -halfY + (ImageHeight - Origin_Y);
```

When calculating the values of x with respect to the origin, which is located in the very bottom left corner, we take half of the negative value of the width and add it to the new origin. Y is

altitude, so this is equal to the value located in the csv value. Z can be calculated by taking half of the negative height and adding it to the ImageHeight + Origin_Y, which will invert the Z axis.

WIDTH: 1985, HEIGHT: 995

In this figure, you can see that the Z value actually refers to the 2 dimensional up/down values. X is left and right. Y refers to altitude in three.js 3D space, where positive Y goes upwards above the plane, and negative Y goes downwards below the plane. The scene sets up the plane in a 3D cartesian coordinate system. The quadrants above show how the points in space are laid out. Setting the position of the plane at (0,0,0) center world causes the map to be split up as shown on the map. The points can be calculated by dividing the dimensions in half. The "z",referring to the height of the 2D map, axis goes from (-Height/2) to (Height/2). The "x' axis, which corresponds to width, goes from (-Width/2) to (Width/2). The scene takes the image which was initially serialized into the CSV as a 64 byte array using a base64 encoder and easily deserialized using:

```
function CreateFloor(dataset, FloorNumber, FloorDimensions) {
            image = dataset[FloorNumber].image
            FloorPlan = "" + image,
```

When the floor is created, I also add point light at the center using three.js.

```
var light = new THREE.PointLight(0xffff00, 1, 0);
light.position.set(BaseOrigin_X + Origin_X, FloorMesh.position.y + 250,
BaseOrigin_Y + Origin_Y);
scene.add(light1);
```

This render function sets up the viewport for the screen, because we need to allocate space on the top for the charts and space on the left for the signal table. The render function also constantly updates the new tween animations.

```
function render() {
   TWEEN.update();
   var left   = Math.floor( window.innerWidth  * 0.248 );
   var bottom = 0;
   var width  = Math.floor( window.innerWidth );
   var height = Math.floor( window.innerHeight );
   renderer.setViewport( left, bottom, width, height );
   renderer.setScissor( left, bottom, width, height );
   renderer.setPixelRatio( window.devicePixelRatio );
   renderer.enableScissorTest ( true );
   //renderer.setClearColor( new THREE.Color().setRGB( 0.5, 0.5, 0.7 ) );
   renderer.render(scene, camera);
}
```

4.b. controller.js

public/resources/controller.js contains the event handlers for the page.

```
function onDocumentMouseDown(event) {
   event.preventDefault();
   var offset = $('#ThreeJS').offset();
   var top = offset.top;
   var left =  Math.floor( window.innerWidth  * 0.248 );
   mouse.x = ((event.clientX - left) /
renderer.domElement.width ) * 2 - 1;
   mouse.y = -((event.clientY - top) / renderer.domElement.height)
* 2 + 1;
   FindIntersects();
```

This is the most important function in this file. It takes in the renderer size and offsets of the inner window to calculate the position in space using mouse coordinates. FindIntersects() constantly uses this event handler on mouse click to determine if an object has collided with the

mouse. Offset top refers to the spacing of the Three.js element. In this case, it is about 190px from the top when in fullscreen mode. Offset.left had to be calculated differently because I set the renderer's viewport from the left about `window.innerWidth * 0.248` pixels. If the renderer's viewport size changes, then this equation needs to be adjusted.

```
function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);}
```

This listens to whenever the size of the window changes and adjusts the renderer. It needs to adjust for the signal table data and chart data.

```
function OnKeyDown(event) {
    switch (event.keyCode) {
        case 76: //'l'
            event.preventDefault();
            if (!Loading) {
                //Slice is used to play a current selection
                var slice = _filteredSelection[0].values;
                var sliceBegin = slice[0].TCODE;
                var sliceEnd = slice[slice.length-1].TCODE;
                currentTimeIndex = sliceBegin;
                var start = window.setInterval(function () {
                    DataPump();
                    _currentTimeIndex++;
                    if (_currentTimeIndex > sliceEnd) {
                        _currentTimeIndex = sliceBegin;// 0;
                    }
                }, 3000);
            }
            break;
```

This function listens for any key presses. In this case, when the "L" key is pressed, we obtain the current filtered selection. Selected[0] is a selection of signal data, from the CSV, sorted by time code. The initial time code is obtained by using the first index of the selected array. The last timecode is obtained by using the very last index of this array.

The second parameter of the window.setInterval(function(), time(ms)) allows us to control how often this function gets called. Each time the function gets called, it increases the

timecode by one and renders the signals of the same timecode until it reaches the end of the slice. Then, it will start over again afterwards.

4.b. animations.js

```
function AnimationHandler()

Animator = new AnimationHandler();
```

This function is defined as a class that handles creating all of the tweens for the signal objects. An instance of AnimationHandler is created within the init function at the same time the scene is added.

Tween animations:

We can tell a signal object to tween by setting the current animation of the signal object and telling it to start with a tween.js start() method.

```
this.PopSizeIn = function (Signal) {
   //Size bounces as it pops into existence.
   return new TWEEN.Tween(Signal.scale)
        .to({
            x: 2, //absolute values
            y: 2,
            z: 2
        }, 500)//duration of animation in milliseconds.
        .easing(TWEEN.Easing.Bounce.In) //type of easing animation
        .onComplete(function () {
            Signal.userData.animations.anim =
Dwell(Signal).start();
        });
};
```

Pop: When a signal object is read in through the data pump, if it doesn't exist in the current signal dictionary, which holds all current visible signal objects, then the object is added to the dictionary and assigned a "Pop" animation. "this" refers to the instance of AnimationHandler(), which is called "Animator" currently. This function causes the object to do a bouncing size animation into existence. The scale which is originally set to 0 eases to a scale of 2. On completion of this function, it calls Dwell.

Dwell: Does nothing for 3000 milliseconds. It literally sits on the screen for 3 seconds. If new data doesn't get received for this signal object, Then it calls Fade.

FadeOut: On completion of the dwell stage, fadeout reduces the current opacity to 0 in 1500 milliseconds and deletes the signal from the current signal dictionary.

```
this.Move = function (Signal, pos_x, pos_y) {
   return new TWEEN.Tween(Signal.position)
        .to({
            //These are absolutepositions
            x: parseInt(pos_x), //Moves Signal's current x value by
"+pos_x"
            z: parseInt(pos_y) //Moves Signal's current y value by
"+pos_y"
            //duration: .5
        }, 1000)//Time it takes to finish this tween in millisecond
```

Move: If an existing signal object was found based on TxID in the signal dictionary, then the Animator calls move on this object. If the position has changed, then the object will tween to the new position based on location relative to the floor's origin. The position relative to origin has to be calculated before the move() call and the x,y parameters must be passed in. If implementing movement for altitude, add a pos_z parameter and parse the Signal.Y coordinate located in the signal data file.

TweenColor: Tweens from the rgb values of color to the rgb values of newColor in 200 ms. The input **color** is the color of the mesh.material.color and **newColor** should be created using **new THREE.Color(colorvalue)**, which converts HSL/HEX color values into RGB.

```
this.TweenColor = function(color,newColor) {
   return new TWEEN.Tween( color )
        .to({ color: newColor }
        , 200);
};
```

4.b. charts.js

This file contains all of the crossfilter and d3 handling. First, the server passes in the csv file parsed as a csv object filled with arrays representing each line of data. Each line is parsed, converting all number strings into integers and the timestamp is formatted.

```
signals.forEach(function (d, i) {
   d.index = i;
   d.date = parseDate(d.TIMESTAMP);
```

Afterwards, the record of each signal is grouped by all the dimensions we want to filter. Frequencies, bandwidth, amplitude, timecodes, and TLWs are grouped together in this project.

```
timecode = signal.dimension(function (d) {
   return d.tcode;}),
timecodes = timecode.group(function (d) {
   return Math.floor(d / 10) * 10;}),
```

This is a feature of Crossfilter that allows us to quickly identify the record of the signal that fits within the parameters of the various filters.

```
tcMax = d3.max(signals, function (d) {
   return d.tcode;
tcMin = d3.min(signals, function (d) {
   return d.tcode;
```

The maximum and minimum range of each variable is calculated using d3.min() and d3.max.

```
_sizeScale = d3.scale.linear().domain([ampMin, ampMax]).range([5, 10]);
```

These ranges are used to determine the domain of the bandwidth, frequency, and amplitude scales. They also determine the domain of the bar charts.

```
_tlwScale = d3.scale.linear().domain([tlwMin, tlwMax]);
//Dependent on domain, output the respective color.
_tlwScale.domain([0, 0.3, 0.6, 0.9]
//Create a threshold for each color
   .map( _tlwScale.invert))
```

```
    .range(["green", "orangered", "orange", "red"]);
```

TLW color is the default color scheme when the program starts currently. I bound the toggles of bandwidth, frequency, and tlw to their names on the corresponding bar charts.

```
_charts = [
   barChart()
        .dimension(frequency)
        .group(frequencies)
        .x(d3.scale.linear()
            //I attempted to fix these ranges by hard coding in some constants
            //The issue is that these current domains are not inclusive
            //At freqmax, the bar representing a signal with value of freqMax will
render off the _charts
            //at the end. It occurs with all of the domains in this chart array. I
believe it has something to
            //do with rangeRound or the setup of the grouping function. see above.
            .domain([freqMin, freqMax + 200])
            .rangeRound([0, 2 * 130])),
```

All of the charts can be found in this global variable _charts array. To create a chart, we set the dimension and the group that the chart uses. `barChart().x` refers to the axis. We set the domain of the x-axis to be displayed and set the range of values.

```
<div class="charts">
   <div id="frequency-chart" class="chart">
        <div class="title">
            <a href="javascript:toggleFrequency()"
class="toggleColor">Frequency</a></div>
   </div>
```

To display the chart on the webpage, create a new div within the charts div in **index.html.** Order matters. Crossfilter will look for the chart class as it loops through the _barCharts array. Frequency is the first item in the array, and frequency-chart is the first div in the index.html, so it will render the data that was used earlier in this example on that specific chart.

```
Example of a toggle
window.toggleFrequency = function () {
    _tlwToggle = false;
    _bwToggle = false;
    _freqToggle = true;
    var _SignalDictLength = _SignalDictionary.length;
    if (_freqToggle && SignalDictLength != 0) {
        var SignalFreq, color;
        for (var id in _signalDictionary) {
            if (_signalDictionary.hasOwnProperty(id)) {
```

```
                SignalFreq =
((_signalDictionary[id].userData.freq));
                color = new THREE.Color(_freqScale(SignalFreq));
                  __signalDictionary[id].material.color = color;
            }
        }
    }
};
```

---

5. Transitioning/Post Mortem

5.a. Obstacles

My biggest challenge during this internship was my lack of familiarity with the frameworks. I had only learned basic Javascript, HTML/CSS before coming to work with TS. The node server-client relationship was the biggest pain for me to learn due to the way concurrency works in Javascript. I also had no understanding of WebGL or how to utilize GLSL in order to make custom shaders that would have significantly enhanced the performance of this program. By the end, I gained confidence in all of the new frameworks and I feel that if I had enough time, I could eventually implement all these features properly, so that it can become a polished product.

5.b. Optimization

Rendering was definitely the biggest area that could be improved. Each time a render call was issued, the frame rate dropped significantly. (see **Section 5: Testing** for more details). There are several ways I believe this performance could be improved.

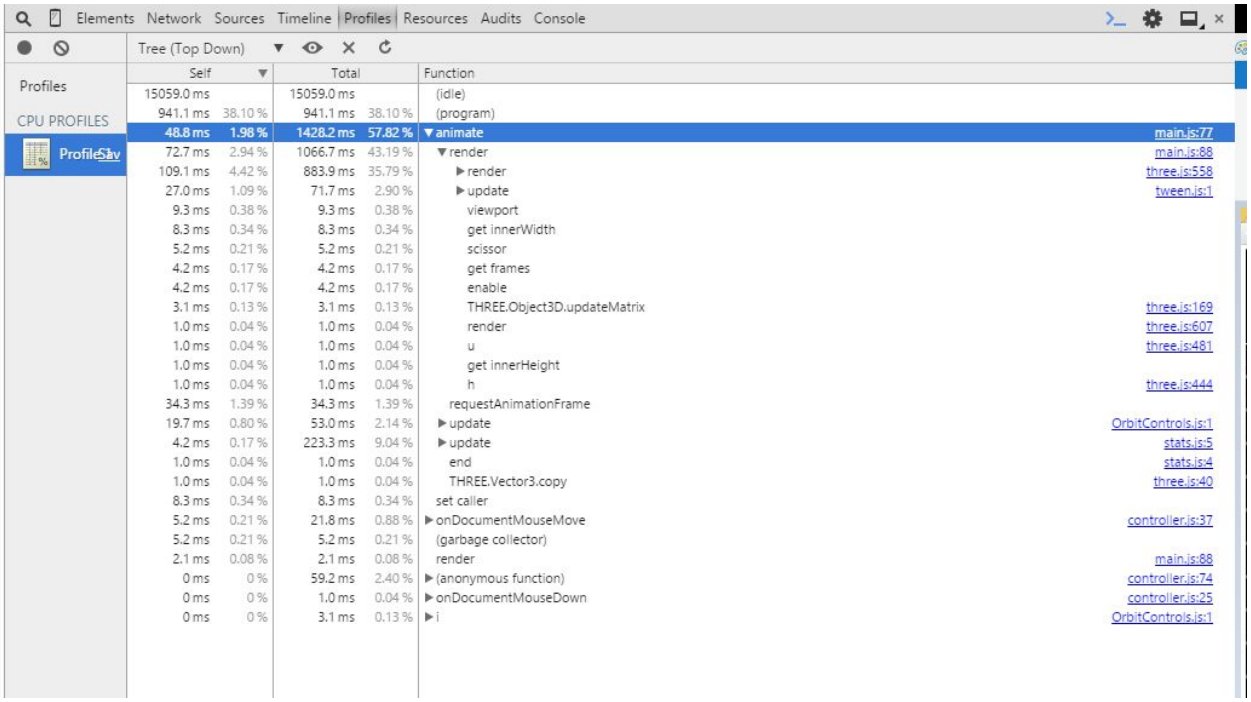
First, the biggest improvement that could made is to create custom shaders that render each circle. A web worker(see Section: Resources for more details on web workers) would have to made to get the attributes of the shader and modify variables such as positioning, color. This requires great understanding of the GLSL library.

A web worker could be implemented without shaders to also improve performance. A web worker performs a task asynchronously and it could calculate new positions for the signals, greatly decreasing the workload of the client in the long run.

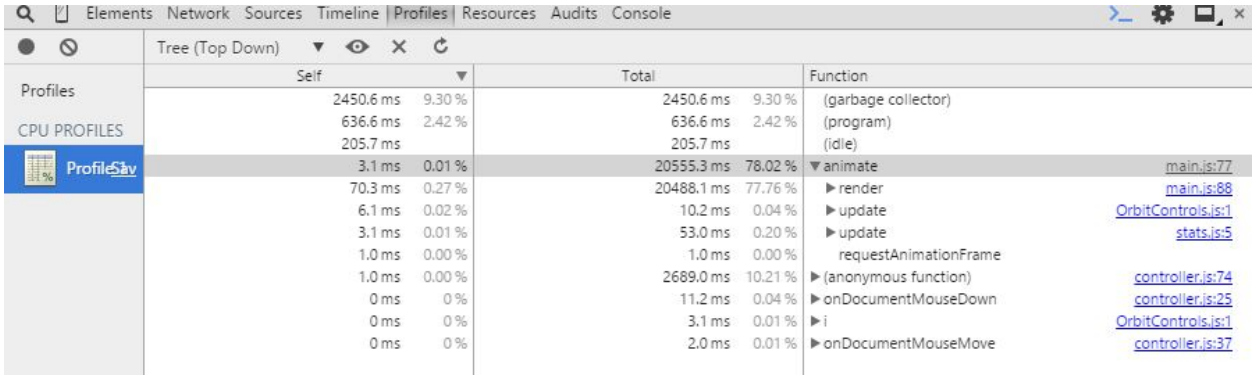
Currently when a signal is formed, it creates a brand new instance of a Cylinder geometry object and a brand new instance of material object. I attempted to test reducing the amount of "new" calls by using the same geometry object and material object to create **ALL** of the circles, however it was unable to give each circle unique characteristics and all geometry objects had the same size, all material objects had the same color. I'm not sure if it is possible, but this implementation did significantly improve the frame rate of the program.

5.c. Testing

I tested this application using Chrome's Developer Tools, specifically the memory profiler. The animate and render functions guzzles up 90% of the memory that gets used. This was tested on the T2Signal data file that had about 20 or so signals per time code. It easily renders these amount of data, and I believe that having 2000-3000 signals on screen isn't much of a problem, however it's definitely the animation and new render calls that causes stress on the system.



Test Data 1: Small signal data file with 11000 lines and 10-20 signals per timecode, 60 FPS



Test Data 2: 1000 signals rendered at a time. 10-30 FPS

The second test data file shows a massive difference between rendering time. Each individual call does not take up that much time, but it increases with the amount of data due to the increased amount of garbage collection and amount of objects that must be rendered and animated with each frame.

5.d. Next Steps

If I were to continue working on this project, I would ask for a clear explanation on how signals would be displayed in relation to the floor they are on. Currently, the floors define the scaling for the movement and positioning on signals. However, I know that later on, there will be signals that aren't inside a building and are at some variable altitude.

The current coordinate system has an incomplete malleable origin system. Problems were caused because of the way the signal data is formatted by ASP.net. I would create a function that successfully takes in values from the signal data and converts the x,y,z positions to three.js's coordinate system and scales it appropriately. I would probably begin attempting to implement shaders, only the circles would need to have shaders built for because later on, it would help for huge amounts of data.

The technology that I used is sufficient to demonstrate the features of the product that I was asked to make a prototype for. However, this prototype has too many performance issues to be used as an actual product. This prototype proves the ease of processing large amounts of data and the ability to render filtered data live based on user selections. All functions that were asked of me to make have been implemented, but there is room for so much more improvement.

I would also work on scaling the signal table elements and charts when the window is resized. The server code can also be vastly improved to lessen the workload of the application. The amplitude, tlw, frequency, and bandwidth scales can be processed server side and be sent from the server at the same time the initial csv data is sent.

If we request each an individual time slice from the server, I believe that it would also increase the performance of the application. Currently a global variable of the current filtered

selection is passed into the Data Pump and it processes each time slice based on the selection's first time code until the last time code. The code should still be compatible with this sort of implementation. The data pump would still be called every "x" amount of ms and be fed into the crossfilter. Crossfilter takes all the data that is given and renders that information. It would be not be a drastic change to make this program take live data.

Next, I would implement pause, play, and reset features for the data pump. The data pump has no way of knowing when to restart. Upon pressing "L", it just plays the current selection. Pause can be implemented by using binding an event using jQuery and the window.clearInterval function to stop the data pump. Play would start the interval again. This requires constant knowledge of indices of the first object's timecode and last object's timecode. On reset, it pauses the pump and sets the time codes to 0.

5.e. Resources

Chrome Developer Testing Tools

---

https://developer.chrome.com/devtools/docs/network

Three.js

---

http://threejs.org/
https://stemkoski.github.io/Three.js/
http://www.smartjava.org/content/all-109-examples-my-book-threejs-threejs-version-r63
https://github.com/mrdoob
Useful Three.js extensions: https://github.com/mrdoob/three.js/tree/master/examples/js

---

Performance Tips

Google Performance Talk: https://www.youtube.com/watch?v=rfQ8rKGTVlg#t=31m42s
Google examples: http://webglsamples.org/google-io/2011/index.html
Simplified WebGL: http://twgljs.org/
http://www.html5rocks.com/en/tutorials/webgl/million_letters/
http://www.ianww.com/blog/2012/11/04/optimizing-three-dot-js-performance-simulating-tens-of-thousands-of-independent-moving-objects/
http://www.paulirish.com/2011/requestanimationframe-for-smart-animating/
http://chimera.labs.oreilly.com/books/1234000000802
Web workers:
https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers
http://www.w3schools.com/html/html5_webworkers.asp

D3
http://chimera.labs.oreilly.com/books/1230000000345
SignalR
http://www.dotnetcurry.com/signalr/1040/using-d3js-signalr-realtime

---

Node.js

---

http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js
https://nodejs.org/
Node/Express with Intellij Idea IDE Tutorials: https://www.youtube.com/watch?v=-u-j7uqU7sI

Crossfilter

---

http://square.github.io/crossfilter/

Tween.js
https://github.com/tweenjs/tween.js/