Visualyzic: A live-input visualizer

by

Disha Garg
B.Tech., Guru Gobind Singh Indraprastha University, 2013

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

© Disha Garg, 2018
University of Victoria

All rights reserved. This project may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.
Visualyzic: A live-input visualizer

by

Disha Garg
B.Tech., Guru Gobind Singh Indraprastha University, 2013

Supervisory Committee

Dr. George Tzanetakis, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)
Supervisory Committee

Dr. George Tzanetakis, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

ABSTRACT

Audio visualization is the process of generation of images based on the audio data. This audio data is extracted from the music and this extraction of information from audio signals is known as content-based audio processing. It is a part of Music Information Retrieval (MIR). MIR is a young and active multidisciplinary research domain that addresses the development of methods for computation of semantics and similarity within music [1]. This project is a single-page application of an audio visualizer that takes in the live-input through your device’s microphone and visualizes it. The visualizations have been done by using the canvas tag, WebGL library and THREE.js library. The extraction of the audio features has been done by using Web Audio API. This web application can be used in studying the various aspects of Web Audio API in MIR. This application can also be used as a visible speech unit for the deaf.
# Contents

**Supervisory Committee**  ii

**Abstract**  iii

**Table of Contents**  iv

**List of Tables**  vi

**List of Figures**  vii

**Acknowledgements**  viii

**Dedication**  ix

## 1 Introduction  1

1.1 Structure of the Project Report  1

## 2 Problem and Related Work  3

2.1 Google Chrome Music Lab  3

2.2 Web Audio API  4

2.2.1 The Audio Context  4

2.2.2 The Frequency Domain  5

2.3 Spotify API  5

2.4 Polymer 2.0 framework  6

2.5 Three.js library  6

## 3 Design  7

3.1 Technologies/Libraries Used  7

3.2 Data Extraction  8

3.3 Application  9
List of Tables

Table 4.1 Table describing the Project structure . . . . . . . . . . . . . . . 15
List of Figures

Figure 3.1 Data Extraction by AnalyserNode ........................................ 9
Figure 3.2 Allow Mic Permission ......................................................... 10
Figure 3.3 Spectrogram with and without colour ................................... 11
Figure 3.4 Visualization 1 of Time domain .......................................... 11
Figure 3.5 Visualization 2 of Time domain .......................................... 12
Figure 3.6 3d Visualization ................................................................. 12
Figure 3.7 Beat tracking of Spotify tracks .......................................... 13

Figure 4.1 Taking audio input from the mic ......................................... 16
Figure 4.2 Creating an Analyser .......................................................... 17
Figure 4.3 Rendering Time Domain .................................................... 20
Figure 4.4 Rendering Frequency Domain .......................................... 21
Figure 4.5 Collecting features for 3d animation .................................... 22
Figure 4.6 Setting up the Scene ............................................................ 22
Figure 4.7 Animating the Scene ............................................................ 23
ACKNOWLEDGEMENTS

I would like to thank:

**my supervisor, Dr George Tzanetakis,** for his continuous support, guidance, patience and for giving me this wonderful opportunity to work under his supervision.

**my father, Bhushan Garg,** for his love, financial support and for always believing in my abilities.

**my mother, Suman Garg,** for her loving upbringing and nurturing, without which I would not have been where I am today and what I am today.

**my brother, Abhishek,** for his love and support throughout.

‘Who you are tomorrow begins with what you do today.’
- Tim Fargo
DEDICATION

I would like to dedicate this to my parents who have been a great source of inspiration.
Chapter 1

Introduction

The visualization of audio is the process of generation of images based on the audio data. The audio data is extracted from the music and rendered on the screen in real time for the users to view. The visualizations are in a way synchronized with the music as it is played. Effective music visualization aims to reach a high degree of visual correlation between a musical track’s spectral characteristics such as frequency, amplitude, beat etc. and the objects or components of the visual image being rendered or displayed [2]. This project allows a live audio input into the browser’s microphone and transforms it into three kinds of visualizations: 1. The time-domain plot, 2. Spectrogram, and 3. 3D Animation. The time-domain plot draws the time-domain graph of the audio on the canvas. The frequency-domain paints the canvas with the spectrogram of the audio. The 3D animation tracks the beat, volume and waveform of the audio and animates the objects on the canvas. Lastly, there is an oscillator, which produces a sine audio when mouse-down and mouse-move events are triggered. The controls for the visualizer are also built in order to analyze the visualizations in-depth or with some more clarity. This project has been developed as a web-application that follows Polymer 2.0 framework. It uses several apis and libraries like Web Audio API, Spotify API, Threejs Library and WebGL.

1.1 Structure of the Project Report

This section provides information on what each Chapter of this Report will discuss:

Chapter 1 gives an overview of the visualization project.
Chapter 2 talks about the problem being worked upon in this project and the related work and the overall motivation for this implementation.

Chapter 3 highlights in detail discuss the technical design of the visualization web application. The UI and various features of the application are shown.

Chapter 4 explains the implementation of the visualizer. Here, the logic to implement the visualizations are talked about.

Chapter 5 talks about various limitations and the future work related to this project.

Chapter 6 concludes the purpose and implementation of the project.
Chapter 2

Problem and Related Work

Music visualization is an artistic way of rendering the features found in the music in real-time. Media players or music visualizers generate visual effects based on the real-time audio input. The imagery or graphics generated are synchronized with the music as it is played. Examples of various historical music visualizers are: Atari Video Music launched by Atari Inc. in 1976, then computer music visualization came into existence in the 1990s as desktop applications such as Winamp (1997), Audion (1999), and SoundJam (2000) etc [2].

2.1 Google Chrome Music Lab

Adding to it, in 2016, Google launched a web browser application named as Chrome Music Lab. It comprises of hands-on experiments including visualizers. Apart from the visualizer’s capability to be an entertainment source by whisking one away to an imaginary world, it can now be used as a medium to teach music to beginners. Chrome Music Lab and its experiments are making learning music fun and more accessible [3]. This open-source project by Google aims to make the learning of music more interesting by letting users to experiments with the visual representations [4]. All these experiments and many more are listed on a website called Chrome Music Experiments. Through these projects/initiatives, Google encourages all the developers to use and contribute to their community as a way of learning from each other. These experiments inspired me to create one by myself. I started by researching on how to create a spectrogram. A spectrogram is a kind of visualization of sound that displays its frequencies. It plots them as low to high or high to low
as these frequencies are variable over time. The work to extract these frequencies is done by applying a Fast Fourier Transform to a sound signal, which separates the frequencies and amplitudes of the audio. The result is plotted/visualized by displaying the amplitudes as the colour and frequencies on the vertical axis and time on the horizontal axis [5]. Few blogs and projects that I followed were: Exploring the HTML5 Web Audio: visualizing sound by Jos Dirksen [6], Web Audio Spectrogram by Boris Smus [7], and Calculating BPM using Javascript and the Spotify Web API by José M. Pérez [8]. These tutorials helped me in kick-starting this project.

2.2 Web Audio API

Moving from the desktop applications like Winamp or Media players to now a web-browser application (or SAAS apps) has its own advantages in terms of being cross-platform, architecture, better user experience, no installations or upgrades etc. Now, to playback music on the web, earlier we needed to use the `<bgsound>` tag to automatically play the music whenever someone visits the webpage. However, this feature was only supported by Internet Explorer. After this, `<embed>` tag by Netscape came into the picture with the similar functionality. The first cross-browser way to play the music was introduced by Flash from Adobe requiring a plug-in to run. To overcome that limitation, HTML5 came with a `<audio>` element that also can be used in any modern browser. However, with the `<audio>` element, one cannot apply real-time effects, analyze sounds, have timing controls or have pre-buffering of the music. Hence, to address these limitations several APIs have been created. One of those APIs is the Web Audio API that is designed and prototyped in Mozilla Firefox [9].

Web Audio API is for processing and making audio in web applications. So, the extraction of the music data is made easier by integrating your application with the Web Audio API. This API can be used in various audio-related tasks, games or any web related interactive applications.

2.2.1 The Audio Context

AudioContext is an interface in Web Audio API that represents an audio-processing graph built from the linking of various audio modules. These audio modules are represented by AudioNode.
Initializing an Audio Context

The Web Audio API is currently only supported by the Chrome, Firefox and Safari browsers (including MobileSafari as of iOS 6) and is also available for developers via JavaScript. So, as the API is not yet stable, the audio context constructor is vendor-prefixed in these browsers, meaning that instead of creating new AudioContext, you create a new webkitAudioContext or mozAudioContext. Once the API is stable, this constructor could be made un-prefixed or will be prefixed depending upon the other browser vendors.

With this in mind, we can initialize the audio context in a rather generic way that would include other implementations (once they exist):

```javascript
var contextClass = (window.AudioContext ||
window.webkitAudioContext ||
window.mozAudioContext ||
window.oAudioContext ||
window.msAudioContext);
```

2.2.2 The Frequency Domain

Graphs of sound where the amplitude is plotted to see how it varies over the frequency are said to be in the frequency domain. Sound waves are cyclical in nature. So mathematically, periodic sound waves are the sum of multiple simple sine waves of different frequencies and amplitudes. By adding together more such sine waves, we can get a better approximation of the original function. Fourier transformation can be applied to get the component sine waves from a signal. Fast Fourier Transform (FFT) is one of the algorithms that can get this decomposition. And, Web Audio API has a direct function to achieve that.

2.3 Spotify API

To provide the user with a search for a song, a Spotify look-up has been implemented. This Web API lets the web applications fetch data from the Spotify music catalogue and lets them manage user’s playlists and saved music [10].
2.4 Polymer 2.0 framework

Polymer is an open-source JavaScript library launched by Google for building web applications using Web Components \[^{11}\]. Its feature of creating reusable custom components promotes loose coupling in the code. These components interoperate seamlessly with the browser’s built-in elements. It divides the app into right-sized components, making the code cleaner and less expensive to maintain \[^{12}\].

2.5 Three.js library

Three.js library has been used to add animations to this project. Three.js is a cross-browser JavaScript library and API that is used to create and render the animation in a web browser. It is open-source and uses WebGL \[^{13}\].

In order to be able to actually display anything on the browser with three.js, we need three things: scene, camera and renderer. Meaning, rendering the scene with the camera \[^{14}\].
Chapter 3

Design

This project has been made as a single-page web application. Since there is no data to maintain and for the heavier business logic we are integrating third-party libraries so this application does not have any back-end or database layer. The visualizer produces the data in real-time and the web-page renders it as animations, spectrogram and time-domain graph.

3.1 Technologies/Libraries Used

To develop this single-page application, Polymer 2.0 framework has been used. The choice to use this framework was made due to its several features over the vanilla Web Components, such as: Simplified way of creating custom elements, Both One-way and Two-way data binding, Computed properties, Conditional and repeat templates, Gesture events [11]. Custom elements not only make the code cleaner but also makes it more readable. For example:

Using a custom element of spectrogram could be done by something like:

```html
<v-spectrogram-core><v-spectrogram-core/>
```

It can have attributes/bindings/properties as well:

```html
<v-spectrogram-core labels log oscillator><v-spectrogram-core/>
```

So, for the user interface (UI), HTML5, CSS, JavaScript have been used as scripting languages and Polymer has been used as the framework. For animations, the three.js library has been used. With three.js, a scene, a camera and a renderer must be created. The scene acts like a canvas where we can place objects, lights or cameras
to be rendered by three.js. The camera just creates a new camera. However, the camera is an abstract base class in three.js. So, if a new camera has to be created then we need to call the other inheriting classes like PerspectiveCamera or OrthographicCamera. The renderer in three.js is of three types: WebGLRenderer, WebGLRenderTarget and WebGLRenderTargetCube. Their functionality is to display the completed scenes using WebGL [14].

Apart from using all the UI libraries, Web Audio API and Spotify API have been integrated into the code. Web Audio API has been used to take-in the audio, analyze it, extract the data and play it back in the web browser. Spotify API has been used to get the tracks corresponding to the text written in the search bar.

### 3.2 Data Extraction

The live audio is taken-in with the help of the browser’s microphone. Once the user allows the mic permission, they can sing or talk to it and the visualizer will generate the visual effects based on the audio taken-in. There are two samples of audio provided in the application and a Spotify song look-up has also been implemented. These options will play the user selected/searched songs and visualize them. A beat tracker has also been embedded along-with the Spotify look-up. To extract the relevant data from the audio signal, the AnalyserNode of the Web Audio API has been used. AnalyserNode is an interface in Web Audio API that represents a node, which is able to compute the real-time frequency and time-domain analysis information. It passes the incoming audio input through it as it is, but also gives out the analyzed data for the user to take it and generate the visualizations based on it [15] as shown in the figure 3.1.

AnalyserNode has methods to get the frequency or time domain data as an array of values. These methods are:

1. **getByteFrequencyData()** — copies the current frequency data into a Uint8Array (unsigned byte array) passed into it.

2. **getByteTimeDomainData()** — copies the current waveform, or time-domain, data into a Uint8Array (unsigned byte array) passed into it.

3. **getFloatFrequencyData()** — copies the current frequency data into a Float32Array array passed into it.
4. `getFloatTimeDomainData()` — copies the current waveform, or time-domain, data into a `Float32Array` array passed into it.

### 3.3 Application

The visualizer web application has been deployed on firebase at [https://visualyzic.firebaseapp.com](https://visualyzic.firebaseapp.com). On start-up of the application, the browser asks for a mic permission. The code-snippet is shown in figure 4.1

By default, the spectrogram visualization starts to run. The user can select the full-colour option to get a better visual of the spectrogram as shown in 3.3. The ticks on the right side of the screen show the frequency scale. The user is able to increase or decrease the number of ticks. The user can also change the speed of the spectrogram going from right to left on the screen.

The controls on the left let the user select the kind of visualizer. The user can select the time domain visualization, which in turn enables another visualization of the time domain. The two types of visualizations in time domain are shown in 3.4 and 3.5.

The user can also select the animation setting, which lets the user view a 3d animation. This 3d animation is a visualizer that responds to the volume, beat and waveform of the audio signal. The 3d scene can also be viewed from any perspective by moving the mouse on it. The 3d visualization is shown here in figure 3.6.

A Spotify look-up of songs is also integrated with this application, which fetches...
the first result from the matched songs to be played and also detects its beat per minute (BPM). The panel is shown in figure 3.7.
Figure 3.3: Spectrogram with and without colour

Figure 3.4: Visualization 1 of Time domain
Figure 3.5: Visualization 2 of Time domain

Figure 3.6: 3d Visualization
Figure 3.7: Beat tracking of Spotify tracks
Chapter 4

Implementation

4.1 Application Architecture

This web application is built using Polymer framework, HTML5, CSS and JavaScript. To keep the UI simple and modifiable; material designed UI experience through PolymerElements has been achieved. Node Package Manager (npm) and bower have been used to install the packages and dependencies. npm has also been configured to build and start the app on a local server. Travis CI has been used to build, test and deploy the app at firebase. The package.json file for npm helps others to look at all the direct dependencies used for the project. The package-lock.json file is automatically generated by npm to list all the direct and indirect dependencies needed for the project. It is read by npm when the installation is run the next time to automatically install all the libraries. The polymer.json is a configuration file for the polymer framework. It has all the commands to run the builds and tests in a Polymer framework.

Firstly, after npm is installed, the installation of Polymer CLI must be done from npm. Then, to have a boilerplate code for a Polymer-based project, Polymer recommends using its starter kit. The starter kit can be installed by running a command:

```
$ polymer init polymer-2-starter-kit
```

This generates the polymer.json file. The main entry point of the project would be index.html. Polymer can also serve up this project on localhost by running a command:

```
$ polymer serve --open
```

The project structure using this framework is shown in 4.1.
<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bower.json</td>
<td>bower configuration</td>
</tr>
<tr>
<td>bower_components/</td>
<td>app dependencies</td>
</tr>
<tr>
<td>images/</td>
<td>folder to keep the png/jpg</td>
</tr>
<tr>
<td>index.html</td>
<td>main entry point into the app</td>
</tr>
<tr>
<td>manifest.json</td>
<td>app manifest configuration</td>
</tr>
<tr>
<td>package.json</td>
<td>npm metadata file</td>
</tr>
<tr>
<td>node_modules/</td>
<td>npm installed packages</td>
</tr>
<tr>
<td>polymer.json</td>
<td>Polymer CLI configuration</td>
</tr>
<tr>
<td>build/</td>
<td>generated build of the project</td>
</tr>
<tr>
<td>service-worker.js</td>
<td>auto-generated service worker</td>
</tr>
<tr>
<td>src/</td>
<td>app-specific elements</td>
</tr>
<tr>
<td>v-spectrogram-core.html</td>
<td>top-level element</td>
</tr>
<tr>
<td>v-spectrogram-core.js</td>
<td>js file for business logic</td>
</tr>
<tr>
<td>v-audio-load.html</td>
<td>sample views</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>sw-precache-config.js</td>
<td>service worker pre-cache configuration</td>
</tr>
<tr>
<td>test/</td>
<td>unit tests</td>
</tr>
</tbody>
</table>

Table 4.1: Table describing the Project structure
Once npm install and bower install are run in the project, they create two folders `node_modules` and `bower_components` respectively. npm install also creates the package-lock.json file. The entry point of the application, i.e. index.html calls the top-level element from the src folder. This top-level element in this application is v-spectrogram-core.html. This element then calls other custom polymer elements in the src folder to make the desired application. Hence, any new custom element will go inside the src folder.

Git is being used as the version control system for this project. The source code for this application is uploaded to GitHub and made open-source for anyone to see, use and contribute to it. Proper documentation of the code is done and README file has been written for anyone to configure the application on their system and make use of it.

### 4.2 Creating Visualizations

We first initialize a canvas for the rendering of the real-time data on the page. Then we take the input from the user by the help of the microphone as depicted in the Figure 4.1. A 2d context is taken from the canvas.

```js
this.tempCanvas = document.createElement('canvas'),
console.log('Created spectrogram');
// Take audio input from the microphone.
if (navigator.mozGetUserMedia) {
  navigator.mozGetUserMedia({audio: true},
    this.onStream.bind(this),
    this.onStreamError.bind(this));
} else if (navigator.webkitGetUserMedia) {
  navigator.webkitGetUserMedia({audio: true},
    this.onStream.bind(this),
    this.onStreamError.bind(this));
}
this.ctx = this.$canvas.getContext('2d');
```

Figure 4.1: Taking audio input from the mic

After getting the input from the mic, it needs to pass through an analyzer so that it computes the frequency data for the audio, which gets rendered on the canvas. The code is shown in the Figure 4.2.

This render function calls the render function for different visualizations based
on the domain as selected by the user. For time-domain rendering, the time-domain data is fetched as an array by calling `getByteTimeDomainData()` of the analyser. This data is plotted on the rectangular context where the values are arranged based on the height. The `renderTimeDomain` function is shown in Figure 4.3.

For frequency-domain, a spectrogram is created. The frequency data is taken as an array by a call to the `getByteFrequencyDomainData` of the analyser. This data is then plotted on the rectangular context where the colour and height is given based on the frequency value. The canvas is given a scrolling effect from the right side to left by calling the translate function of the context. The code is as depicted in Figure 4.4.

For 3d animation, volume, beats and waveform are computed from the audio. The code is depicted in Figure 4.5.

This data is transformed into a visualization by using the three.js library. For this, a scene is created and the three.js objects are placed on the scene. These objects change their position or colour based on the features collected. OrbitControls let the user move the scene or zoom-in/zoom-out the 3d animation.
4.3 Adding Controls

The controls are added in the application for the user to do the following:

1. Select domain
2. Change visual effect
3. Add colour to the spectrogram
4. Change the scale to log
5. Increase/decrease the tick of the scale
6. Increase/decrease the speed of the canvas scroll

4.4 Spotify look-up

For integrating the Spotify API, the wrapper spotify-web-api.js is taken from the open-source project by José M. Pérez [16]. It is a client-side wrapper for the Spotify API. A look-up on the web-page is created that calls the wrapper to call the API and gets the list of tracks based on the user’s input. The first song in the list is chosen and the beat is detected for the same.

4.5 Adding Oscillator

An oscillator playing a sine wave is created by calling the createOscillator function of the context from the Web Audio API. On every mouse press event, an oscillator is created and the previous oscillator is deleted. On the mouse up event, the running oscillator is deleted. On every mouse move event, if the oscillator is there then the pointer is updated according to the mouse move. It also returns the frequency that it plays back. It can be used to measure internal latency [17].

4.6 Implementation Challenges

The application responds quickly and effectivley to the user’s live-input. The visualizations have a quick response time.
However, a few challenges were faced while the implementation of this application:

1. The application was previously started in Polymer v1.5. But, the version did not support the Polymer paper elements for UI such as paper-button, paper-radio-button etc. So, an upgrade to Polymer v2 was done. This upgrade resulted in a lot of changes in the project and code structure. Because Polymer v2 follows classes and objects type structure.

2. The upgrade to Polymer v2 brought some configuration issues.

3. For the continuous integration to be able to run, a lot of tools were explored and finally, Travis was integrated with the application. Its build automatically starts running when an update is pushed to the git repository. It also deploys the latest update automatically to the server (in this case, firebase).

4. Polymer framework being a really smooth and cleaner way to write code does have a bit of downside to it. It is a new technology that is not yet stable. Not a lot of documentation is there on the web about it. So, when faced with any issue it was harder to get through it.
renderTimeDomain() {
  var times = new Uint8Array(this.analyser.frequencyBinCount);
  this.analyser.getByteTimeDomainData(times);

  var WIDTH = this.width;
  var HEIGHT = this.height;
  for (var i = 0; i < times.length; i++) {
    var value = times[i];
    var percent = value / 256;
    var barHeight = this.height * percent;
    var offset = this.height - barHeight - 1;
    var barWidth = this.width/times.length;
    if(this.visual){
      this.ctx.fillStyle = 'rgb(200, 200, 200)';
      this.ctx.fillRect(0, 0, WIDTH, HEIGHT);

      this.ctx.lineWidth = 2;
      this.ctx.strokeStyle = 'rgb(0, 0, 0)';

      this.ctx.beginPath();

      var sliceWidth = WIDTH * 1.0 / this.analyser.frequencyBinCount;
      var x = 0;
      for(var i = 0; i < this.analyser.frequencyBinCount; i++) {
        var v = times[i] / 128.0;
        var y = v * HEIGHT/2;
        if(i == 0) {
          this.ctx.moveTo(x, y);
        } else {
          this.ctx.lineTo(x, y);
        }
        x += sliceWidth;
      }

      this.ctx.lineTo(this.width, this.height/2);
      this.ctx.stroke();
    } else {
      this.ctx.fillStyle = 'black';
      this.ctx.fillRect(i * barWidth, offset, 1, 1);
      this.ctx.beginPath();
    }
  }
}

Figure 4.3: Rendering Time Domain
renderFreqDomain() {
    var freq = new Uint8Array(this.analyser.frequencyBinCount);
    this.analyser.getByteFrequencyData(freq);

    var ctx = this.ctx;
    // Copy the current canvas onto the temp canvas.
    this.tempCanvas.width = this.width;
    this.tempCanvas.height = this.height;
    // console.log(this.$canvas.height, this.tempCanvas.height);
    var tempCtx = this.tempCanvas.getContext('2d');
    tempCtx.drawImage(this.$canvas, 0, 0, this.width, this.height);

    // Iterate over the frequencies.
    for (var i = 0; i < freq.length; i++) {
        var value;
        // Draw each pixel with the specific color.
        if (this.log) {
            var logIndex = this.logScale(i, freq.length);
            value = freq[logIndex];
        } else {
            value = freq[i];
        }
        ctx.fillStyle = (this.color ? this.getFullColor(value) : this.getGrayColor(value));

        var percent = 1 / freq.length;
        var y = Math.round(percent * this.height);

        // draw the line at the right side of the canvas
        ctx.fillRect(this.width - this.speed, this.height - y, this.speed, this.speed);
    }

    // Translate the canvas. to have scrolling effect <-
    ctx.translate(-this.speed, 0);
    // Draw the copied image.
    ctx.drawImage(this.tempCanvas, 0, 0, this.width, this.height,
                  0, 0, this.width, this.height);
    // Reset the transformation matrix.
    ctx.setTransform(1, 0, 0, 1, 0, 0);
}

Figure 4.4: Rendering Frequency Domain
renderAnimationScene() {
  // init options
  this.volSens = this.volSens || 1;
  this.beatHoldTime = this.beatHoldTime || 45;
  this.beatDecayRate = this.beatDecayRate || .9;
  this.beatMin = this.beatMin || .2;
  this.levelsCount = this.levelsCount || 16;

  this.freqArray = new Uint8Array(this.analyser.frequencyBinCount);
  this.timeArray = new Uint8Array(this.analyser.frequencyBinCount);
  this.dataArray = new Float32Array(this.analyser.fftSize);
  this.analyser.getByteFrequencyData(this.freqArray);
  this.analyser.getByteTimeDomainData(this.timeArray);
  this.analyser.getFloatTimeDomainData(this.dataArray);
  this.bufferLength = this.analyser.frequencyBinCount;

  this.waveform = this.getNormalizedWaveform(); // used for the plane geometry in scene
  this.levels = this.getNormalizedLevels(); // used for nothing but volume
  this.volume = this.getAverageVolumeLevel(); // used for the ball in scene
  this.isBeat = this.getBeatTime(); // used to change the color of the ball on every beat
}

Figure 4.5: Collecting features for 3D animation

The code snippet initializes an animation scene with various options and configurations, including frequency and time arrays, data arrays, and volume levels. It then uses these configurations to render and animate a scene, setting up the camera and scene background in 3D space.

Figure 4.6: Setting up the Scene

The second snippet sets up a 3D scene with a perspective camera, a three-color background, and camera controls. It also sets up a point light and a material for a sphere, rendering a plane geometry with a specified material and plane geometry. The scene is configured to display a sphere and a plane in a specified width and with a double-sided wireframe.
// change sphere color every beat
if (this.isBeat) {
    var color = this.sphereColors[this.activeColor];
    this.sphereMaterial.color = new THREE.Color('rgb(' + color.r + ', ' + color.g + ', ' + color.b + ');');
    this.activeColor = this.activeColor < this.sphereColors.length - 1 ? this.activeColor + 1 : 0;
}

// change sphere size based on volume
this.sphere.scale.x = .3 + (this.volume / 2);
this.sphere.scale.y = .3 + (this.volume / 2);
this.sphere.scale.z = .3 + (this.volume / 2);

this.waveform.forEach(function(value, i) {
    if (i % 2 === 0 && this.plane.geometry.vertices[i/2]) {
        this.plane.geometry.vertices[i/2].z = value * 86;
        this.plane2.geometry.vertices[i/2].z = value * 86;
        this.plane3.geometry.vertices[i/2].z = value * 86;
        this.plane4.geometry.vertices[i/2].z = value * 86;
    }
}).bind(this));
this.plane.geometry.verticesNeedUpdate = true;
this.plane2.geometry.verticesNeedUpdate = true;
this.plane3.geometry.verticesNeedUpdate = true;
this.plane4.geometry.verticesNeedUpdate = true;

// re-render scene every update
this.renderer.render(this.scene, this.camera);

Figure 4.7: Animating the Scene
Chapter 5

Limitations and Future Work

5.1 Limitations

Some of the limitations of the project are listed below:

1. Making it work in mobile browsers: Right now, it only works on the larger screens but the work is being done for it to be able to work on smaller screens as in mobiles.

2. Uploading an audio file: Feature to upload an audio file to this application is not there right now but can be added. This audio file when uploaded can be played and be visualized.

3. Importing/saving of audio with visualization: The visualizations are not getting saved or imported right now. They work with the real-time data only. They should be able to get saved into the application for later use and get downloaded and imported as well.

4. Feature to Zoom-in/zoom-out: The spectrogram right now cannot be seen on a finer/more-precise level as it does in the sonic visualizer application.

5. Beat detection: For now, beat detection is done only on the songs that are searched from Spotify API. However, we should be able to detect the beat of an audio file played to it.
5.2 Future work

Almost all of these limitations described above can be seen as the work for future of this application. A database can be connected to it to save the visualizations. Audio files can be uploaded to the application to view their visualizations. The data of the time-domain plot and spectrogram can be downloaded/exported in a CSV file of a particular audio. Feature to add 3d objects to the scene can be added for the user to see different visualizations as desired.

The code can be containerized as a docker image, which will make it easier to pull and run on any machine. It can be hosted on paid services (like AWS, Azure) that could make it available to a large number of audience.

In future, this application can be extended to mobile browsers and the FFT results could be made more precise by following a better algorithm instead of making the use of the one by Web Audio API.
Chapter 6

Conclusion

This project is a Software as a Service application that provides a platform to the user to observe the visual effects generated by the live audio input. This can come handy to the game developers or interactive application builders for several reasons. A good audio analyzer or visualizer can serve as a debugging tool to make the music sound just right. Moreover, the visualizations are enjoyed as a source of entertainment in the music players or applications like Guitar Hero. Hence, this project can be used to serve all these purposes.

The major challenge faced while implementing this project was in the configuration of different APIs in the Polymer framework, which is rather a new framework by Google and it lacks in proper documentation or solutions over the web.

So far, the visualizer only provides one 3d animation. Work can be done by allowing the user to select from multiple 3d animations or drag-drop objects into the scene and make a custom animation.
Bibliography


