CS 4621, Spring 2018	Final Project Report
Name: Wayne Li	NetID:wyl6

WebGL Audio Visualizer

1 Introduction/Goals

The goal of this application is to create a fun and interesting way to visualize audio using the WebGL concepts and skills that I have learned in the practicum course this semester. The visualization method that the application uses is to display a user inputted triangle mesh, centered at the origin, and displace the vertices of the mesh in the direction of their normals (note, the mesh must be indexed from 0, not 1!). The application takes in audio data in one of two ways:

1.) An uploaded audio .mp3 file

2.) Streamed audio from the user through the computer's microphone

The audio is processed and analyzed by the application, with the assistance of the Web Audio API, and a Fast Fourier Transform is performed. The application takes in the outputted frequency data and maps each frequency bin to a vertex. The intensity of that frequency value then corresponds to how much that vertex is translated. In this way, the mesh is displaced in sync with the audio being played/streamed. The overall effect is then an interesting visualization of said audio. Additional features that are added on to the application are then quality-of-life features that help the user navigate the application or additional parameters that allow the user to change the appearance of the mesh. One of the biggest goals of this project that I accomplished was to learn how to use Frame Buffer Objects and double buffering in order to implement a bloom filter. This gives the appearance of the mesh "glowing" and radiating off light. While I was able to accomplish most of the goals that I set for myself for this project, there are some additional features that I did not have time to fully implement, such as a skybox and an audio-mixer/editor. Perhaps they can be added in to the project at a future date.

2 Description and Relation to Areas of Computer Graphics

2.1 Animation

Animation is a big part of this application as the visualization of the audio has to happen in real time. As the audio changes, so too does the mesh for that particular point in time. The canvas essentially redraws itself in every single frame update as it reads in the new audio data for that point in time. Since each frame update happens extremely fast, the transition between frames appears smooth and connected. In addition, there is a small animation in face display mode involving "strobe" lights. Essentially, there are three light sources (1 red, 1 green, 1 blue) that surround the displaced mesh in a roughly circular manner. A simple Phong shading model is then used to illuminate the mesh with these lights, which rotate at a constant speed in a circle around the mesh. As a result, the mesh is given the appearance of an animated, spinning disco ball, as the lights change over time.

2.2 Interaction

The application supports various user interactions to allow the user to change what is being displayed on the screen. There are a number of basic camera controls (which has a fixed camera target at the origin):

- The up and down arrow keys control moving the camera towards and way from the origin, respectively.
- The left and right arrow keys rotate the camera left and right around the origin, respectively.
- The page up key translate the camera in the upwards direction (+y axis). The page down key translates the camera in the downward direction (-y axis).
- Scrolling up with the mouse wheel decreases the field of view of the camera (zooms in). Scrolling down with the mouse wheel increases the field of view (zooms out).

In addition, the user-input audio mode relies heavily on user interaction, as they provide the live audio that is being visualized in real-time. Whatever sounds they make into their microphone directly control what is shown on the application's canvas; loud sounds by the user displace the mesh more than no sound (barring noise of course). There are also several smaller parameters that the user can set, which alter the appearance of the mesh. Users have the choice of selecting between two display modes:

1.) Face mode displays the faces of the triangle mesh, along with 3 spinning strobe lights.

2.) Edge mode displays only the edges of the wire mesh. In this mode, there are no lights but rather, each edge is given a simulated "fluorescent" effect. The user also has control over how wide this florescent effect should be and what color the edges should give up through additional parameters in the side bar.

2.3 Modeling

The application also contains a small amount of modeling aspects, as it must read in, parse, and create the mesh from the .obj file that the user has inputted. In addition, the application must generate the mesh precisely as specified, and modify the mesh constantly to create the desired visualization effect.

2.4 Rendering

In order to implement the "fluorescent" effect of the wire mesh in edge display mode, the application makes use of a bloom filter to post-process the scene. After rendering the original scene to a frame-buffer object, the post-processing bloom filter is applied and the resulting image is redrawn to the screen (more specific details in the next section).

3 Framework Structure

The overall structure of the code base is broken down into 4 separate files, index.html, webgl.js, interaction.js, and audio.js. Each file is supposed to handle that particular aspect of the application's implementation.

- index.html contains all of the html for the user interface of the application. All of the glsl vertex and fragment shaders are located in this file. This file also handles the main drawing loop as well.
- audio.js contains all of the functions needed to set up the Audio Node Graph and audio nodes. The FFT is called in this file as well and the calculations for the displaced vertices are performed in functions in this file.
- interaction.js handles most of the user input data, including JQuery event handlers and reading and parsing user-input or parameters. This file also handles reading in and parsing the user obj mesh.
- webgl.js is meant to handle the majority of the webgl function needed for initialization and drawing commands. There are various functions here to pass information to the shaders and command them to render the scenes.

4 Implementation and Technical Details

There are two important technical details that I implemented in this project. The first is figuring out how to displace the meshes in a manner that reflects the audio being played. To do this, I make a call to the Web Audio API to perform an FFT on the slice of audio data at a particular frame. I am returned an array of frequency data, which I then map to each vertex in the mesh. Since the bin count must be a power of two, the way I implemented it was so that there will always be more bins then vertices; any leftover bins in the higher frequencies is simply discarded. We then take each frequency and normalize it in the range [0..1]; this is then used as scale by which the vertex is translated in the direction of its normal. We then take these transformed vertices and pass them to the vertex and fragment shaders, where they are ultimately rendered and displayed to the screen.

The second large undertaking that this application does is to post-process the scene using a bloom filter for the edge display mode. To do this, I render the wiremesh scene to a frame buffer object, rather than the canvas. This scene gets stored to a texture, which I then render to a full-screen quad. Using a double buffer, I use a 9x1 Gaussian Kernel to blur the image in the horizontal direction and then a 1x9 Gaussian Kernel to blur the image in the vertical direction. I then take this blurred image, and add it on top of the original image, and render that to a full screen quad which ultimately gets displayed on the canvas. This produces a "fluorescent" effect for the wire mesh. The one drawback of successfully implementing this bloom filter is that the canvas sizes and texture sizes must be a power of two; as a result, our canvas will always be at a fixed size of 512x1024.

In addition, there were some fun HTML/CSS things that I had to learn for this project as well in order to improve the overall theme and feel of the application.

5 Instructions to Run

In the local directory containing all of the files, run a simple web server using:

python -m SimpleHTTPServer

Then using a web browser of choice such as Chrome, navigate to localhost:8000 and the user interface for the WebGL application should pop up. The page should contain a large, horizontal ≡ button at the top of the screen and also a blank, black canvas. Click on the ≡ button and a sidebar should pop up with display and audio settings. Here you may upload a text file containing in .obj mesh with vertices, vertex normals, and faces in the form of "v//vn" only. The mesh should then pop up on the screen. You will also be allowed to upload an audio .mp3 file to visualize. Or, if you choose the user-input mode, you can visualize the audio that is recorded by your computer's microphone. For the display settings, there is face mode, which displays the faces of the triangle mesh, or edge mode, which displays only the wire mesh. For your convenience, there are several sample mesh files and audio files in the "/assets" folder that you can use for the application.

6 Acknowledgments and Resources

This application uses and acknowledges the following sources:

- w3schools for their w3.css file, which is used to design the user interface, including the sidebar
- The Web Audio API library for all of the audio analysis functionality. This includes setting up the audio sources, creating the Audio Node Graph, creating the analyzer node to read in and preform the FFT on the audio source, and examples of how to use the API, etc.
- The glMatrix and JQuery libraries for matrix operations and user interaction, respectively
- "https://github.com/mattdesl/lwjgl-basics/wiki/shaderlesson5" for inspiration in how to set up the fragment shaders that I eventually used for our bloom filter
- Stanford Bunny Mesh: "https://graphics.stanford.edu/ mdfisher/Data/Meshes/bunny.obj"
- Suzanne Mesh: "https://github.com/OpenGLInsights/OpenGLInsightsCode/blob/master/Chapter%2026%20Indexing%20Multiple%20Vertex%20Arrays/article/suzanne.obj"
- Eston Schweickart for his great general advice and assistance
- CS 4261 lecture exhibits for inspiration in setting up our framework including boilerplate code and frame buffer objects/double buffers

7 Progression of Project and Changelog

For more information about the overall progression for this project, including all of the things implemented, please see the README.md and git log in the source code framework. Thanks!