Interactive Audio Visualizer using p5.js

Installation instructions

Development Environment:

We recommend **Visual Studio Code (VS Code)**.

Extensions:

Live Server: To run the project locally and view it in the browser, you need the Live Server extension for VS Code.

Running the Project:

- Inside VS Code, locate the index.html file in your project folder.
- Right-click on the index.html file and select **Open with Live Server** from the context menu.
- The project will automatically open in your default web browser, and any changes made to the code will be reflected in real-time.

Dependencies:

- The project uses **p5.js** and **p5.sound** for audio visualization and analysis. These libraries are included through a CDN link in the index.html file, so no additional installation is required for the libraries themselves.

High-level documentation

There are two visualization modes available to switch between while listening to music:

Dots:

The Dots mode creates a grid of moving dots that respond to the audio. The Dot class handles the individual movement and appearance of each dot, while the Dots class manages the entire grid, ensuring a smooth and coordinated display.

Dot class: Each dot's behavior and appearance are influenced by the audio input, particularly the amplitude (volume) and its oscillation based on an angular movement. The position of each dot is updated dynamically and the oscillation is directly affected by the audio's volume, meaning that as the sound changes, the movement of the dots accelerates or decelerates accordingly. The color of each dot changes depending on its position on the screen.

Dots class: manages a grid of Dot objects, organizing them into columns and rows across the canvas. The dots are arranged in a 2D grid, with their positions spaced evenly across the canvas. Each dot's movement is influenced by its distance from the center of the screen, and the farther the

dot is, the more intense the oscillation. The user can control both the speed and the oscillation effect through sliders.

Why this mode?

The moving dots serve as an intuitive representation of the audio's energy and rhythm. The fluid, responsive grid creates a dynamic and visually appealing experience that's directly tied to the changes in the audio signal.

Flowers:

The flowers mode is designed to display dynamic, audio-reactive flowers using Fourier analysis. Each flower consists of petals that respond to the frequency spectrum of an audio signal, in our case a song.

Flower Class: Spectrum is split into segments to represent different frequency bands for petals. Users can change the number of petals and the size of the flower using sliders. These adjustments dynamically affect how the flower appears and reacts to the audio.

Bezier curves: We chose Bezier curves for the petals because they give us a lot of control over the shape of the curves while keeping things simple mathematically. They allow to build easily flowing and symmetrical shapes for a flower, which are hard to achieve with simple lines or polygons. Also, by controlling just a few points (start, control and end point), these curves offer an efficient and precise manipulation of the shape.

splitFreq() function: This custom function divides the spectrum into smaller, logarithmically spaced bins, and calculates the weighted average of the frequencies in each bin. This method provides a more musical and natural representation of the spectrum than fft.linAverages() would achieve, emphasizing lower frequencies while spreading the higher ones more sparsely.

Flowers class: handles multiple flowers. Initially generates a variety of flowers with different positions and sizes. The class also includes functionality to generate new flowers at different positions with random sizes and colors, by clicking on the canvas.

Why this mode?

The flower visualization mode was chosen for its nature inspired design which offers a calming representation of the audio and creates a contrast to the more geometric Dots mode.

Sliders:

Sliders adjust key parameters for visual effects, such as speed, oscillation range, or petal count, offering users a hands-on approach to shaping the visuals.

Media Player Elements:

Provides all core playback functions:

- Play/Pause, Next/Previous buttons.
- Volume Slider, Progress Bar, Track Duration Indicator

Volume adjustment modifies visual intensity. Although, when the user sets the volume to zero, particles still have a slight movement.

Dots mode has additional stabilization. Since their movement is analogous to the volume, if the volume is less than maximum, we adjust the volume value to prevent limiting the movement. This means that a Dot's speed is proportional to the volume, while being inversely proportional to the Volume Slider's value.

Real-time analytics:

Waveform analysis represents the sound signal's amplitude.

Spectrum analysis divides the audio into frequency bands and visualizes their amplitude as vertical bars. This analysis gives an insight into the frequency distribution of the audio.

Bass frequencies visualization uses the energy of bass frequencies to visualize its significant changes.

Problems Encountered:

Integration with Streaming Platforms (YouTube/Spotify):

Initially, we wanted to avoid using local sound files by integrating a popular streaming platform, such as YouTube or Spotify, to play songs directly. This would allow users to select any song they wanted, making the experience more dynamic and user-friendly. However, we ran into the issue that there was no available API to stream raw audio data from these platforms. YouTube, for example, doesn't provide access to sound data in a way that would allow us to analyze frequencies in real-time.

Solution: To move forward with the project, we decided to use a predefined playlist of songs instead of relying on streaming. We implemented a user interface with media controls to navigate through the playlist and demonstrate the visualizer's functionality.

Frequency Range Issues in Flower Mode:

During testing of the flower mode, we noticed that the high frequencies were hardly visible in the visualizer, while the lower frequencies were exaggerated and louder than expected. This wasn't the desired outcome, as we wanted the visualization to more evenly represent the audio spectrum.

Solution: After examining the issue, we realized that we didn't need to use the entire frequency spectrum to visualize the flower's petals. We could focus on a more limited frequency range to produce a more balanced result. To address this, we implemented the splitFreq() function (see more on high-level project documentation).

Machine learning implementation missing:

During the research phase of the project, we realized that implementing machine learning for mood or genre detection would be quite a heavy workload. It would take a lot of time and add complexity

that might distract from the main goals of the project. As a result, we chose to save the machine learning aspect for our thesis, where we can dedicate the necessary time and effort to properly develop and implement it. This will allow for a deeper exploration of machine learning techniques, which can later be integrated into the visualizer as an advanced feature.

p5.js onended() Bug:

We encountered a bug with the onended() function in p5.js, which is triggered twice when the audio playback is paused. This behavior was identified as a known issue with p5.js since 2021, and unfortunately, it had not been fixed.

Solution: To work around this bug, we removed the onended() function before pausing the audio and then re-added it after playback resumed.

Team members' contributions:

Both of us worked equally on the JavaScript code, including building the visualization modes and designing the overall UI.

Georgakas Georgios: Took care of the styling and layout using CSS and worked on creating the media player, including the audio playback controls.

Georgaka Evangelia: Focused on adding real-time audio analysis to the project. Was also responsible for researching how to integrate machine learning for mood/genre detection but decided to skip that part for now and explore it in her thesis.

Software resources used:

- p5.js and p5.sound: for visual elements and processing audio input.
- FontAwesome icons: used for UI elements (e.g. for the volume icon).
- **Firefox**: Browser used for testing and running the project.
- **Live Server (VS Code extension)**: For hosting the project locally and enabling real-time updates in the browser.

Research resources and tutorials:

- https://www.generativehut.com/post/using-processing-for-music-visualization
- https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aFcVjlDAkkGIixw70s7jpW
- https://medium.com/@konjkar/creating-an-interactive-audio-visualizer-with-vanilla-javascript-46640283b91b