Cornell University

CS 4621 Practicum Final Report

A♯ – Music Visualizer

Shane Moore
swm85

Zachary Zimmerman
ztz3

Emre Findik
ef343

Joseph Vinegrad
jav86

December 15, 2014

1 Summary

A♯ (A Sharp) is a music visualizer designed for meaningful sound information conveyance. While other visualizers have flashy and intricate animations that pleasantly accompany the music, they do not successfully convey, interperet, or even begin to replace the music. This is an enormous shortcoming of music visualizers — current applications certainly do not meet the standards of a music visualizer in the true sense of the term. A♯ attemps to make strides toward filling that gap, with hopes that other visualizers may follow suit.

A♯ models a song using a single sphere mesh. The sphere is animated through sets of transformations based on comprehensive data analysis of the song's sound file. The application analyzes several important features of a song, including the overall key, beat event times, and the frequency amplitude spectrum at each instant of the song. The results of the song analysis determines the appearance of the sphere during the playback animation.

2 Conceptual Mapping

In order to achieve an intuitive representation of the chosen song, the conceptual mapping of A♯'s visuals was designed with great care, as follows:

- Overall volume determines object size
- Beats are shown by pulses in object size
- Overall pitch controlls object elevation (higher up means higher pitch)
- Sound complexity (distribution of frequencies) determines object shape
- Mood is conveyed with color

3 Software Design

Our codebase is divided into two main modules: (1) Sound analysis and (2) Graphics.

(1) Sound analysis takes place as a pre-processing stage, before we display any graphics. We obtain data for the sound analysis in `app/analysis.py`. We utilize an audio and music processing library called librosa to do the bulk of the work in obtaining the data. The data we receive from this includes separated harmonic and percussive sounds, a melspectrogram (frequency amplitudes), tempo, beats, frame times, and a chromagram (???). All of these data sets are compiled for each frame in the song. Finally we perform operations to condense this data in order to make use of it in our graphics module (probably want more info here or in implementation section).

(2) Once we receive the sound data from analysis.py, we pass them along to our graphics module in `app/graphics.py`. First we setup our screen and initialize constants and data structures based on the sound data.

At each frame, we update features of the sphere, including radius, color, vertical positon, and displacement map. This information is passed into the shaders, `DispMapped.vert` and `DispMapped.frag`, which update the appearance of the sphere on screen.

4 Sound Analysis

tonic major/minor statistical analysis

separate foreground and background beat tracking and tempo

mel spectogram

average pitch as a weighted average

5 Graphics Implementation

physically based animation spring model

displacement mapping

contributions to radius

color cosine function for color scheme, synced with beats

For our graphics pipeline, we wrote vertex and fragment shaders for our sphere to set positional and color data. After receiving data for each frame from the sound analysis, we set the radius of the sphere according to beat pulse and sound amplitude. Then, we use frequency data to compute vertical translations of the sphere. The idea behind this is to create an upward movement of the sphere when the pitch rises and a downward movement when pitch falls. We model this vertical motion of the sphere using a damped harmonic oscillator.

We also use frequency amplitude data to set displacement magnitudes. We dynamically create a texture map using this data at each frame. Essentially, the objective here is to displace higher points on the sphere according to the amplitudes of high frequencies, and lower points on the sphere according to the amplitudes of low frequencies. Finally we set shader uniforms based on the computed radius and texture map at each frame.

6 Results

Conclusion