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.

The source code for this project, as well as any extensions to it, can be found at `https://github.com/Oneman2feet/a-sharp/`.

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 as 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 by the color of the object

3 Sound Analysis

Sound analysis takes place as a pre-processing stage, before the visualization is run. The sound data is collected in the module `analysis.py`. Much of the waveform analysis is done with the help of the audio and music processing library LibROSA.

The data received from LibROSA includes:

- The uncompressed waveform
- Separated harmonic and percussive waveforms
- Beat frames (list of time frames for which a beat event occurs)
- Mel Spectrogram (amplitude of each frequency bin over time)
- Chromagram (amplitude of each pitch-class over time)

From this initial data collection, more information about the song is gleaned. The frequency spectrum is truncated to a reasonable range and formatted as a displacement texture for the sphere. The spectrum is also condensed, via a weighted average, into a measure of the overall pitch at each instant. The overall key of the song is statistically predicted based on the prevalences of each pitch-class found in the chromagram, as discussed in `http://ismir2004.ismir.net/proceedings/p018-page-92-paper164.pdf`.

The final conclusions of the analysis are then formatted into a python dictionary for use in the graphics module.

4 Graphics Implementation

The results from `analysis.py` are recieved by the module `graphics.py`. This module controls playback of the visualization with the help of the windowing and multimedia library Pyglet.

The graphics module uses vertex and fragment shaders to set the sphere mesh's positional and color data. After initialization of the window and global variables and constants, the update function is scheduled to be called regularly by the Pyglet app. At each frame, the sphere's radius, position, color, and displacement map are calculated according to the current sound data. This information is passed into the shaders, `DispMapped.vert` and `DispMapped.frag`, which update the appearance of the sphere on screen.

The radius of the sphere is calculated as a sum of contributions

$$\text{radius} = \text{minimum radius} + \text{current volume} + \text{proximity to beat}$$

Where the proximity to the nearest beat is expressed as

$$\left(\frac{\text{time between beats}}{2} - \text{time since previous beat}\right)^2$$

in order to have the attack and decay of the beat appear as a pulse.

The vertical positioning of the sphere is affected by the overall pitch as determined in `analysis.py`.

blahblah

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.

physically based animation spring model

displacement mapping

contributions to radius

color cosine function for color scheme, synced with beats

5 Results

Conclusion

fallbacks

improvements