VisualizeR: Real-Time Audio Visualizer in MATLAB

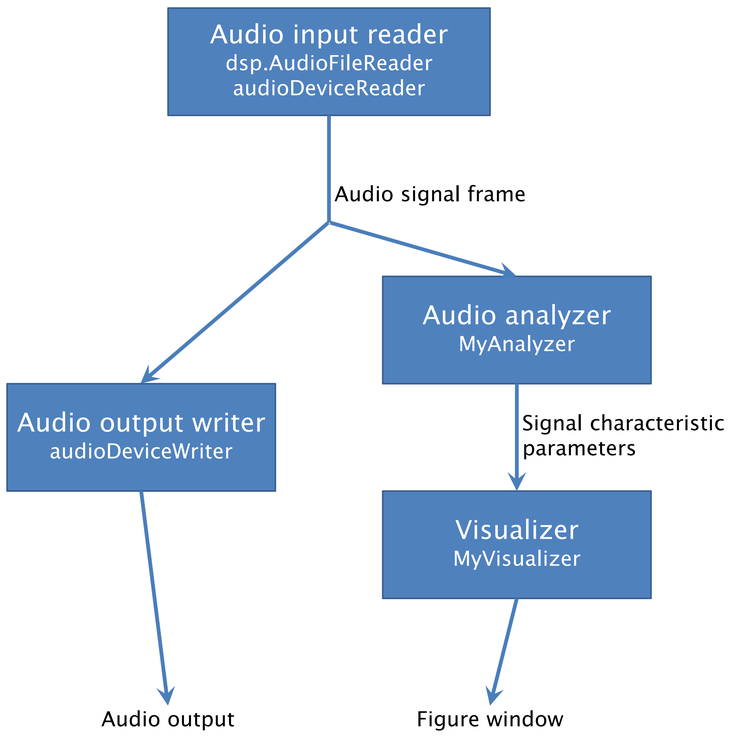
Rebecca Gillie, Vincent Mateo
AME 272 (Audio Signal Processing)

Overview

Many modern music player applications come with a built-in audio visualizer. An audio visualizer is a graphic that evolves in time with audio playback, changing according to the characteristics of the sound. These characteristics can include volume, frequency spectrum, spectral envelope, onsets, and tempo.

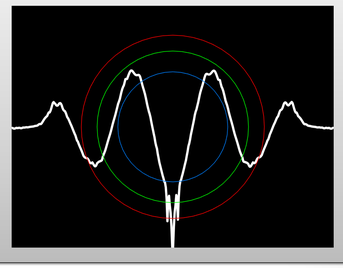
This project implements such an audio visualizer in MATLAB. Using MATLAB accomplishes the whimsical goal of transforming the dull MATLAB plot, with all its negative, schoolwork-related associations, into an object of artistic beauty. In addition, we show that the Audio Systems Toolbox and MATLAB's normal plotting functions are sufficient for implementing the project in real time with low latency, preventing the need to interface with another language or API.

Structure



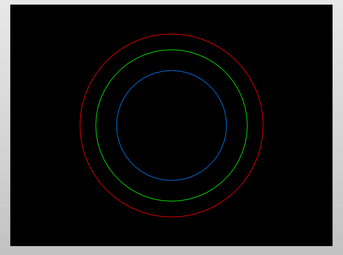
Graphic display

The graphic display is implemented in a custom class called MyVisualizer. This class uses a figure window to display three circles and a waveform against a black background. The menu bar and toolbar are hidden in the figure window.



Circles

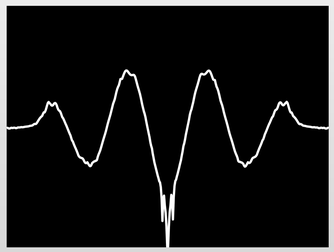
There are three circles: red, green, and blue. The sizes of the circles correspond to the average magnitudes of three frequency bands: bass (red), mid (green), and treble (blue). There is a peak-hold behavior built into the size of the circles, so that the movement is smoother and more pleasing. When a circle grows, its outline thickens according to the growth amount.

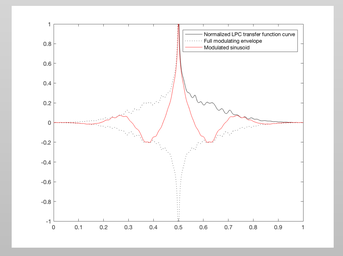


Waveform

The rear waveform (top) is a sinusoid modulated by an envelope. This envelope is a normalized version of the spectral envelope, concatenated with its mirror image (bottom). The spectral envelope is calculated using 50th-order linear prediction. For the purposes of the envelope curve shape, both the frequency (x-axis) and amplitude (y-axis) are in linear scale.

The phase of the carrier sinusoid increments with each frame at a frequency of one cycle per second. Currently, this frequency in constant; in the future, it will vary based on a tempo-related analysis parameter, e.g. tempo estimation or onset detection.





Audio I/O

The Audio Systems Toolbox provides two ways for the project to input audio. The first is from an audio file, read frame-by-frame in real time using dsp.AudioFileReader. The second is from the sound card, also read in real time using audioDeviceReader. This second way allows the project to process microphone input. Both classes have the same step() function syntax to read a frame of audio, making it easy to switch between the two.

Audio output is performed in real time using audioDeviceWriter (also from the Audio Systems Toolbox), by passing the input frame unchanged to the device writer's own step() function.

Even though the audio signal passes through unchanged, frames must still be taken so that the audio analysis and graphical rendering can occur in time with those frames. Good visual results are achieved with a frame length of 512 samples, which produces just 11.6 ms of latency.

Audio analyzer

The audio analyzer is implemented in a custom class called MyAnalyzer. It processes a single, mono frame of audio at a time. After it has processed this frame, it outputs the parameters used to control the circles and waveform.

These are the analysis techniques used to generate the output parameters:

- Spectrum analysis: The frequency spectrum of the frame is taken using the built-in FFT function, and then the average absolute value is taken over each of the three bands (bass, mid, and treble). The averages are converted to decibel scale before being passed to the visualizer.
- Spectral envelope: The linear prediction coefficients are computed using the built-in LPC function with an order of 50. These coefficients are converted into a frequency-response curve using freqz. This curve is normalized to a range between 0 and 1 before being passed to the visualizer.

Results

A pop music recording was used to test file input. The red circle followed the bass drum hits in a very regular pattern, whereas the other two circles varied more unpredictably, since there were more sonic elements in the mid and treble ranges. The complexity of the sound made it difficult to predict the envelope changes.

Microphone input was tested as well. Clicks and snaps produced large, noticeable visual changes. For voice input, vowels and singing tended to trigger the green circle, while shrill consonants (e.g. "s" and "k") triggered the blue circle. Steady vowel sounds caused the rear waveform to exhibit a nice-looking envelope.

Conclusion

The visualizer implemented in this project gives a listener visual insight into the characteristics of an audio signal. These insights should trigger associations with the listener's sonic observations—for example, the circles, representing frequency bands, could be associated with specific instruments. Calling attention to frequency content in this way is especially helpful to an untrained listener.

That said, plenty of audio metering tools already exist for those looking for strict visual-to-sound correspondence. The visualizer in this project presents a degree of abstraction, such that the visual elements are only suggestions of sonic characteristics. This allows for some artistic interpretation on the part of the listener.