VISUALIZER: REAL-TIME AUDIO VISUALIZER IN MATLAB

Vincent Mateo and Rebecca Gillie

University of Rochester, Audio and Music Engineering

ABSTRACT

A real-time audio visualizer is implemented in MATLAB using the Audio Systems Toolbox. The visualizer produces graphics that more directly reflect changes in the sound, compared to similar consumer products, while maintaining a degree of artistic creativity and interpretation.

1. INTRODUCTION

Audio visualizers are included in many modern music player applications such as Apple's iTunes and Windows Media Player. As pointed out by Keyes and Wierckx [1], the images in these visualizations typically change as the amplitude and frequency characteristics of the sound they are processing change. However, the properties of the image that are changed seem arbitrary. While these visualizations are pleasing to look at, they don't convey information about the audio in a way that is easy to comprehend. Our goal was to create a real-time visualizer that clearly shows information about the audio input to give the viewer an idea of what they are hearing while also being visually appealing. The aspects of the audio we chose to visualize are the spectral envelope and the energy contained in the bass, mid-band, and treble ranges of the sound.

2. METHOD

Implemented entirely in MATLAB, VisualizeR takes audio input either from an audio file specified or from a microphone input. It processes the audio by dividing it into frames of 512 samples. After a frame is read, it is passed both to the audio output and to the analyzer by `main.m`. The analyzer calculates the spectrum of the signal and converts this into three frequency band magnitudes. It also calculates the spectral envelope curve using linear prediction coefficients. The results of the analyzer are then passed to the visualizer that implements the information in a MATLAB figure. An example output of the visualizer is shown in Figure 1.

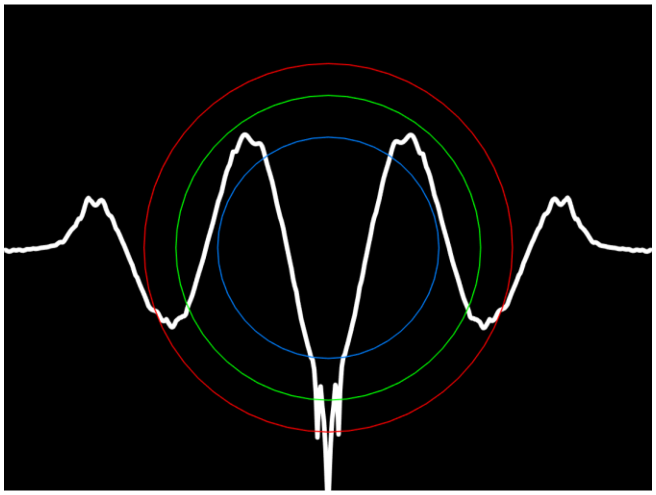
Figure 1: Full VisualizeR image

2.1. Audio Input

The audio input from a sound file is processed by a `dsp.AudioFileReader` object from MATLAB's DSP System toolbox. Audio input from the microphone is processed through an `audioDeviceReader` object from MATLAB's Audio System toolbox. Both objects stream the audio information in real time. In each iteration of the while loop of `main.m`, the `step()` function is called on the reader object, loading the audio input for one frame. If the audio is a stereo file it is converted to a mono signal. `main.m` then feeds the new frame information between the `MyAnalyzer` and `MyVisualizer` objects.

2.2. Analysis

A custom class called `MyAnalyzer` is implemented and used to process frames of audio. `main.m` calls functions on a `MyAnalyzer` object each time a new audio frame is processed.

For spectral analysis, each frame is ten times zero-padded and the discrete Fourier transform is calculated using MATLAB's built-in `fft` function. The spectrum is broken into three frequency bands, and the average absolute value of the spectrum within each band is stored as the magnitude. The bass band contains frequencies from 20 to 200 Hz, the mid-band from 200 to 2000 Hz, and the treble band from 2000 to 20,000 Hz.

The spectral envelope is estimated by linear prediction using the autocorrelation method as described in DAFX [2]. MATLAB's `lpc` function is used, with an order of 50, to return the coefficients of a filter that models the spectral shape of our signal. The coefficients are converted to a filter curve representing the frequency response of our audio using MATLAB's `freqz` function, and the curve is passed to the visualizer.

2.3. Visualization

The frequency band magnitudes and spectral envelope curve are passed to a `MyVisualizer` object. The custom class `MyVisualizer` controls a figure window and allows for manipulation of the items plotted in it.

2.3.1. Circles

A circle represents each of the three frequency bands. Each circle is created using a stem object with a single point at (0.5,0), and the circle is the stem marker. The circle size is controlled by the magnitude of its corresponding band, passed from `MyAnalyzer` and converted to a dB scale. The red circle shows the information for the bass frequency band, the green shows the levels for mid-band frequencies and the blue for treble frequencies. For example, in Figure 2 the magnitude of the bass frequencies is higher than mid-band and both are higher than the treble.

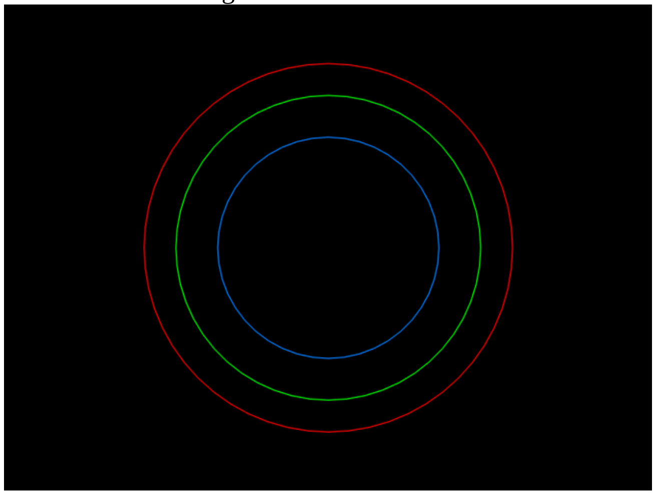

Figure 2: Frequency Bin Magnitudes

To make the figure changes less flashy and distracting from the music, the shrinking rate of the circle when the magnitude of that band decreases is limited. The minimum size of the circle in one frame compared to the previous is:

$$old\ size \times e^{-1/\tau}$$

In this expression, τ is a time constant in units of frames; approximately 86 frames is equivalent to 1 s. The result is a pleasing peak-hold behavior.

When a circle grows by any amount, in order to accentuate the growth, the circle outline thickens by that amount, up to a certain maximum. This is shown in the green and blue bands of Figure 3. The outline resets to its normal thickness when the circle shrinks.

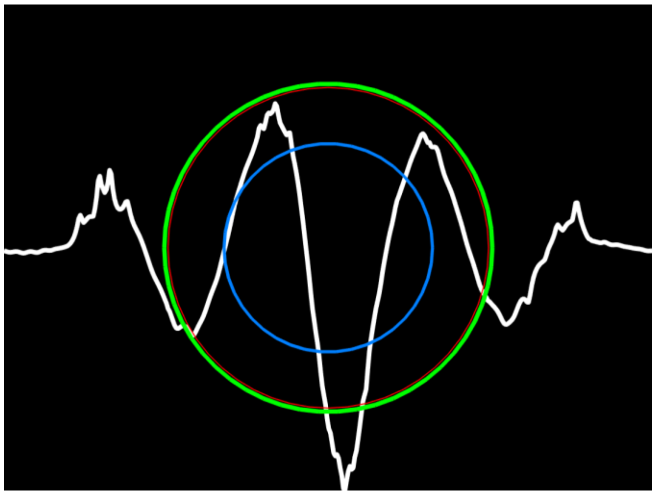

Figure 3: Full VisualizeR image

2.3.2. Waveform

The background waveform, shown in Figure 4, is a carrier sinusoid modulated by an envelope. The sinusoid has a wavelength of 0.25, so that four wavelengths are included in the figure window. The phase of this sinusoid increments with a frequency of 1 cycle/s.

The modulating envelope is constructed from the spectral envelope filter, passed from `MyAnalyzer`. The filter curve is normalized to the range [0,1] and concatenated with its mirror image to produce a symmetrical envelope. Because the carrier sinusoid has the range [-1,1], the modulating envelope serves as an upper boundary for the waveform, and the negative of the envelope serves as a lower boundary. The envelope, its negative, and the modulated sinusoid can be seen in Figure 5.

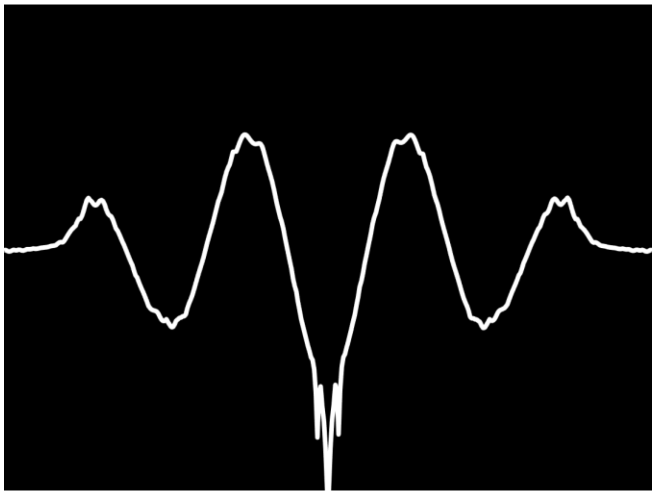

Figure 4: Visualizer waveform

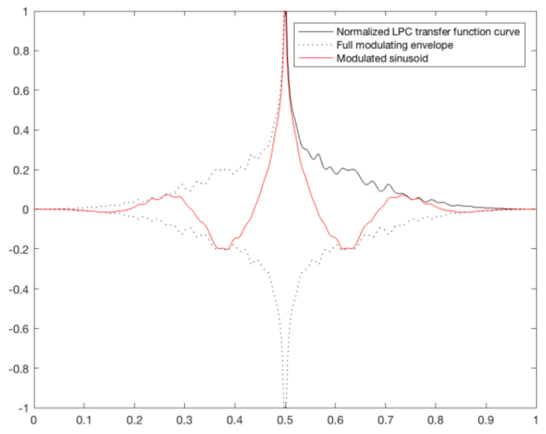

Figure 5: Envelope to visualizer waveform

2.4. Audio Output

To play the audio along with the visualizer, an `audioDeviceWriter` object from the Audio Systems Toolbox is used. In each iteration of the while loop of `main.m`, the `step()` function is called on this object, sending one frame of audio signal to the output device. The `step()` function is called before the input is summed to mono, so the audio output will continue to be stereo if the input is a stereo signal.

3. TESTS

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.

4. CONCLUSIONS

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.

5. FURTHER WORK

A feature to be added to our visual rendering is tempo information of the audio. Through the course of this project we struggled to find a good method to estimate the tempo for a real-time input. We didn’t want to store too much audio information or increase our frame length as this would increase the processing time. Increased processing would prevent us from creating the visual image quickly enough that it could play in time with the audio.

If we can find a method to estimate tempo in real-time we would match the rate of the phase shift for the background sinusoid to this tempo. It currently is set to a constant 1 cycle/s. Adjusting the speed of this waveform with the tempo would provide more visual information to the viewer increasing their perception of the audio.

6. REFERENCES

[1] Keyes, Christopher J., Wierckx, Marcel “The ArrtsSync Project: Methods and Architetures for Mapping Foreground, Middle-ground and Background Musical Structures to Visual Images,“ International Computer Music Association, Web, 2006.

[2] D. Arfib, F. Keiler, U. Zölzer, V. Verfaille. *DAFX: Digital Audio Effects* , “Chapter 8: Source-filter processing,” , John Wiley & Sons, pp. 279-300, May 2011.