Music Waveform and Spectrum Visualizer

Benjamin Curths - EE384

Motivation

- Audio is Everywhere
  - Music is a fundamental part of everyday life and is extremely relatable to the average person.
- Data is Beautiful
  - A visual representation of the data that comprises a digital audio stream can be very appealing, especially when synchronized with the encoded audio. Combining visual and auditory stimuli provides a more enriching experience than either alone.
- Experience Promotes Understanding
  - Working with audio files and performing operations with the contained data can provide a deeper understanding and familiarity of how the data is encoded and what it represents.

Background

- Two Formats of Interest
  - .WAV files are a direct encoding of the digital signal values at each sample point of the data signal. They contain leading header data, usually around 44 bytes in size, that contains information about the encoding such as the format designator, bitrate, sample rate, data size, etc. Following this header is a linear stream of the (usually) uncompressed digital sample data aligned by sample time in the case of multiple channels (samples 1a, 1b, 2a, 2b, …, *n*a, *n*b for *n* samples per channel.)
  - .MP3 files (MPEG Layer-3) are compressed audio and much smaller in size than an equivalent .wav file. They are also considerably more complex, consisting of a series of *frames* that each contain a frame header that describes the characteristics of the following compressed data (such as bitrate, sample rate, stereo encoding, or error protection). These frames are not defined in size, but each header begins with a sync code to indicate the start of a new frame.
- Fortunately, MATLAB handles the reading of the encoded audio data for us!

Procedure

Three Primary Steps

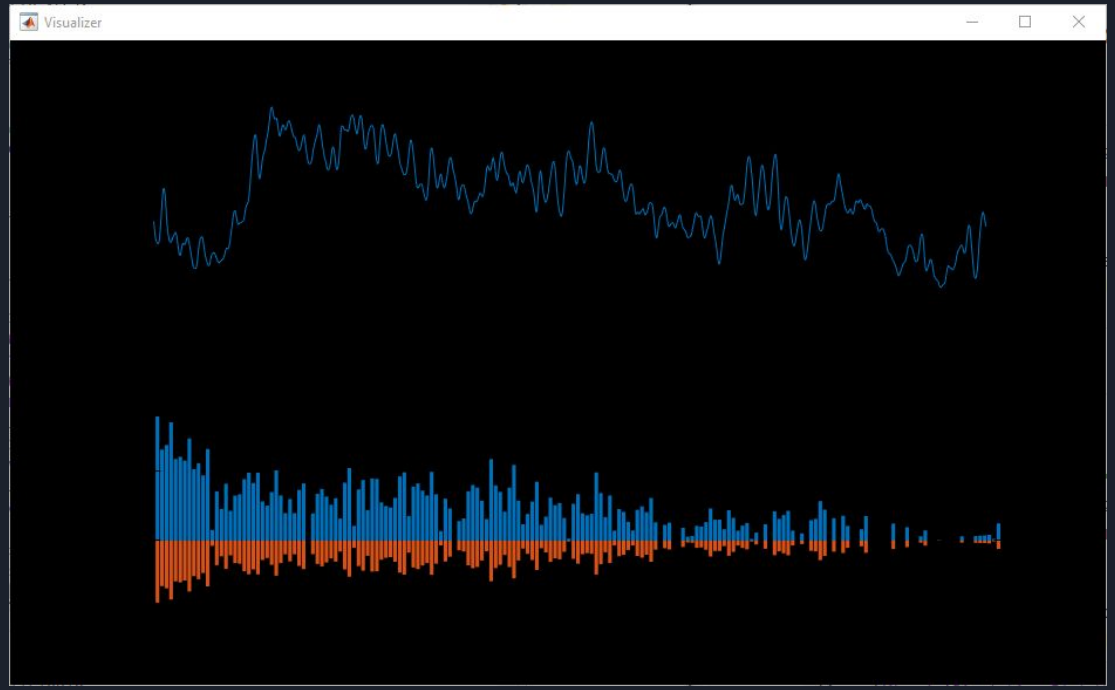
1. Initialization
    1.1. Prompt for a file and read in the audio data.
    1.2. Initialize the player object and variables needed for computation.
2. Configure the Display Window
3. Main Processing Loop
    3.1. Resync and compute the processing window indices.
    3.2. Perform FFT of samples within the processing window.
    3.3. Convert complex FFT values to real power in dB.
    3.4. Plot the waveform of the current sample data.
    3.5. Plot the frequency power spectrum of the current sample data.
    3.6. Repeat the processing on the next time window until the end of the file.

Results

Runtime Output:

Waveform

Power Spectrum



Challenges

- MATLAB figure formatting is unintuitive.
  - Much more time than should have been necessary was spent learning to navigate the window configuration for MATLAB's figure displays. The properties are viewable, but the methods for setting them from within a code script are not as well documented as they could be. Several third-party sources were referenced to determine the correct procedure for getting the desired results, and they're still not perfect.
- Multiple options exist for frequency spectrum scaling.
  - It was difficult to decide which frequency representation to use such that the plot was representative of the original data while being visually appealing. Both the spectral magnitude and unscaled power representations left too much unutilized space in the display window, but initial attempts to use the logarithmic dB scale provided unappealing results, especially with negative dB values. The final decision was to use the dB scale with a dynamic upper range limit and a lower limit of zero, trimming negative values in favor of visual appeal.

Conclusion

The primary purposes of this project were achieved with visual display of both the raw data waveform and real-time spectral analysis of the audio stream. However, there is still room for improvement. Desired features for an expanded version include frequency-band filtering for both improved speed of spectrum analysis versus a standard FFT and real-time audio equalization as well as a more interactive player interface that includes the ability to pause or resume playback and processing. At the moment, there is no graceful way to terminate the program flow before the end of the data stream. A more interactive design would solve this issue, but regrettably the development time for implementing the necessary features was allocated elsewhere.

Overall, this project was a fine opportunity to refresh and improve our understanding of both the Fast Fourier Transform and some of the most common audio file formats in a manner that was both engaging and appealing to the senses even if the result was not, at the end, practically useful.