# ENHANCING MUSIC PERCEPTION THROUGH VISUALIZATION

**Cyrus Parsons**
UVic
cyrusp@uvic.ca

**Michael Schmidt**
UVic
michaelschmidt@uvic.ca

**Nick Baker**
UVic
nicholasbaker@uvic.ca

## ABSTRACT

This project is a music visualization application that transforms audio into dynamic visuals. Users can upload audio files and listen to their music accompanied with elaborate graphics responsive to various musical elements including tempo, frequency, and key. We implement multiple unique visualizers utilizing various techniques such as dancing roving particles, waveform monitors, and sine waves. This project was built in python, utilizing **Librosa** for feature extraction, **NumPy** for efficient computations, **TKinter** for front-end GUI, **Pygame** for generating visuals, and **Pandas** for our backend. We support various audio formats, developed and tested with open-source datasets like Free Music Archive (FMA) and Last.fm free music downloads. Our tool creates an immersive, visually engaging experience which enhances the music listening experience and allows users to visualize and recognize fine details of the music they love.

## 1. INTRODUCTION

Our proposed project is a GUI-based application that will allow users to upload sound files, listen to them, and view visualizations synchronized with the audio playback. The purpose of our project is to generate accessible visuals that enhance users' listening enjoyment and provide some insights into the underlying music - helping them to understand the tempo, instruments, and interactions of various frequency ranges. We have created 4 different visualizers:

1. A traditional EQ bar based visualizer, with frequency averaged into bins and mirrored on the x-axis.

2. A Mel-spectrogram-based visualizer, with a spectrogram in the background, and a pulsating circle in the foreground, which changes size in proportion to the song's volume and changes colour on beats.

3. A sine-wave-based visualizer, which changes its amplitude and wavelength in response to the volume, frequencies, and tempo of the music. In particular, the amplitude of the waves is linearly proportional to the volume of the song, except for during a frame with a beat, where the amplitude of the waves have a brief spike. Also, increases in amplitude in the bass frequencies in particular cause the waves to change phase at a slower rate, and for them to be drawn with proportionally thicker lines.

4. A particle drum feature based visualizer. Whenever a kick drum is hit, red particles emit moving quickly away from a random point on screen. Whenever a snare drum is hit, blue particles are emitted. When a hi-hat is hit, white particles appear at random points on the background. On every beat, a flash occurs with a color generated based on the observed frequencies in the song at that moment. Frequencies are separated into either lows, mids, or highs, and a background color is generated based on the relative amplitude of each frequency bucket. For instance, if there are more lows than mids or highs, the background color will be very blue. Conversely, if there are more highs than mids or lows, the background color will be very red. If there's a mix of the two, the background color would be purple. The brightness of the flash varies with the overall amplitude of the song, so at a loud point the flash will be very bright.

There is a wealth of related works in the visualizer field, talked about more in-depth in the following "Related Works" section. The visuals we create are intended to supplement existing visualizers and make them more engaging and insightful. In terms of the features needing to be extracted from the music to produce accurate visualizations, there is also a plethora of related works - as well as some class materials. We aim to recreate existing work in this section of the project. The brunt of our novel work will be in creating engaging visuals and an intuitive user experience.

## 2. RELATED WORKS

Much interesting work has been done in the vibrant and burgeoning field of music information retrieval, and the explorations which have been done into music visualization have come to stand as an integral part of this intriguing area of study. Visualizing music is not only aesthetically pleasing, adding an additional artistic dimension to the pieces, [1] especially with the dynamic interplay of movement and colours, [1–4] it also has been shown to have the

potential to add value as a means of scientific analysis - uncovering insights about the pieces' musical characteristics via the interplay of our visual and auditory senses. [5–7] All the way back in 1979, J.B. Mitroo et. al. were some of the first researchers to investigate the use of musical visualization, sparking a discourse that would only grow more and more ubiquitous in the field as time went on [8]. Besides using visualizations to explore or add to the artistic qualities of pieces or to uncover emergent musical characteristics, they can also be used to analyse the differences in various pieces in a new, interesting, and educational light; differences such as genre [9,10], volume dynamics, [6] and tones. [2, 3, 11] This comes together to bring about perhaps the most fundamental and important aspect of music visualization: a means to enhance our understanding and appreciation of the pieces and of music in general from a holistic and encompassing scientific perspective. [11–15]

## 3. METHODOLOGY

This project is built in Python using several libraries. First, Librosa is used to process and analyze the audio and to extract target features from the songs for visualization. Librosa provides essential functionality for MIR projects. Numpy is used to enable efficient computation with large arrays and matrices, such as for some audio feature arrays. Numpy integrates well with Librosa to perform signal processing and manipulation. Tkinter is used for the front-end admin panel, where users can select a visualizer, and upload and play songs. Pygame is used to drive our visualizations, providing dynamic graphics which synchronize with audio playback. Pygame also allows for real-time user interaction in future work, providing a potentially immersive and customizable experience. Finally, we use Pandas for creating our database and backend functionality.

For the data used in development and testing, we will source it from the Free Music Archive (FMA) dataset available on GitHub (https://github.com/mdeff/fma). Additionally, we will use Last.fm's free music downloads for additional access to songs (https://www.last.fm/music/+free-music-downloads).

## 4. TIMELINE

There are 4 main sections of work:

- Creating the GUI itself
- Extracting features from user-uploaded sound files
- Creating the afore-mentioned visualizations
- Testing and fine-tuning

We tentatively intend to follow this schedule:

1. Begin the GUI skeleton and implement music file management and parsing.
2. Following the completion of (1), begin work to complete the GUI, and begin work to create visualizations.
3. Following the completion of (2), integrate the visualizations with the GUI to create a functional proof-of-concept.
4. With (1)-(3) finished, begin to test and iterate on existing features, pursuing stretch goals as necessary.

We hope to spend roughly 15% of our time on item (1), 60% on item (2), and 15% of our time on item (3). This will leave 10% of our time for item (4) and any other issues that may arise.

## 5. ROLES

The roles for this project have been tentatively outlined as follows:

- Nick: responsible for GUI for file management and music player windows
- Michael: responsible for basic functionality of backend, including file management and parsing music into metrics for use by the visualizers. Also, for making certain visualizer functionality.
- Cyrus: responsible for flushing out backend functionality, tweaking file management and adding calculations for about half of the metrics needed. Also, for making the bulk of visualizer functionality and integration with the GUI window.

Each team member will contribute to some deliverable for each schedule item outlined above.

## 6. MARCH SNAPSHOT

The following outlines our progress as of March 17, 2025.

### 6.1 Progress

We have created a minimum viable product of our project which exercises all of the basic functionality we would expect for the finished project. We have created a responsive GUI using Tkinter and PyGame which allows users to upload and play .wav files, with a set of sample files included with the program. When playing a song, a bar-style EQ visualizer is displayed at 30 fps based on the observed frequencies and amplitudes.

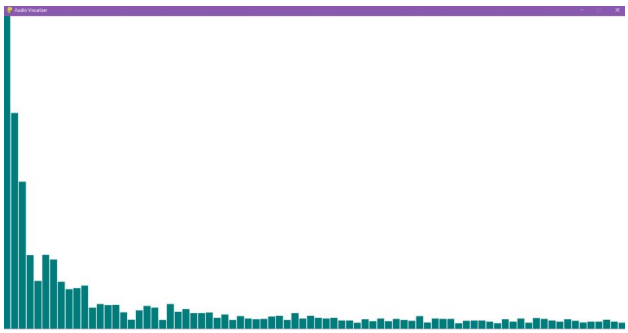

**Figure 1**. A screenshot from our initial dynamic visualization.

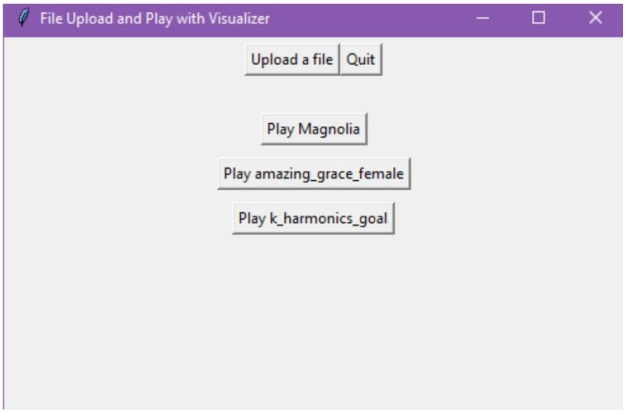


**Figure 2**. Initial Menu GUI.

The remaining work lies in extracting more audio features (tempo, key, etc) via MIR techniques, creating more advanced visualizations that respond to those audio features, and creating a more intuitive UI for the user.

### 6.2 Lessons

We learned the importance of managing code through a version control system like Git, establishing team-wide improvements to the code by writing and committing to the repository. We also learned the importance of storing system dependencies (like numpy version) as we had issues with an outdated version of scipy and numpy allowing the code to work on one computer but not on another. We tried a couple different options for creating the GUI, and tkinter seemed to be the most intuitive for what we wanted to do: while PyGame seems great for creating visualizations, there was a lot of boilerplate code needed to make a responsive GUI that Tkinter does not need.

### 6.3 Goals Moving Forward

- **Basic goals:**
  - Allow users to manually define song title and artist at time of file upload.
  - Conduct MIR at the time of file upload to parse, compute, and store data needed for visualization.
  - Polished GUI with modern feel.
  - Implement two styles of visualizer from the four listed in the introduction section.
- **Expected goals:**
  - Add a GUI for the visualizer player allowing users to play, pause, and skip back and forth through the song and corresponding visuals.
  - Implement three styles of visualizer from the four listed in the introduction section.
- **Advanced goals:**
  - Incorporate more complexity in the audio-visual relationship, for example, isolate individual instruments or sound types (e.g., vocals, drums, bass) from the audio and assign unique visualizer elements (e.g., shapes, colors, or motion patterns) to each sound.
  - Implement all four styles of visualizer listed in the introduction section.

## 7. REVISED TIMELINE

Now that we have made progress and fleshed out the important areas, our priorities have shifted from de-risking all of our project's components to implementing as many features as possible to achieve our goals. As a result of the progress we've made, we have revised our timeline and it has become more fine grained. This is the schedule we intend to follow:

1. Successfully extract MIR features needed for basic and expected visualizations - we will need to compute and store the tempo, volume, tones, and keys of the song.
2. Implement our visualizations based on the additional MIR features we now have access to - all visualizers described should be able to be implemented at this point but we will choose three of the four to implement at this point to meet our expected goal.
3. Update the GUI to have a more modern feel, incorporating pause/play buttons and a better file explorer menu. We will need to incorporate the existing 2 windows into a 1 window setup with several dropdowns and customizable UI options like light / dark mode.
4. At this point we will have completed our basic and expected goals and will be pursuing more advanced topics. We will attempt to extract further MIR features such as separating the melody and harmony lines and iterate on our existing visualizers with this data (or potentially implementing the fourth visualizer).

With respect to our remaining time, we intend to spend approximately 10% on item (1), 50% on item (2), 20% on item (3), and 20% on item (4).

## 8. FEATURES

1. **Beats**

   We use beat tracking to estimate the timing of beats in the songs. Our implementation leverages Librosa's beat tracking function **beat_track** with some added preprocessing for better results. We found that by first isolating the percussive component of the signal using harmonic-percussive source separation (HPSS) the beat tracking accuracy improved. This is because the drum-like sounds of the percussive signal are more likely to correspond to the musical beats compared to other components. We then convert beat times from frames to seconds using Librosa's **frames_to_time** function, and store this into a .csv for use by the visualizers.

2. **Drum Hits**

   For rhythmic detail beyond beat tracking, we detect specific drum-hits: kicks, snares and hi-hats.

   (a) **Kicks and Snares**

   We detect kicks and snare hits by analyzing energy in specific frequency bands using the Short-Time Fourier Transform (STFT) via Librosa's **stft** function to obtain a time-frequency representation of the audio signal. We first tried using the Discrete Fourier Transform (DFT), but found STFT to produce better results due to the short, punchy nature of drum sounds. We then convert the resulting amplitude spectrogram into decibel scale using Librosa's **amplitude_to_db** to enhance contrast between low and high-energy regions making the peaks easier to distinguish. To isolate relevant frequency content, we use Librosa's **fft_frequencies** to go from time-domain to frequency-domain for frequency analysis. This allows us to create frequency masks to isolate bands corresponding to kicks (typically 20-150 Hz) and snares (typically 150-2500 Hz). We then sum the energy within the masked regions over time to obtain an energy curve. After smoothing the curve using a Savitzky-Golay filter (**scipy.signal.savgol_filter**) to reduce noise, we use **scipy.signal.find_peaks** to identify the likely kicks or snares. Finally, we convert the drum hit times from frames to seconds with Librosa's **frame_to_time**.

$$STFT = X(m,\omega) = \sum_{n=-\infty}^{\infty} x_n w_{n-m} e^{-i\omega t_n}$$

   (b) **Hi-Hats**

   After trying to identify hi-hats using the same method as kicks and snares focused on the hi-hats frequencies, we found that a different approach yielded better results. Hi-hats produce fast, bright transients concentrated in higher frequencies making them better suited to onset detection methods. This tracks sudden spectral changes rather than absolute energy. We used Librosa's onset strength envelope (**librosa.onset.onset_strength**) to measure the rate of spectral change over time and highlight moments of sudden energy increase. We then applied peak picking to this envelope using a dynamic threshold which we defined as the 93rd percentile of the onset strength values rather than fixed amplitude. This approach allows the system to adapt to the overall energy of the song, ensuring that peaks are only selected if they stand out *relative to the rest of the signal*. This is especially useful for detecting hi-hats as their intensity can vary greatly across different tracks. We also enforced a minimum time separation between peaks to avoid detecting multiple peaks for what should be one hit.

3. **Mel Filtered Spectrogram**

   We use the fast fourier transform (FFT):

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-i\frac{2\pi}{N}kn}$$

   to convert the song data into its frequency domain representation, and then dot product this with a corresponding Mel filter bank to project these magnitudes onto the Mel scale. This allows our spectrogram to more closely align with the human auditory spectrum, and is an approach widely used in tasks such as speech recognition and music analysis.

4. Frequency Amplitudes

   We again use the FFT to convert the playing song, frame by frame, into its frequency components. For visualizer 3, the bass and treble amplitudes calculation we calculate the mean of the frequency amplitudes between the range of 20-250 Hz for bass and 4000-20000 Hz for treble. The frequency bars are calculated in a similar fashion, except the ranges are defined by simply dividing the length of the frequency array by the desired number of bars. Then, the mean of each range is calculated and the height of the corresponding bar is calculated from there, normalized to fit on the screen. For visualizer 4, when calculating the frequency bins to blend a flash color together, we use FFT to separate the current wave into bins. Each bin is $\frac{sample\_rate}{chunk\_size}$ = 43 Hz wide, and we define "lows" as 0 - 430 Hz (bins 0 to 10), "mids" as 430 - 6450 Hz (bins 10 to 150), and "highs" as anything higher than that. The

bins were chosen based roughly on existing notions of what ranges “lows”, “mids”, and “highs” are and then tweaked to make the visualizer more more engaging.

## 9. RESULTS

### 9.1 Visualizer 1

The real time EQ bar graph, while simpler than the other visualizers, nonetheless offers a unique and dynamic window into the songs, showing how the frequencies of music evolve over time. This helps reveal the structure and balance of the sound, providing an intuitive glimpse into key musical elements such as rhythm, harmony, and instrumentation.

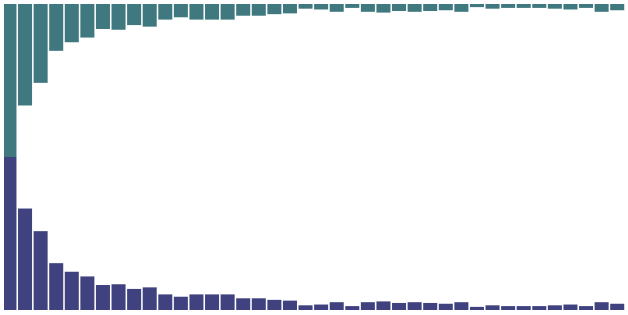

**Figure 3**. Frame from Visualizer 1

### 9.2 Visualizer 2

This Mel-spectrogram-based visualizer creates a smooth and interesting visual experience. The scrolling spectrogram in the background helps to see the song’s energy across different frequency ranges, giving a sense of the song’s progression and overall texture. In the foreground, a circle pulses with the volume of the track, growing and shrinking in sync with the music’s dynamics. The bands within the pulsing circle show how rapidly the song changes volume - thicker bands represent sharper changes, while smaller vibrations within the bands show the song sticking to a more constant volume. On each beat, the circle transitions to a new colour, providing a satisfying visual cue to track the rhythm. This visualizer is especially interesting for songs which have many layers or instruments as the spectrogram will display a wide range of frequencies to visually identify the various layers.

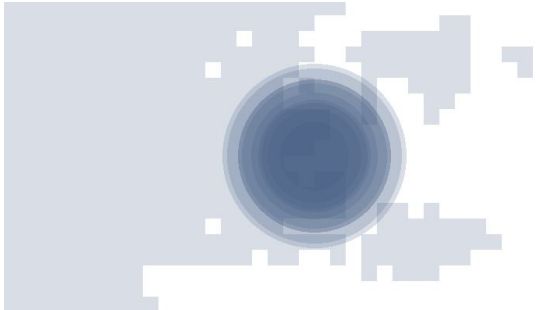

**Figure 4**. Frame from Visualizer 2

### 9.3 Visualizer 3

The sin-wave-based visualizer is another useful tool for visualizing music. The separation of bass and treble frequencies means that bass is clearly and pleasingly visually represented, with the sine wave slowing down and becoming thicker with higher amplitudes of bass. Beats are once again incorporated as they cause the waves to pulse in size, and volume is represented with larger waves overall. These elements together mean that listeners are afforded a pleasing and educational glimpse into the underlying frequency-based amplitudes that constitute their favourite songs.

**Figure 5**. Frame from Visualizer 3

### 9.4 Visualizer 4

This drum based visualizer is very intuitive to watch for more drum-heavy songs (like “Mars is Dangerous” in the samples, for instance). When listening to songs like that, you can clearly hear when each different type of drum hits and receive a satisfying visual cue to match your listening. The particles are very pleasing and create a space-like visual, making even simple songs engaging. This visualizer is perfect for listening to instrumental music, learning to identify different drum components, or potentially for developing a drum line.

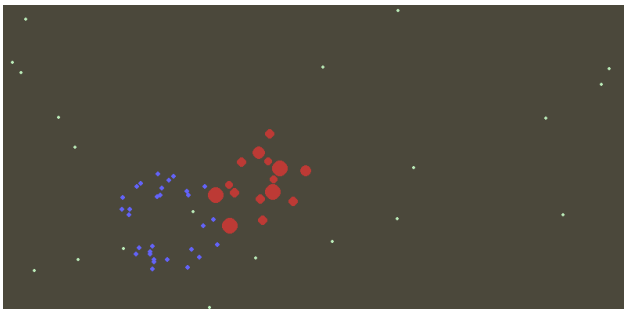

**Figure 6**. Frame from Visualizer 4

## 10. CONCLUSION

We created four intuitive, unique visualizers leveraging various MIR techniques to create pleasing visual representations of music. The MIR techniques included onset detection based beat tracking and drum detection, fast fourier transforms, and mel spectrogram analysis. The set of 4 visualizers allows a listener to visually experience the music

in any way they desire - whether they're interested in observing the respective treble vs. bass amplitudes of a song or they want to concentrate on an engaging drumline. Future work could include exploring other MIR techniques in a new visualizer (for instance, detecting the key of a song and creating visual effects based on major or minor) or perhaps combining existing visualizers to show an even deeper synthesis of musical features. Our code is available at https://github.com/nickbakeruvic/CSC475-visualizer.

## 11. REFERENCES


[1] J. H. Fonteles, M. A. F. Rodrigues, and V. E. D. Basso, "Creating and evaluating a particle system for music visualization," *Journal of Visual Languages & Computing*, vol. 24, no. 6, pp. 472–482, Dec. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1045926X13000566

[2] A. Mardirossian and E. Chew, "Visualizing Music: Tonal Progressions and Distributions," in *International Society for Music Information Retrieval Conference*, 2007. [Online]. Available: https://www.semanticscholar.org/paper/Visualizing-Music%3A-Tonal-Progressions-and-Mardirossian-Chew/769b9f5687f85d9c122c7852477200c1384b3a18

[3] P. Ciuha, B. Klemenc, and F. Solina, "Visualization of concurrent tones in music with colours," in *Proceedings of the 18th ACM international conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 1677–1680. [Online]. Available: https://doi.org/10.1145/1873951.1874320

[4] D. Politis, D. Margounakis, and K. Mokos, "Visualizing the chromatic index of music," in *Proceedings of the Fourth International Conference onWeb Delivering of Music, 2004. EDELMUSIC 2004.*, Sep. 2004, pp. 102–109. [Online]. Available: https://ieeexplore.ieee.org/document/1358106

[5] G. Bernardes, M. E. P. Davies, and C. Guedes, "Automatic musical key estimation with adaptive mode bias," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 316–320, iSSN: 2379-190X. [Online]. Available: https://ieeexplore.ieee.org/document/7952169

[6] N. Kosugi, "Misual: music visualization based on acoustic data," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, ser. iiWAS '10. New York, NY, USA: Association for Computing Machinery, Nov. 2010, pp. 609–616. [Online]. Available: https://doi.org/10.1145/1967486.1967581

[7] P. McLeod and G. Wyvill, "Visualization of musical pitch," in *Proceedings Computer Graphics International 2003*, Jul. 2003, pp. 300–303, iSSN: 1530-1052. [Online]. Available: https://ieeexplore.ieee.org/document/1214486

[8] J. B. Mitroo, N. Herman, and N. I. Badler, "Movies from music: Visualizing musical compositions," in *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '79. New York, NY, USA: Association for Computing Machinery, Aug. 1979, pp. 218–225. [Online]. Available: https://dl.acm.org/doi/10.1145/800249.807447

[9] R. Kamolov, P. Machado, and P. Cruz, "Musical flocks," in *ACM SIGGRAPH 2013 Posters*, ser. SIGGRAPH '13. New York, NY, USA: Association for Computing Machinery, Jul. 2013, p. 1. [Online]. Available: https://doi.org/10.1145/2503385.2503487

[10] K. Ohmi, "Music Visualization in Style and Structure," *Journal of Visualization*, vol. 10, no. 3, pp. 257–258, Sep. 2007. [Online]. Available: https://doi.org/10.1007/BF03181691

[11] P. Toiviainen, "Visualization of tonal content with self-organizing maps and self-similarity matrices," *Comput. Entertain.*, vol. 3, no. 4, pp. 1–10, Oct. 2005. [Online]. Available: https://doi.org/10.1145/1095534.1095543

[12] M. Taenzer, B. C. Wünsche, and S. Müller, "Analysis and Visualisation of Music," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan. 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8706365

[13] J. Snydal and M. Hearst, "ImproViz: visual explorations of jazz improvisations," in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '05. New York, NY, USA: Association for Computing Machinery, Apr. 2005, pp. 1805–1808. [Online]. Available: https://doi.org/10.1145/1056808.1057027

[14] Y. Shi and C. Yang, "Celestia: a vocal interaction music game," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: Association for Computing Machinery, Apr. 2013, pp. 2647–2650. [Online]. Available: https://doi.org/10.1145/2468356.2479485

[15] M. Cooper, J. Foote, E. Pampalk, and G. Tzanetakis, "Visualization in Audio-Based Music Information Retrieval," *Computer Music Journal*, vol. 30, no. 2, pp. 42–62, Jun. 2006. [Online]. Available: https://doi.org/10.1162/comj.2006.30.2.42