Synesthetic Music and Motion Visualization in Augmented Reality

Lucy Zhang

Advisor: Dr. Leslie Collins

April 15, 2018

Abstract

This paper describes the development of two variations of music visualizers that attempt to provide meaningful insight on a signal as well as a pleasant user experience. The first visualizer is web based and uses signal processing and machine learning to extract information such as the genre classification and instrument detection. Using these concepts, a second visualizer is created. This second visualizer is a mobile application that brings the visualization to augmented reality. The application incorporates the effects of a user's motion through utilizing the mobile device's built-in hardware sensors: the gyroscope and accelerometer. With machine learning, the user's motion type can be determined. This data is used to enhance the visualization by adjusting parameters based on motion.

Introduction

There have been a number of efforts to visualize music in both the signal processing domain as well as the generative media and art domain, where generative media is defined as media or art that has been created with the use of an autonomous system [14]. Existing music visualizers quantitatively visualize parameters such as the frequency spectrum and time domain amplitude signal. More academic visualizations have visualized other musical aspects including self similarity-based displays that allow the identification of structural and rhythmic characteristics [10]. These visualizations are largely static or developed with the sole purpose of providing music structure analysis and other quantitative insight on a audio signal.

Commercially, numerous music visualizers exist including desktop applications such as the Windows Media Player and the iTunes visualizer. However, a significant number of these visualizers often have trade-offs between displaying meaningful information from the audio signal and being aesthetically pleasing. For example, the iTunes visualizer is based on waveform input of the audio file and can have potentially misleading and uninformative visuals. Spikes in amplitude make the iTunes visual more elaborate and fill more of the screen while differences in frequencies are depicted by the size of the charged particles. How the audio input impacts the visualization is not clear. In fact the iTunes visualizer's selection of parameters such as color tend to be arbitrary [11].

Furthermore, desktop visualizers (such as iTunes and Windows Media Player) are strictly limited to the computer machine. For commercial applications such as enhancing concert experiences, using a mobile platform makes more sense due to its accessibility by a wider audience. Most mobile devices are equipped with sensors for location detection as well as rudimentary motion analysis such as pedometer functionality. Both the iPhone and the Android have accelerometer and gyroscope sensors as well as software based sensors such as the step counter. With these sensors as well as the GPU's computing power for augmented reality visualizations, a far more immersive mobile-based music visualizer can be created to enable a synesthetic experience for applications such as enabling the deaf to experience, motivating exercise to music, and enhancing music performances.

Methods and Results

Parameter Visualization

Features are extracted or computed from the audio file using the python scipy and wave modules as defined in Table 1. The visualization consists of several components, the main three being the center geometrical sphere that pulses according to amplitude, the first smaller surrounding ring that rotates faster or slower according to higher or lower entropy, and the second larger surrounding ring that depicts the frequency spectrum, as shown in Figures 1 and 2.

Audio Signal Information Classification

Genre Detection

Supervised learning is used to detect the genre of a song. A classifier is trained on the GTZAN music genre dataset which was used in a number of existing genre detection focused papers

Feature	**Description**
Zero Crossing Rate	The rate of sign-changes of the signal during the duration of a particular frame.
Energy	The sum of squares of the signal values, normalized by the respective frame length.
Entropy of Energy	The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
Spectral Centroid	The center of gravity of the spectrum.
Spectral Spread	The second central moment of the spectrum.
Spectral Entropy	Entropy of the normalized spectral energies for a set of sub-frames.
Spectral Flux	The squared difference between the normalized magnitudes of the spectra of the two successive frames.
Spectral Rolloff	The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
MFCCs	Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
Chroma Vector	A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
Chroma Deviation	The standard deviation of the 12 chroma coefficients.

Table 1: Audio features

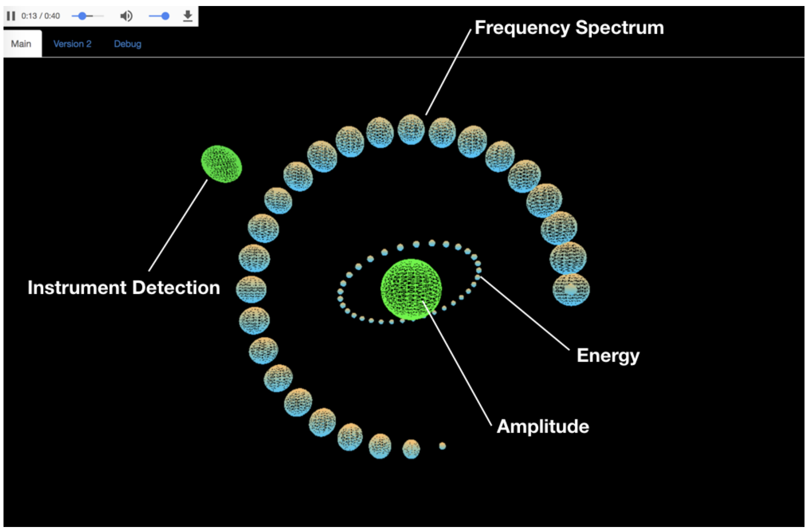

Figure 1: Image of web application music visualizer

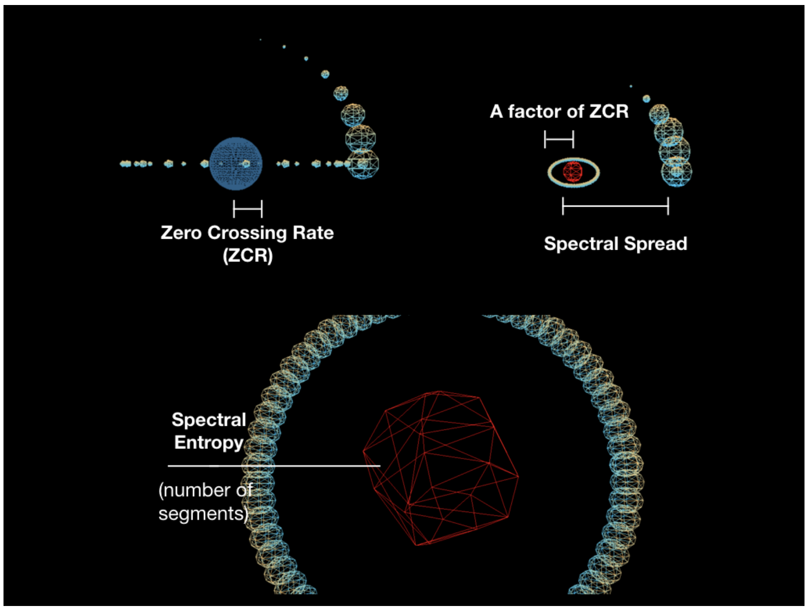

Figure 2: Web application music visualizer parameters

[12]. The features used in genre detection are the mel frequency cepstral coefficients (MFCC) and the spectral centroid. MFCC are a short-time spectral decomposition of an audio signal that convey the general frequency characteristics important to human hearing and are thus often used in speech recognition. MFCC allow the representation of a time domain waveform in a few frequency domain coefficients. MFCC are also widely used in literature for genre classification as it represents short-duration musical textures and thus is assumed to best representative of the music's timbral information [12][9]. Timbre is described as the character or quality of a musical sound or voice as distinct from its pitch and intensity. The other feature, the spectral centroid, is a measure used to characterize the spectrum of an audio signal, defined as the center of gravity of the short term Fourier Transform magnitude spectrum. Spectral centroid has been shown by previous research to be strongly correlated with timbral brightness [8].

For the classification of the music's genre, the K Nearest Neighbor (KNN) algorithm was selected due to existing evidence for its ease of implementation and use for classification. For KNN, new unclassified data is compared with training data. The Euclidean distance, calculated as the square root of the sum of the squared differences between a new point x and an existing point x_i across all input attributes j, is calculated between the training data and the new data. The algorithm then searches for its k nearest neighbors based on these distance measures from the training samples. Running the algorithm yielded the confusion matrix in Table 2 for the 6 genres: classical, electrical, jazz, rock, rap and blues. The classifier tends to confuse rock with electrical music, which is intuitive as rock music has typically centered on the electric guitar and other electronic instruments. Similarly, electrical music is confused with rap music, which may also include electronic components. Performance for the classical, jazz, and rap genres is mostly accurate with fewer relative mis-classifications.

Despite the existence of many music genres, three genres (classical, electrical and jazz) were chosen to correspond with the visualization parameters which are the RGB values as seen in Figure 3. The KNN algorithm returns a probability for each genre, ie. 30% classical, %20 electrical and %50 jazz. The probabilities are multiplied by 255, the maximum value in the range for each individual color, resulting in a value between 0 and 255. Each of the three

	Classical	Electrical	Jazz	Rock	Rap	Blues
Classical	27	0	2	0	0	5
Electrical	0	21	1	9	1	3
Jazz	3	1	19	2	0	5
Rock	0	0	1	16	3	0
Rap	0	8	1	3	26	0
Blues	0	0	4	0	0	17

Table 2: KNN genre classification confusion matrix

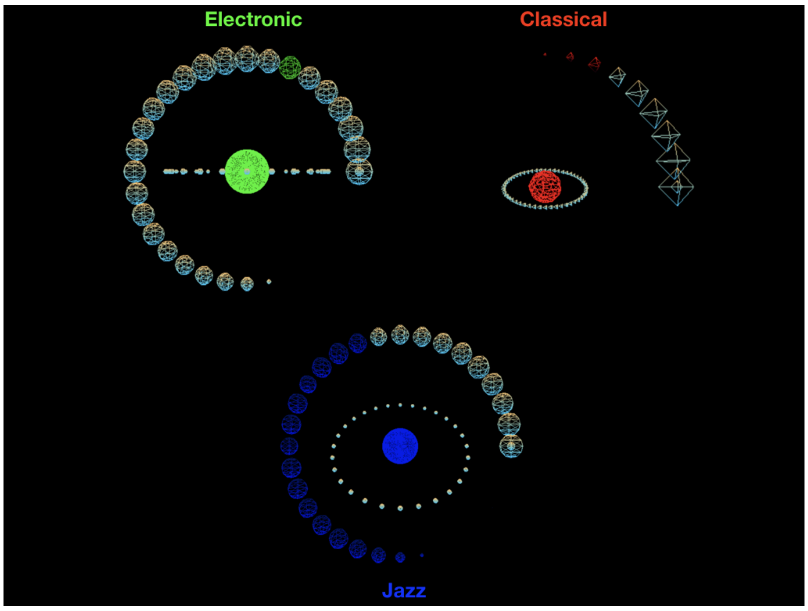

Figure 3: Genre classification depicted through color for three different visualizations

resulting values represents the R, G, and B values respectively. Thus the amount of redness in the visualization is dependent on how classical the music is classified as, the green on electrical and the blue on jazz. Because genre itself is ambiguous and songs can be multiple genres, the selection of three genres with which to classify music is more a decision of convenience than one that introduces error. Furthermore, with a greater number of genres (classical, electrical, jazz, rock, pop and blues), the confidence in a single genre decreases significantly, which would make the color of the visualization less indicative and informative of the perceived genre.

Instrument Detection

With audio segmentation, which describes splitting an uninterrupted audio signal into homogeneous segments, segments can be classified separately using a model. Audio segmentation serves to find changing points in the content of an audio stream. For example, audio segmentation is used in speaker diarization where it attempts to determine “who spoke when”. This process utilizes unsupervised identification of a speaker within an audio stream and the intervals during which each speaker is active. Applications of speaker diarization include speech and speaker indexing, speaker recognition in the context of multiple speakers, and speech-to-text transcription [17]. In this project, the ideas behind identifying differences in an audio signal through audio segmentation are used for instrument detection. To draw a parallel from instrument detection to speaker diarization, each instrument represents a speaker. In this process, audio segments are first clustered into instrument-specific clusters. Then, unsupervised learning is leveraged in order to discern the different instruments based on the sequential structure of audio signals. The features used to represent each instrument are critical to the effectiveness of the algorithm. There are two main steps:

1. First, features are extracted from the audio. In this case, the features are exclusively the MFCC (Mel-frequency Cepstral Coefficients). MFCC represent the short term power spectrum of a sound, and are often used in speech recognition as well as music information retrieval.

2. K-means clustering is an unsupervised algorithm used to extract labels from unlabeled data, where K, the number of groups or clusters is not specified. The algorithm iteratively assigns each data point to one of the K groups dependent on feature similarity of the data point to the other group. This process results in the centroids of K clusters which are used to label new data. Each centroid of its respective cluster defines the feature values that define the group. A benefit of K-means clustering is that it allows the algorithm to determine organically formed groups rather than require defined groups before analyzing the data. The silhouette width criterion is used to determine the optimal value of K, which in this case is the optimal number of instruments or speakers. Silhouette analysis examines the separation distance between clusters and its plot shows how close each point in one cluster is to points in the neighboring clusters. Coefficients near 1 indicate the sample is far from the neighboring cluster; coefficients near 0 indicate the sample is on or very close to the decision boundary of a cluster; coefficients near -1 indicate the sample may have been assigned to the wrong cluster. A high silhouette coefficient value indicates that the object is well matched to its own cluster. Figure 4 shows two graphs for a piece of music. The top graph shows 4 "speakers", where a speaker represents an instrument. The bottom graph shows a plot of the silhouette width criterion, where the optimal number of clusters is determined to be 4.

In the web application visualization, instrument detection is depicted by spawning a spherical geometry into orbit from the center when the new instrument is detected. When a new instrument is detected, the old instrument is visualized to shatter into small pieces and disappear as shown in Figure 5. The green sphere that is being spawned from the center represents the change detected from the audio segmentation and K-means clustering process–that is, the newly detected instrument.

Augmented Reality Visualization

The second part of this project looks to bring music visualization to a relatively new, accessible platform, augmented reality. The combination of Augmented Reality (AR) technology and audio presents a way to enhance the interaction experience between a user and music. In 2017, Apple released ARKit, an augmented reality framework that enables developers to build AR applications accessible to those who own an iPhone SE or iPhone 6s and up. ARKit combines information from the iOS device's motion sensing hardware with computer vision analysis of the scene visible to the device's camera. iOS is one of the most popular mobile operating systems and the native IDE, Xcode provides support to call C or C++ wrapper library functions and has C APIs for vector and matrix math, digital signal processing, large number handling, and image processing. These factors make iOS an easily accessible mobile platform with performance and computational benefits.

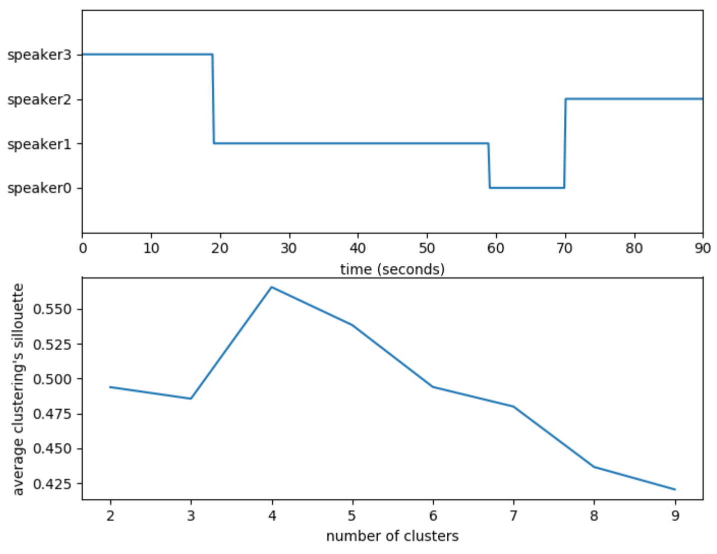

Figure 4: Instrument detection and silhouette plot

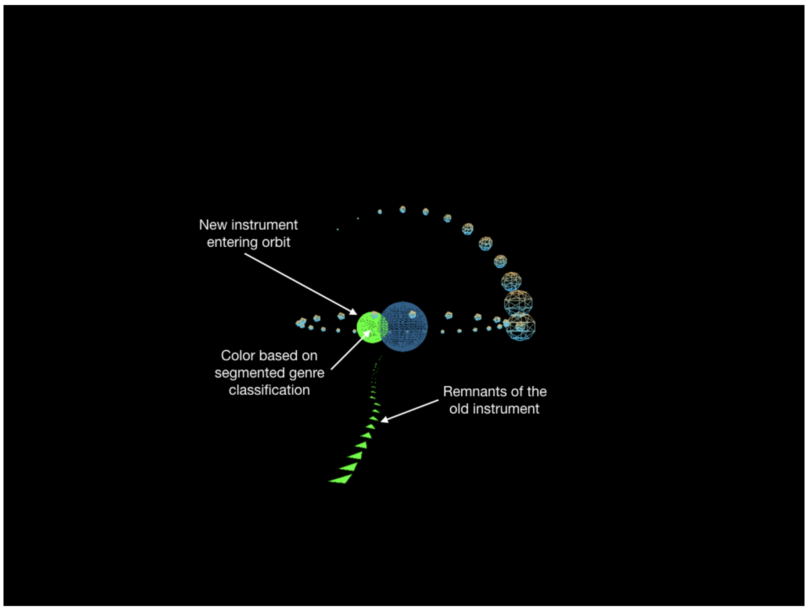
New instrument entering orbit
Color based on segmented genre classification
Remnants of the old instrument

Figure 5: Instrument detection visualization

As part of the interoperability between Apple's newer language Swift and Apple's legacy language, Objective C, Swift has the ability to call C APIs (as well as Objective C). Swift is also compatible with a number of common C constructs, language types and patterns. Therefore, Swift can make optimizations in algorithm implementations by calling lower level APIs.

Several additional factors have to be taken into account for development in an iOS environment with code written in Swift. With the advantage of Apple hardware specifically built for the software in iOS, the Fast Fourier Transform can be implemented to exploit this. Apple's Accelerate framework provides the low level interface to do computationally heavy data processing techniques and makes large-scale mathematical computations and image calculations optimized for high performance. The FFT is computed using the following process:

1. Perform Hann window to reduce frequency leakage
2. Buffer transformations.
3. Pack input into complex buffer
4. Perform FFT
5. Normalization

Human Activity Detection

The purpose of human activity detection is to enhance the augmented reality visualization to change based on human movement in addition to the audio signal. The data from the built-in iPhone sensors can be accessed using Apple's CoreMotion framework. CoreMotion allows developer access to acceleration data, gravity data and rotation data in the x, y and z directions which are shown in Figure 6. Existing research has used the accelerometer data to classify human activity by training a feed forward neural network [5]. With UCI's HAR (Human Activity Recognition) dataset, several research groups have trained their own classifiers to identify human activity. The dataset contains 6 labels: walking, walking upstairs, walking downstairs, sitting, standing and laying. The provided features are the accelerometer data and gyroscope data for the x, y and z axes. A convolutional neural network is trained for the visualization.

The activity detection is visualized in the mobile app by the overall shape of the surrounding spheres. When stationary, the surrounding spheres more closely surround the user while walking and running are increasingly less so.

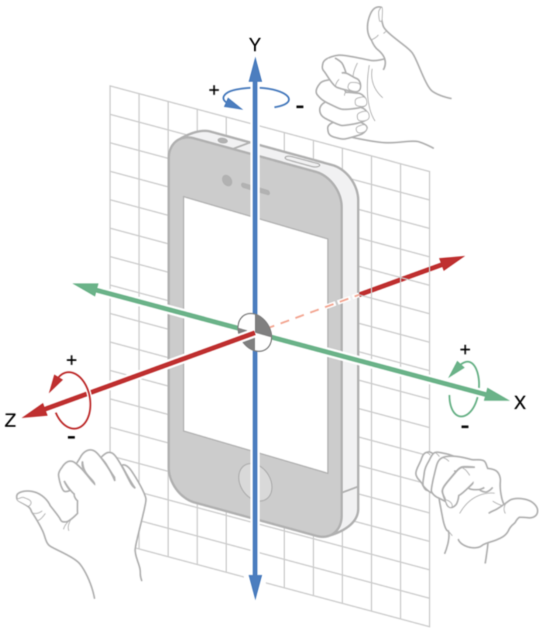

Figure 6: CoreMotion coordinate plane with respect to iPhone

Implementation

Web Application

The web application was built using the Flask web microframework with a Python backend and a webGL based frontend. Flask is a microframework for Python, with its defining features being its simplicity and extensibility. By default, Flask leaves many implementation decisions up to the developer. For example, there is no set database, templating engine, or file directory structure that a developer must use. Furthermore, with the vast number of Python signal and data processing libraries such as scipy, sklearn, numpy and librosa, using a Python based framework makes integration of the algorithmic and data processing code with the web application code fairly seamless.

The user interface of the web application is straightforward: it includes a button to upload an audio file and then presents controls to play or pause the audio. The classification for the audio signal is done after the user uploads the file and before the visualization is displayed. After the audio file is processed and the classifiers are run on the file, the visualization can be played. In the animated visualization, the computation of the FFT and time domain amplitude are also computed on the client side in real time using JavaScript's native Audio API. Even though the FFT is computed on the server side, to stream the server side data to the user client side so that it can be displayed in real time along with the music being played is complex and impacts performance. Thus, the client side JavaScript Audio API is used for visualizing the FFT and time domain amplitude components in real time as the audio is played. A video demonstration is here [1]. The code is here [2].

Performance

Depending on the length of the audio file, the web application could time out after the user uploads a file due to the computational expense of extracting the necessary audio features. This timeout typically happened with files greater than three minutes. Loading and computation time become an issue as the deployment service, Heroku, does not allow for requests greater than 30 seconds. If the preprocessing takes longer than 30 seconds, the application will break. To circumvent this timing issue, Redis Queue is used to start a background worker. Redis Queue is a Python library for queuing jobs and processing them in the background with workers. It is backed by Redis, which is an in memory database that persists on disk. A worker is a Python process that typically runs in the background and exists solely to perform lengthy or blocking tasks that should avoid being performed inside web processes. A function can be invoked asynchronously in the worker process by pushing a reference to the function and its arguments in a queue, defined as enqueueing. When the function is enqueued, a proxy object is returned to check the status of the function being performed. When the function is completed, the object will indicate that it is done and return the desired results. On the client side of the application, a JavaScript function polls the backend at regular intervals (in this case, every 7 seconds) until the data is ready and the application is ready to return. Thus, the timeout after 30 seconds is resolved.

Mobile Application

Unlike when on a computer where many different programs can be open and running at once, when a user has one mobile app up, the GPU and CPU work to power that app for the user. A

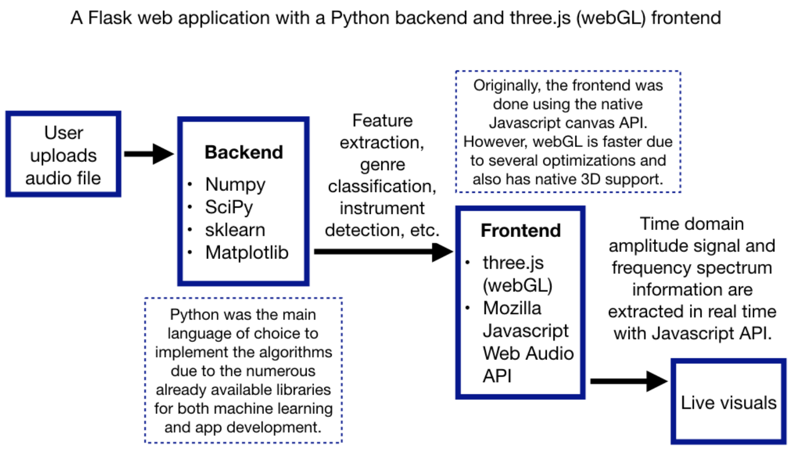
A Flask web application with a Python backend and three.js (webGL) frontend
User uploads audio file
Backend
• Numpy
• SciPy
• sklearn
• Matplotlib
Feature extraction, genre classification, instrument detection, etc.
Originally, the frontend was done using the native Javascript canvas API. However, webGL is faster due to several optimizations and also has native 3D support.
Frontend
• three.js (webGL)
• Mozilla Javascript Web Audio API
Time domain amplitude signal and frequency spectrum information are extracted in real time with Javascript API.
Python was the main language of choice to implement the algorithms due to the numerous already available libraries for both machine learning and app development.
Live visuals

Figure 7: Web Implementation

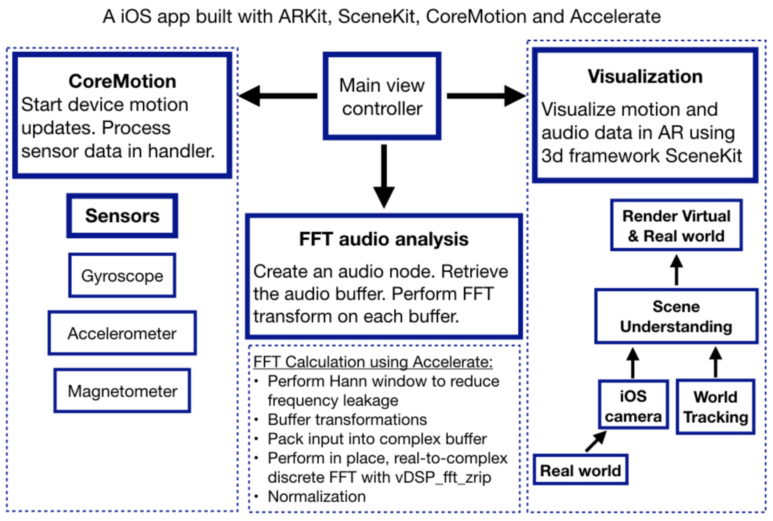
A iOS app built with ARKit, SceneKit, CoreMotion and Accelerate
CoreMotion
Start device motion updates. Process sensor data in handler.
Main view controller
Visualization
Visualize motion and audio data in AR using 3d framework SceneKit
Sensors
Gyroscope
Accelerometer
Magnetometer
FFT audio analysis
Create an audio node. Retrieve the audio buffer. Perform FFT transform on each buffer.
FFT Calculation using Accelerate:
• Perform Hann window to reduce frequency leakage
• Buffer transformations
• Pack input into complex buffer
• Perform in place, real-to-complex discrete FFT with vDSP_fft_zrip
• Normalization
Render Virtual & Real world
Scene Understanding
iOS camera
World Tracking
Real world

Figure 8: Mobile Implementation

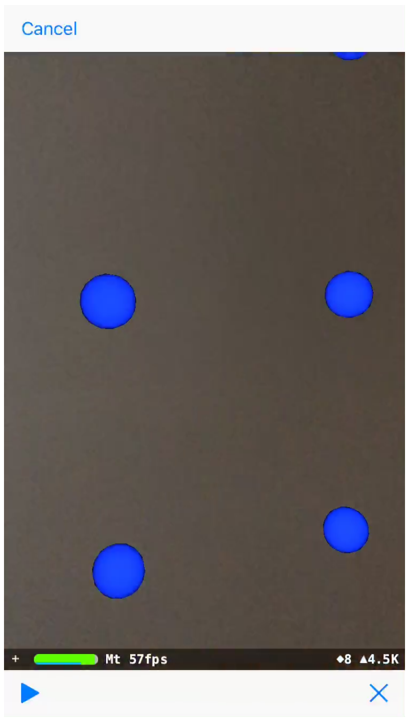

Figure 9: iPhone screenshot of visualization in augmented reality

mobile device's nature is generally meant to run one thing at a time. As a result, not many parallel optimizations can be made for the mobile application. Instead, Apple's native frameworks that are optimized in performance by the hardware are used. The graphics are rendered in augmented reality using Apple's frameworks ARKit and SceneKit. With the concern of rendering 3D objects using Apple's graphics framework SceneKit, retrieving motion data in real time and regular intervals, and processing the audio signal, performance and optimizations become especially important while on a mobile device. A video demonstration is here [3]. The code is here [4].

ARKit

ARKit is Apple's augmented reality framework that enables developers to build iOS applications with AR experiences. Its core function is to track the relation between the real world space of the user and the virtual space of the modeled visual content. As a result, when the app displays visual content in the live camera image, the user gets the illusion that this virtual content is a part of reality. To do so, ARKit uses a function called world tracking which is powered by a technique called visual-inertial odometry. Visual odometry entails estimating the 3D pose (translation and orientation) of a moving camera relative to its starting position, using visual features. This technique entails combining information from the iOS device's motion sensing hardware with computer vision analysis of the scene visible to the device's camera [15].

There are several components in an AR system as depicted under the visualization block in Figure 8:

1. Real world capture: The iOS camera is used to capture the background which is simply reality.

2. Virtual world: Virtual objects are created through code and drawn by the GPU. iOS supports a number of engines for virtual world creation, namely: Apple 3D framework

SceneKit, Apple 2D framework SpriteKit, Apple GPU accelerated 3d graphics engine Metal, OpenGl, Unity3D and Unreal engine.

3. Combining the virtual world and real world through rendering.

4. World Tracking: This process entails tracking the changes in the real world (ie. moving the camera) and moving the virtual object accordingly.

5. Scene Understanding: The AR system detects when virtual objects are placed on a real plane.

6. Hit Testing: The ability to modify, move and zoom into or out of virtual objects.

The main classes of the ARKit API are `ARSession`, `ARConfiguration`, `ARFrame` and `ARCamera`. `ARSession` manages the device camera and motion processing as well as does image processing, establishing the relationship between real world space and the virtual world created in AR. `ARConfiguration` is an abstract class that subsumes tracking configurations that define how the device and movement should be tracked. For example, `ARWorldTrackingConfiguration` tracks the precise device position and orientation and allows for plane detection and hit testing while `ARFaceTrackingConfiguration` tracks the movement and expressions of a user's face. `ARFrame` is a video image containing tracking information that is capture as part of an AR session. `ARCamera` contains information on the camera position for an `ARFrame` in a session.

SceneKit

SceneKit is a 3D graphics framework and allows the creation of nodes (`SCNNode`) which describe a single physical entity such as a sphere or a cube. To animate these nodes, the nodes' properties such as radius or other geometric parameters must be updated. To prevent stalling and lags in the graphics, creating too many nodes must be avoided. In the AR visualization, there are spherical geometries surrounding the user in a ring like form, with each sphere representing a different frequency in the frequency spectrum and changing size based on the FFT. Each ring that surrounds a user contains 32 spheres for a frame count of 64 for the FFT computation. So, to have 4 rings surrounding the user, $32 * 4 = 128$ nodes would have to be rendered and updated regularly for animation. Through trial and error, it was determined that 224 (7 rings) was the maximum amount of nodes where there was no noticeable lag in the graphics updating and rendering. To further reduce graphics rendering, spheres out of the user's line sight are not updated or animated. A screenshot of the spheres is shown in Figure 9.

Conclusion and Future Work

In this work, two music visualizations are created: a web based visualization uses machine learning to extract meaningful data from the audio signal and a mobile application that enables music and motion visualization in a more interactive platform, augmented reality. There are several avenues of application for this type of visualization. Both music visualizers enable people to enjoy music through different senses, particularly beneficial to the auditory impaired. The augmented reality application can also be used to motivate exercise through gamification similar to Pokemon Go.

Several performance optimizations can still be made in regard to the web based visualization.

For example, to minimize the loading and processing time a user must wait after uploading an audio file, a small constant-size representative chunk of the music can be analyzed rather than the entire music file, thus making the processing time independent of the length of the file. This small segment of an entire song that is most representative of the music is described as a thumbnail. The goal of most music thumbnail algorithms is to determine the most common parts of the song and extract a thumbnail. Existing research uses a fitness measure, defined as how well a given segment explains other related segments and how much an overall music recording is covered by all the related segments. The algorithm often consists of extracting short-term audio features, computing the self-similarity matrix, i.e. all pairwise similarities between feature vectors, and finding the position of the maximum value of the new (filtered) self-similarity matrix which represents the most "similar" segment of music (often the most repeated parts such as the chorus) [16]. With a thumbnail, the genre classification and instrument detection and potential future work would need only rely on a smaller amount of information and would improve reliability of the web application.

The style and type of the augmented reality visualizations are currently not flexible, leaving room for user customization in the future. Allowing user choice in what parameters to visualize as well as how to visualize them would make the application more practical for entertainment uses as well as clarification of how parameters are affecting the visuals. Performance may also be able to be improved through more clever appropriation of resources for processing and calculating data in real time. Therefore, future work will involve enhancing visualizations to production standard and also improve on existing algorithms for real time data processing.

Bibliography

[1] Web application demonstration. https://youtu.be/PmZVyJr4Zqk

[2] https://github.com/Lucytheanimefan/music-drawer-browser

[3] Mobile application demonstration. https://youtu.be/YvtPtDnEdhs

[4] https://github.com/Lucytheanimefan/MotionAR

[5] Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2012, December). Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In International workshop on ambient assisted living (pp. 216-223). Springer, Berlin, Heidelberg.

[6] Bendich, P. L., Gasparovic, E., Harer, J., & Tralie, C. (2016). Geometric Models for Musical Audio Data. In Proceedings of the 32st International Symposium on Computational Geometry (SOCG).

[7] Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., & Ruiz, O. (2011, June). Interactive visualization of volumetric data with webgl in real-time. In Proceedings of the 16th International Conference on 3D Web Technology (pp. 137-146). ACM.

[8] D. L. Wessel, "Timbre Space as A Musical Control Structure," Computer music journal, pp. 45–52, 1979.

[9] D. Ellis. Classifying music audio with timbral and chroma features. In Int. Symp. on Music Information Retrieval (ISMIR), pages 339–340, 2007.

[10] Foote, Jonathan. 1999. Visualizing music and audio using self-similarity. In Proceedings of the seventh ACM international conference on Multimedia (Part 1) (MULTIMEDIA '99). ACM, New York, NY, USA, 77-80. DOI=http://dx.doi.org/10.1145/319463.319472

[11] Fourney, D. W., & Fels, D. I. (2009, September). Creating access to music through visualization. In Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference (pp. 939-944). IEEE.

[12] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. IEEE Transactions on speech and audio processing, 10(5):293–302, 2002.

[13] K. C. Ng, "Music via motion: transdomain mapping of motion and sound for interactive performances," in Proceedings of the IEEE, vol. 92, no. 4, pp. 645-655, April 2004. doi: 10.1109/JPROC.2004.825885

[14] Kowaliw, T., Dorin, A., & McCormack, J. (2009, December). An empirical exploration of a definition of creative novelty for generative art. In Australian Conference on Artificial Life (pp. 1-10). Springer, Berlin, Heidelberg.

[15] P. Li, T. Qin, B. Hu, F. Zhu and S. Shen, "Monocular Visual-Inertial State Estimation for Mobile Augmented Reality," 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), Nantes, 2017, pp. 11-21.

[16] Müller, Meinard. (2015). Fundamentals of Music Processing – Audio, Analysis, Algorithms, Applications.

[17] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland and O. Vinyals, "Speaker Diarization: A Review of Recent Research," in IEEE Transactions on Audio, Speech, and Language Processing, vol. 20, no. 2, pp. 356-370, Feb. 2012. doi: 10.1109/TASL.2011.2125954