Music Visualizer for PlayStation Vita

Interim Report

Submitted for the BSc in Computer Science with Games Development

January 2015

By

Adam Connor Lutton

Table of Contents

Introduction

This project is intended to produce a music player for the PlayStation Vita mobile games console. Part of this software is “Music Visualization”, a term describing the use of graphical effects displayed on screen that are manipulated or distorted either in time with the beat of the audio, or just as a general accompaniment of the audio. Ideally, the software should also pick visualizations based on the genre of music.

This document will detail more in-detail research into the processes, tools, and techniques the project is using, such as Fast-Fourier Transformations and Shaders. Furthermore, program design, an evaluation of progress so far, remaining tasks, and additional topics; will also be discussed in detail. The background and project management topics are of particular interest, as it explains the fundamentals of the specific aspects of the project and the results of trying to implement some of them or the reasoning as to why they were not implemented.

Design aspects of the project have changed drastically since the initial proposal, as has methodology. The heavy implication of signal processing is no longer a primary objective, instead the project is now focused on user features and visual effects.

During the remainder of this project, FFT implementation will be added to the currently functional prototype. The current version of the software will be improved to allow superior music playback, functionality, and improved ease of use by fine-tuning the user interface design. That being said, it is an on-going build and certain features have not been implemented yet, particularly graphics. The designs in this document currently reflect working designs and are not final.

Aim and Objectives

The purpose of this project is to create a music visualizer for the PlayStation Vita system, which provides an accurate visual response to an audio input.

This will be obtained by completing a number objectives that deal with multiple aspects of the project. They are listed as such:

Objective 1 – Create software that loads and plays the user's audio files

Simply put, the software is a music player, therefore it needs to meet the requirements that other music players on the market already meet. Specifically, the ability to load in audio files and then allow the user to pause, play the audio, as well as skip to the next or previous songs. Doing this in PlayStation Mobile is a relatively easy task compared to OpenTK, so this aspect will be developed using that tool-set. In terms of evaluation, this particular aspect should “Just work”, so failure to operate as intended is a large flaw in the program.

Objective 2 – Create multiple visualizers to accompany the audio through the use of shaders

The software is expected to allow the user to switch between several visual designs. These shaders will be created and imported into the PlayStation Mobile project. An additional programs such as AMD RenderMonkey, ShaderToy, Shader Tool, and GLSL Hacker will be used to create them. The reasoning for this is to allow easier manipulation of the code with faster visual representation, it also removes the possibility of other parts of the program causing the shaders to not be displayed, which would become an annoyance during development. The visualizers will be evaluated based on their visual quality and effectiveness of audio representation where applicable.

Objective 3 – Implement FFT to improve and create new graphical effects

To obtain the most accurate representation, FFT needs to be used to obtain specific audio data of a track as it plays. Specifically, the frequency of a given audio signal over a given time. Using this, it would be relatively simple to pass in the frequency into the vertex or fragment shader to generate an effect. For example, an animated waveform pattern. This will be evaluated in a similar fashion to the objective above.

Objective 4 – Research and test how to improve the accuracy of the visualization in relation to the audio track

Additional research to improve the synchronization of the audio and graphical effects could be done. This would probably be in relation to audio recognition software, such as voice recognition for example.

Objective 5 – Use audio data to change visualizations

Based on the previous objectives success, if implemented, it could be possible to differentiate between different music genres, know the difference between a music track and a vocal track (podcast), and even determine different instruments within the track. This will be evaluated in a similar fashion to objectives 2 & 3.

Background

There are a variety of aspects that this project revolves around. The most notable and important aspects have to do with audio signal representation and graphical visualization. Furthermore, the method by which the project is being developed.

Music and Audio Signals

Sound is consists of several different attributes, the four main ones being; pitch, loudness, duration, and timbre.

Pitch is defined as a frequency related scale that is designated from low to high. Specifically, it is defined as the frequency of a sine wave that is matched to the target sound that human listeners would expect. In relation to this project, pitch is the defining factor for audio representation, as the process of calculating the wave form heavily relies on the initial frequency of that sound at a given point. (Klapuri & Davy, 2007)

To get a good idea of how sound is defined as a waveform, using an Oscilloscope, it is possible to convert sound into voltages and displayed. The image displayed sound give an accurate waveform formation of the constantly changing formation of audio pitch through the duration of a track. This comes from personal experience of using this tool.

Loudness is defined by its physical properties within audio space. In terms of audio signals however, we tend to apply logarithmic scale based on its magnitude, known as the decibel scale. The magnitude is the second most important aspect to consider when it comes to music visualizations. Quieter and louder parts of an audio track need to be distinctly different from each other (Not required for visual accompaniment, this point will be discussed later), otherwise the representation will be considered far from accurate. (Klapuri & Davy, 2007)

In terms of its calculation, within real-world space it is denoted as:

$$\mathbf{I} = p\mathbf{v}$$

Figure 1 - Equation for calculating the intensity of a sound. See Appendix for image source.

"$\mathbf{I}$" equates to the sounds intensity (Magnitude). "p" defines the sound pressure and "$\mathbf{v}$" defines the particle velocity. Both $\mathbf{I}$ and $\mathbf{v}$ are defined as vectors.

The sound intensity over a given time (*T*) is given by:

$$\langle \mathbf{I} \rangle = \frac{1}{T} \int_0^T p(t)\mathbf{v}(t)\,\mathrm{d}t.$$

Figure 2 - Sound intensity over a given time. See Appendix for image source.

Here, it shows the change in pressure and velocity in relation to time. The amount of time is noted from zero to the maximum duration.

(Wikipedia, n.d.)

However, in relation to signal processing the simplest way to explain the differences in magnitude is to use Sine Waveform graphs.

The volume of a sound would be represented on the graph as amplitude.

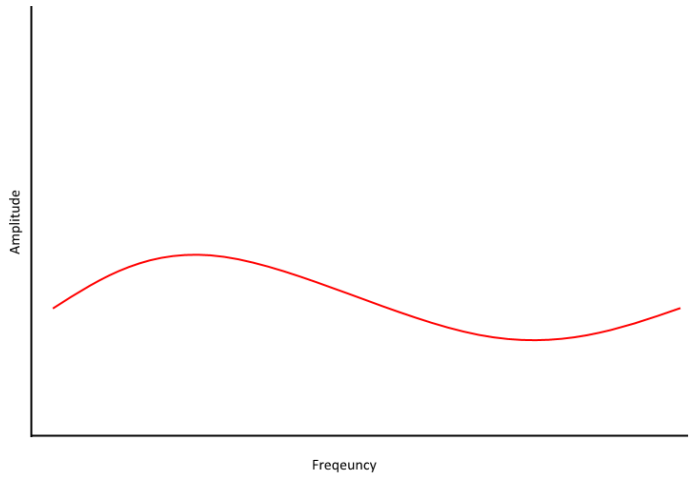


Figure 3 - Low amp graph representing low volume

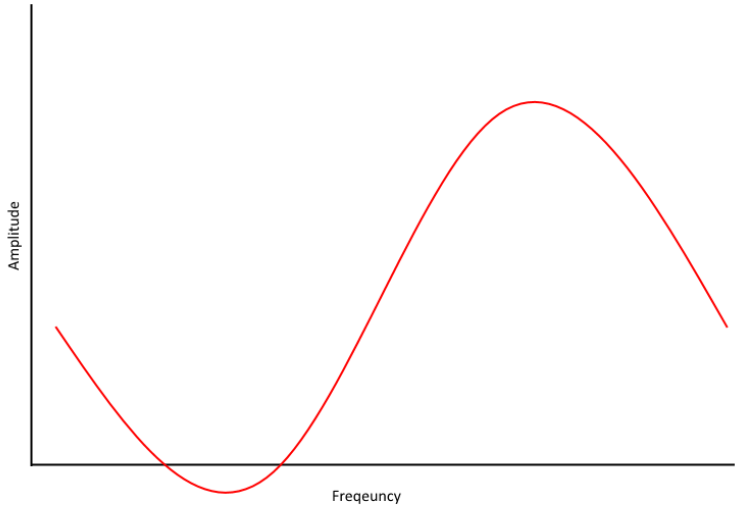


Figure 4 - High amp graph representing high volume.

The first graph shows a shallower curve indicating that the amplitude (Volume) is relatively lower compared to the second showing a steeper curve. As an additional note, for graphs like this, the closer the curves are the higher the magnitude, which is calculated based on the peak-to-peak amplitude.

Timbre is an aspect of sound that specific relates to music. It is often referred to as a sound's "Colour" and relates to how we recognise sound sources. Specifically it relates to how you can have two different instruments play at the same pitch and volume, but are still distinguished by their timbre. (Klapuri & Davy, 2007)

It relies heavily on the fundamental frequency, which is the lowest frequency, one of which that also defines the note of the sound within musical terms. Harmonics is the richness of the sound, which is described as the sum of the number of distinct frequencies. On top of that is the "Envelope" which can help recognise a sound source based on the Attack time, Decay

time, Sustain level, and Release time (ADSR Envelope) (Dodge & Jerse, 1997). Combined, Harmonics and Envelope make up a large chunk of how timbre is processed.

Fourier Transformations

A Fourier transformation breaks down a signal into the frequencies that construct it. There are various forms of the algorithm for different purposes, including discrete or continuous calculations. Specifically for this project, Fast Fourier Transform (FFT) is used to compute a discrete Fourier Transform and its inverse. The algorithm converts time to frequency, however in a much faster method. It factorizes the DFT matrix into a product of sparse (A matrix containing mostly zeroes) factors.

Here is the algorithms for calculating frequency and time:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i 2\pi k n / N}$$

Figure 5 - DFT Algorithm where Xk is equal to frequency.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i 2\pi k n / N}$$

Figure 6 - DFT Algorithm where Xn is equal to Time. Image sources in Appendix.

Xk represents the amount of the Frequency **k** in the signal.
xn represents the value of the signal at time n.
n is the current sample being used.
k is the current frequency.
N dictates the total number of samples with have.

The first figure calculates the amount of times a particular frequency has occurred over a number of samples on a given time spike. The second figure describes all the frequencies that have occurred at a given point in time. (Better Explained, 2012)

For the project, this equation should provide all the frequencies at a given time within an audio track which can then be in turn used to plot graphical data such as vertices, or used to change the colour, and so on. Due to the complexity of the equation however, implementation will be time consuming.

Shaders

These are a program that tells the graphics render to draw an object in a particular way, whether it be a specific shape or colour. Two specific shader types exist, vertex and fragment shaders. Vertex describes the positioning of an object, texture coordinates, and so on. The fragment shader describes the colour, texture data, alpha values, and so on.

In regards to this project, shaders will be used to create unique graphical effects that will be used for the visual accompaniment to the music being played.

Visual Accompaniment VS Visual Representation

During the course of development, considerations towards how the graphic visualization should be implement lead to the thought process and research of visuals that did not react to signal processing at all. What is meant by this is that the on-screen graphics are deliberately not synced up or related to the audio track playing in anyway.

Examples of this exist, the PlayStation 3 has multiple visualizers which only display images that do not relate to the beat or pitch of the music at all. For example, the visualizer that is just a 3D model of the Earth with the camera rotating around it.

Figure 7 - PlayStation 3's Earth Visualizer. Image Source in Appendix.

This led to a design decision that multiple visualizers should exist within the program. Some would accurately represent the music through its graphics by processing the signals, and others would just draw random visuals that are completely separated from signal processing.

An advantage to developing these visualizations is that they can be developed relatively quickly, as they do not rely on configuring with FFT equations nor do they require testing to see whether the synchronization is correct.

Development Method

The project is using aspects of multiple development methodology. It uses aspects of Adaptive, Iterative, and Prototype development models. Large parts of the program are designed in a general way, then tasks that are to be completed first are quickly reviewed. After that, prototyping is used to build aspects of the program. From there it is refined continuously.

The main program prototype has been redesigned completely twice at this point, reflecting changes to improve efficiency, reuse of code, and streamlining of functions. It is also a reflection of issues that have occurred by using various software toolsets.

Designs

Class Design

For the prototype, two slightly different design patterns were used. The first design reflects the structure that was planned and partially used for the OpenTK prototype, the second is a revised design for PSM.

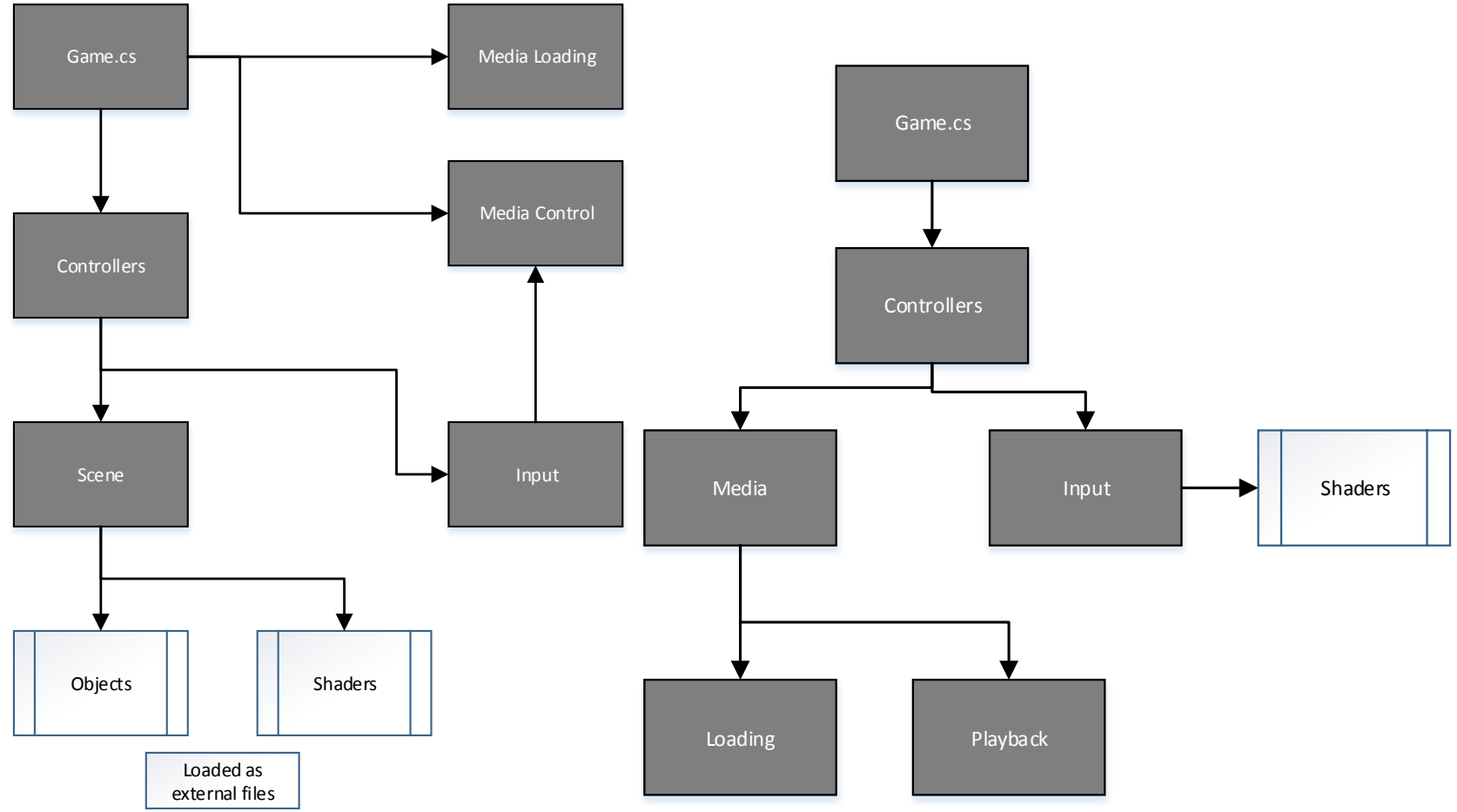


Figure 8- Designs 1 and 2 of the program.

In both designs, the use of Controllers is vital. In left diagram, two controllers exist; one for scene and the other for input. The Scene controller defines what is displayed on screen, and the original design would contain lists of visual objects and shaders to display. The input controller handled button presses but also aspects of the UI that were related to media playback, hence the connection to it. Media loading and playback control were completely separated.

In the second design, aspects were streamlined further. A Media controller was added which handled loading and playback. The scene controller was removed from the design in favour of something more basic. Input now directly controls which shaders to be displayed and sends that information back to the drawing code (Not pictured).

The second design is an improvement from a scalability and efficiency standpoint. Components of the program should be able to communicate with each other without issue, and the drawing code is less complex. It keeps all of the relevant components of the program under one label. Additionally, once FFT is implemented, it should not affect other aspects of the program, in theory.

Media Controller

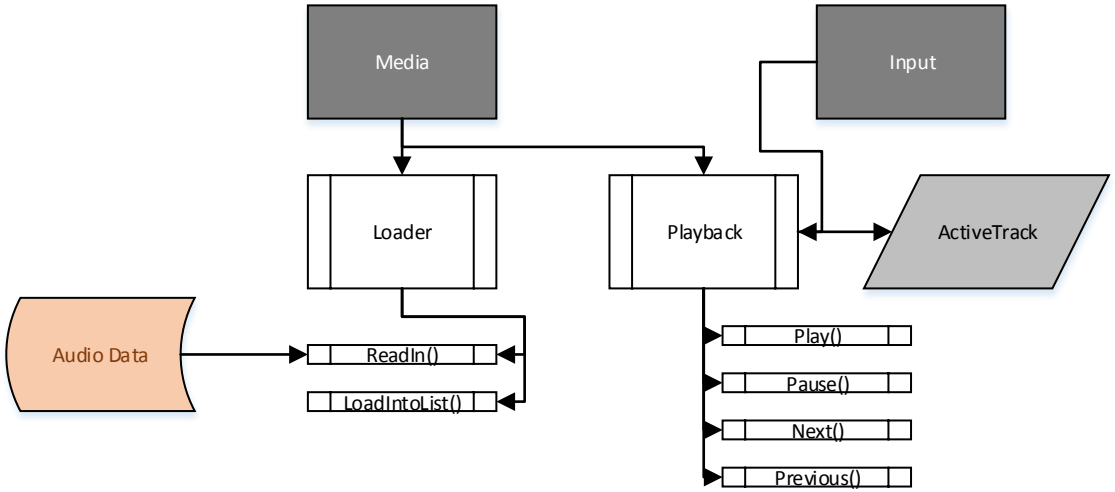


Figure 9 - Media Controller

As mentioned previously, the media controller handling loading and playback. The loader has two main processes, one for reading in the music files contained in the directory folder, and another for compiling that information into a list which the playback features will use. Playback

contains a subclass named “ActiveTrack”, which contains all the information for the current playing audio file; such as name, file directory, file size, duration of audio track, and so on. The process of switching will be described later on.

Shader Controller

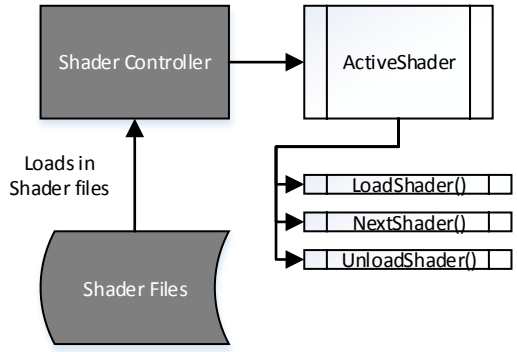


Figure 10 - Shader Controller

The shader controller is quite similar to the media controller in that it has an “ActiveShader” subclass which loads and unloads shader files into the program while it is running. It is also similar in the fact that it loads in and organizes the shader files into a list, making the process of switching between them easier.

Additional design work is expected to be done in this area. Many of the controllers work in similar ways, but often reuse code. Added iterations may streamline the design further through inheritance and the use of interfaces. This will be explored and commented on in the final report.

Switching Tracks

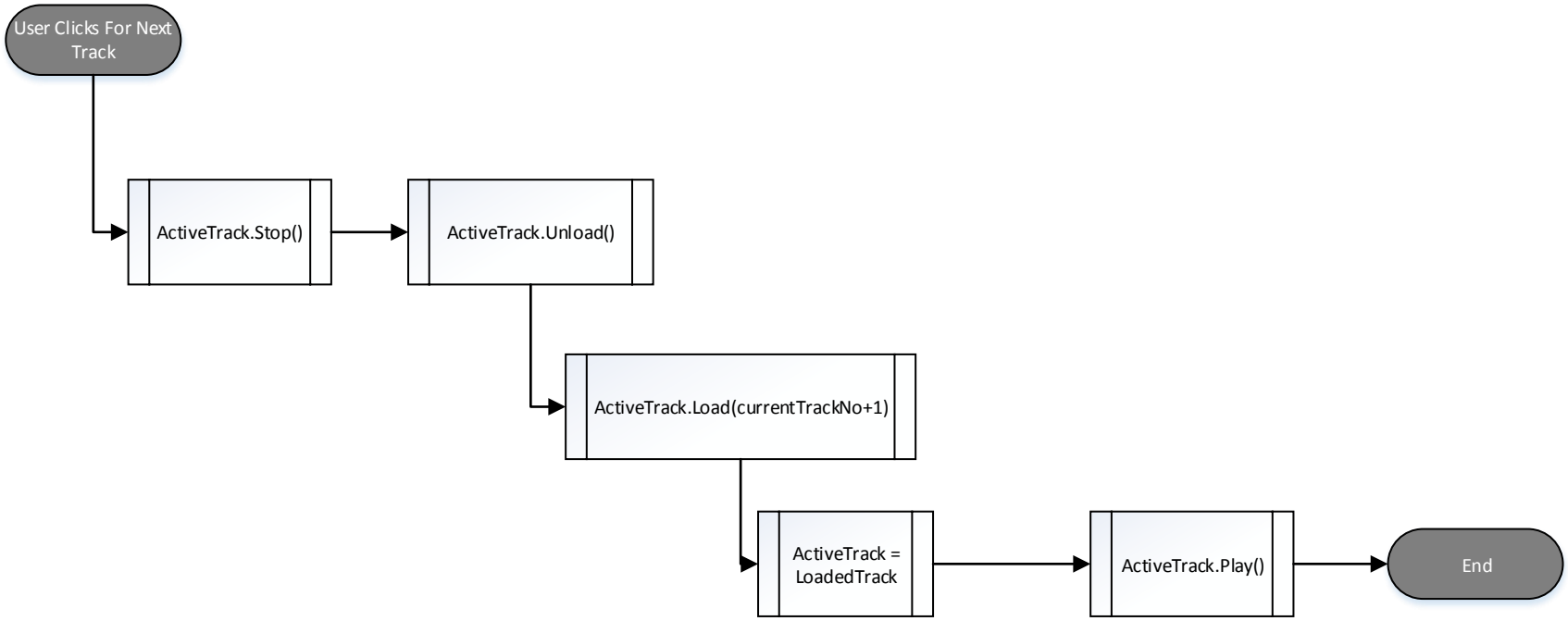


Figure 11 - Loading in audio tracks while the program is playing

Switching audio tracks using “ActiveTrack” works by first stopping the current audio file from playing, then unloading the audio data. Next loads the next audio track by taking in an argument that refers to the index of the next track in the list of tracks already loaded in. It then sets the loaded track as the active track, and plays the new track.

Switching Shaders

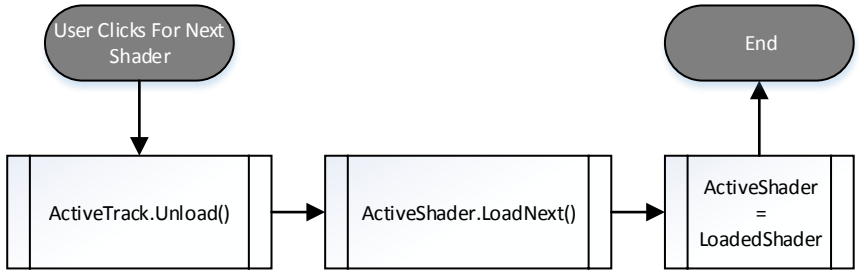


Figure 12 - Shader switching

Shaders are dealt with in a similar way, but with less steps. The shader is unloaded, the next shader is loaded, and then the active shader becomes the loaded shader. Again, very similar to media. However, the possibility of iterating this further is unlikely.

User Interface design

Simplicity is the key to good UI design, as such the following designs attempt to minimize the amount of on-screen elements if possible. Another aspect is the amount of screens the user sees. Ideally, the program should only have two screens; a screen where their music collection is displayed, and a screen with music playback controls.

Music Loading

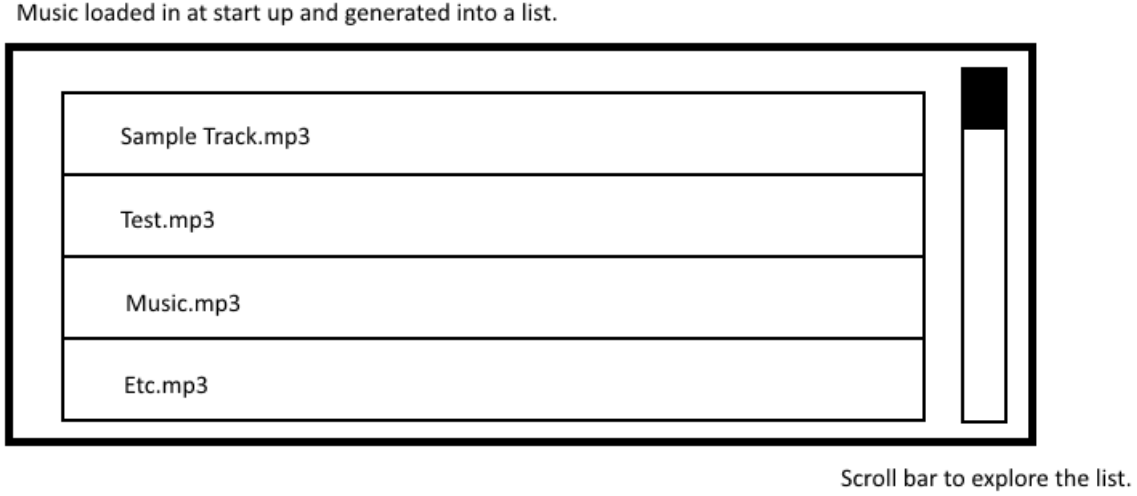


Figure 13 - Music selection menu

At the start of the program after a short loading screen, this screen should appear. The main aspect of it is a loaded list of audio tracks from the user's music collection on their device. They should be able to scroll through it using the scroll bar on the side. Once the user selects a track it should start playing it and take them to the playback screen below.

Audio playback

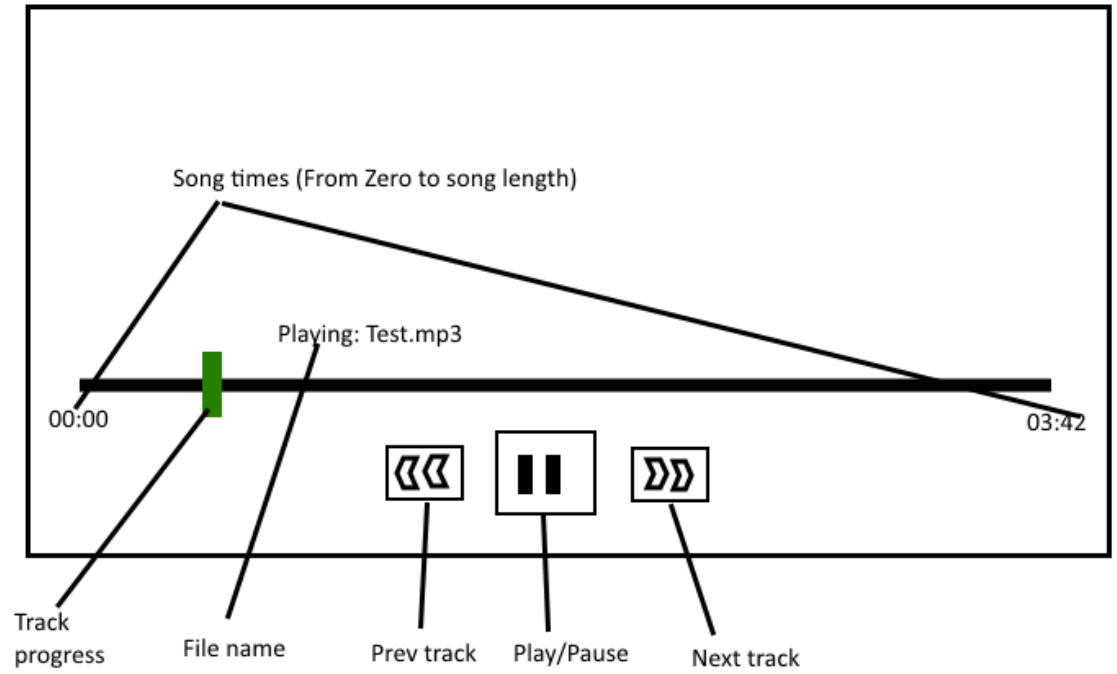


Figure 14 - Music playback controls

The playback controls follow the standard design that you would find on any other media player software; play/pause button, along with next and previous track buttons. Above that is the timeline for the track. These controls appear on-screen then fade away to allow a full view of the visualizer in the background. They appear again when the user taps the screen.

Experimental Design

Without FFT implementation, it makes it quite difficult to theorize tests that could be performed. It is clear however that testing the parity of audio and visuals is important.

The most basic test for this is to have an image flash on the screen in time to an audio signal. Using an audio track which is a continuous 10,000Hz sound wave, setting it up to play at random intervals, and then using Fourier transformation to detect when it is being played. This acts as the most basic test for detecting an audio signal and have it generate a visual response.

From there, more intricate tests can be performed, including waveform patterns, image manipulation, and so on. Unfortunately at this point in time, the use and understanding of FFT is extremely limited. So any and all tests will be written in the final report.

Testing

Additional program testing will take the form of standard User Interface and feature testing, making sure that all parts of the program work as intended. Here is an initial list of tests that will be performed on the first prototype near its completion.

Test	**Description**
Start program	Start the program and have it boot into the first screen.
Music loading in and displaying as a list	The first is a complete list of music in their collection. It should display all songs with the ability to scroll down the list.
User selection	The user selects a track. When the user selects the track it should start playing the file and move on to the next screen with the media controls and visualizer.
Music playing	Music plays. The music should not skip, stop, or any other issues.
Music pause	Music pauses; stops playing until resumed.
Next Track	The next music track is loaded in and begins playing.
Previous Track	The previous music track is loaded in and begins playing.
Music controls fade out	After a few seconds the playback controls should fade of screen to allow a full view of the visualization.
Controls fade in	When the user taps the screen, the controls should appear again.
Shader switching	The visualizations on screen should change.

These tests define whether or not the prototype meets the standard of being feature complete. Even without FFT implementation, meeting all of these test is the bare minimum of being feature complete.

Project Management Review

Task List

The original task list was designed without much insight to how time consuming certain tasks were, or whether or not they could even be completed with the current knowledge I have. As such; many tasks from that original task list either were discarded, are still being worked on, or have yet to be started.

The table below provides the current status of the original tasks in that list.

Task Name	Completed?	Description
Task List	Yes	Completed for Initial Report
Time Management	Yes	Completed for Initial Report
Interim report	Yes	Completed for Initial Report
Risk Assessment	Yes	Completed for Initial Report
Additional initial research	Yes	Completed for Initial Report
Detailed Spec	No	A detailed spec of the design was not created due to time limitations, and because of its limited use. It was cancelled to free up the time for more important tasks.
UML Diagrams	Yes	Created for Interim report
Activity Diagrams	Yes	Created for Interim report
State Diagrams	Yes	Created for Interim report
UI Design	Yes	Created for Interim report
Graphic Design / Visualizer Designs	No	Due to how shaders are implemented, designing them would take time away from actually building them, and the rough ideas would look very different from the finished product. Instead it was opted to build them without previous designs.
FFT Research / Libraries	In Progress	This is currently being worked on.
OpenTK / Shader research	Yes	
Concurrency research	No	This was deemed unnecessary, and overkill for the project. It would only add to the complexity of the program and challenge the already tight schedule.
Hardware Research	Yes	
Basic Visual Response	To Start	FFT not yet implemented.
Prototyping	In Progress	This is currently being worked on.
Prototype Notes	In Progress	This is currently being worked on.
Basic FFT Functionality	To Start	The complexity of FFT forced this task to be delayed to allow the rest of the program to be developed.
Change Notes	To Start	
Class and program structure	Yes	The prototype follows the class structure that is contained in this document.
Beats per minute calculations	No	It is unclear whether this task can actually be completed, as it heavily depends on a variety of factors providing the information necessary for this to be calculated.
Additional debugging and info gathering	To Start	Prototype not in a state to debug.
Graphics / Shader implementation	To Start	Shaders not yet implemented.
More testing with FFT	To Start	FFT not yet implemented.
Concurrency	No	Unneeded.
Functionality Testing	To Start	Prototype not in a state to test.

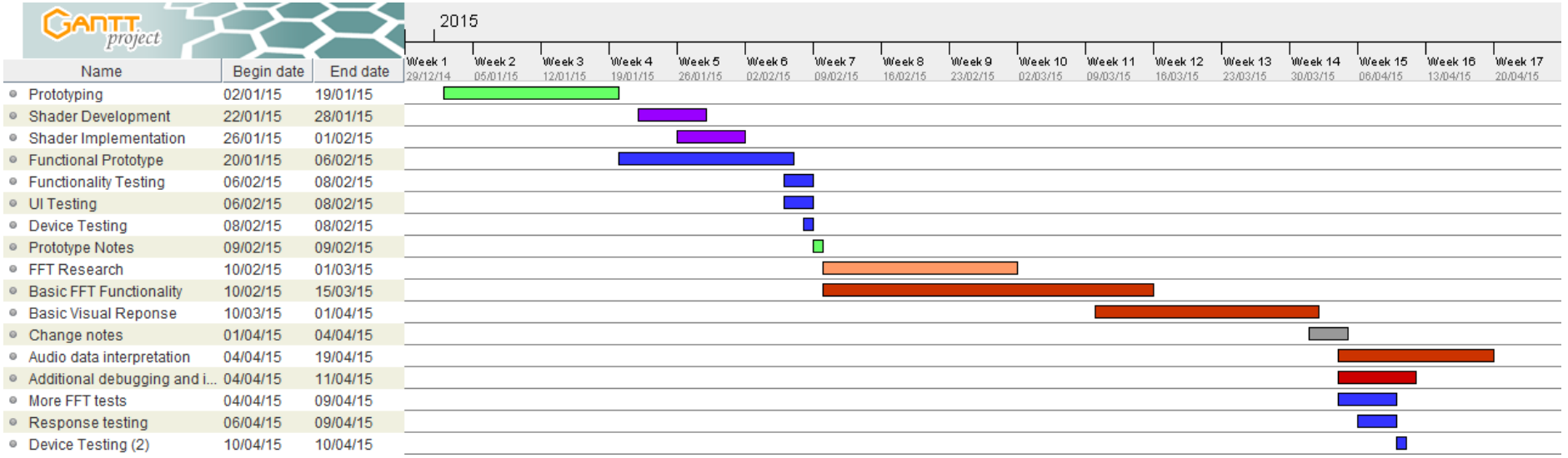
UI Testing	To Start	Prototype not in a state to test.
Response Testing	To Start	Prototype not in a state to test.
Device Testing	To Start	Prototype not in a state to test.

Figure 15 - Comparsion of tasks from original task list to what is currently completed.

When the task list was originally set out, it was noted that this was for the first half of the project. However, as the project has gone on and tasks have not been completed for in time for this “Half-way” deadline. This lead to a restructuring and a reprioritizing of tasks, and eventually cutting away features.

A new task list has been designed to represent these changes as well as a new timeplan (Gantt chart). Note this only covers tasks that have yet to be completed.

#	Task	Description	Duration (Days)
1	Prototyping	Build and test features to see what works and can be added to the program.	18
2	Graphics / Shader development	Develop visualizers using shaders in external tools.	7
3	Graphics / Shader implementation	From the previous task, implement shader work into the program.	7
4	Functional Prototype	Build a prototype that is “Functionally complete” from a usability perspective. (See below for details)	18
5	Functionality Testing	Test features present in the prototype, polish and refine so that they work to the fullest extent.	3
6	UI Testing	Test the UI to see if it works and fix any possible bugs that may occur.	3
7	Device Testing	Test on the PlayStation Vita if possible.	1
8	Prototype Notes	Evaluate the prototype, notes on the program structure, things to change and improve, and so on.	1
9	FFT Research	Continued research on Fourier Transformations.	20+
10	Basic FFT Functionality	Build an additional prototype for audio/visual tests.	20+
11	Basic Visual Response	From using FFT test, generate a visual response of some kind.	20+
12	Change Notes	Note where you would change the program to implement the new visual response.	3
13	Audio data interpretation	Use FFT and other tools to read and understand audio data, and use it to define genres, speed, etc. and use that to modify aspects of the program/shaders.	15
14	Additional debugging and info gathering	Any additional testing and information gathering that I could do.	7
15	More testing with FFT	Any additional tests that could be performed with FFT.	5
16	Response Testing	Test audio/visual response.	3
17	Device Testing (2)	Test the FFT implemented version of the program on the PlayStation Vita to see how it performs.	1



Name	Begin date	End date
Prototyping	02/01/15	19/01/15
Shader Development	22/01/15	28/01/15
Shader Implementation	26/01/15	01/02/15
Functional Prototype	20/01/15	06/02/15
Functionality Testing	06/02/15	08/02/15
UI Testing	06/02/15	08/02/15
Device Testing	08/02/15	08/02/15
Prototype Notes	09/02/15	09/02/15
FFT Research	10/02/15	01/03/15
Basic FFT Functionality	10/02/15	15/03/15
Basic Visual Reponse	10/03/15	01/04/15
Change notes	01/04/15	04/04/15
Audio data interpretation	04/04/15	19/04/15
Additional debugging and i...	04/04/15	11/04/15
More FFT tests	04/04/15	09/04/15
Response testing	06/04/15	09/04/15
Device Testing (2)	10/04/15	10/04/15

The main difference being that the rest of the program is developed before even attempting FFT implementation. Early on in the project, FFT was the main focus and planned to be one of the first things to implement. This became one the largest reasons as to why the project went heavily behind schedule. Bluntly put, I heavily underestimated the complexity of Fourier transformations and their implementation. Weeks were spent researching with very little progress, on top of trying to start building a prototype on top. In the new time-plan, FFT research and implementation now take place towards the end of the project. The reasoning for this is that, even if FFT cannot be implemented fully, the groundwork will be there to be worked on, while the rest of the program will work as previously promised, and from a usability standpoint, near commercial standard.

Current Progress

The current state of the project is the prototyping stage. A “Functional” prototype is currently being built. What is meant by “Functional” is that, it will meet user standards of functionality. Meaning it is “Feature Complete”. The prototype should be able to load in the user’s music folder on the PlayStation Vita, display all the tracks on the screen and allow them to select the one to play. Following that, it should start to play the music on a new screen where the media controls are, and the visualization in the background.

If it meets this specification, it will be complete from a feature standpoint. Even without FFT to create accurate visual representation, there should still be a visual accompaniment to the music. So when the user uses the program, all of the features that they would expect to be in a program like this will be present and functional.

Currently the prototype is being created in PlayStation Mobile, and the shader code will be implemented into it. The shaders will be developed using another piece of software; most likely Shader Tool, ShaderToy, or GLSL Hacker.

Revisions and Changes in Project

As mentioned previously, several changes and decisions were made during the first several months of the project. One of the first was the decision to change from PlayStation Mobile to OpenTK. There reasoning for this was that it provided more information and lower-level access to media files. It should be noted that this decision was made during the point when FFT implementation was a priority and it had not been fully acknowledged to be as challenging as it is.

OpenTK presented problems of its own however. While in PSM, audio playback is trivial; in OpenTK, it requires more work. Additionally, graphics and drawing code require more work to use. In general, more work is required for a similar result. As the project progressed, and FFT was postponed, it became clear that using OpenTK was becoming more and more of a time sink with little to no progress being made in the most basic of features. As such, near the half-way point of the original project timeline, the OpenTK prototype was scrapped in favour of going back to PlayStation Mobile. However, the difference this time is that the prototype will not use FFT, and that will be developed and researched separately.

The program structure was something that was redesigned several times. Initially, there was no real concrete design for the program, as my understanding of all the features that should be a part of it was not fully developed. As it switched from PSM to OpenTK, the program’s structure became more defined, as Figure 8- Designs 1 and 2 of the program. shows. Multiple versions of that structure were created and revised, the one described above shows the version used in the OpenTK and PSM prototypes which was deemed to be fit for purpose and efficient.

Appendices

Figure 1:

http://upload.wikimedia.org/math/7/8/d/78d7430f6a1b49856959b95895337621.png
From the page : http://en.wikipedia.org/wiki/Sound_intensity

Figure 2 :

http://upload.wikimedia.org/math/6/b/1/6b141c2ca91d0666c4f52f7c9c3bb931.png
From the page : http://en.wikipedia.org/wiki/Sound_intensity

Figure 5:

http://betterexplained.com/wp-content/plugins/wp-latexrender/pictures/45c088dbb767150fc0bacfeb49dd49e5.png

Figure 6:

http://betterexplained.com/wp-content/plugins/wp-latexrender/pictures/faeb9c5bf2e60add63ae4a70b293c7b4.png

Figure 7 :

http://gamasutra.com/images/gaia1.jpg

Figure 12:

#	Task Name	Description	Duration (days)
1	Task List	Generate task list of different aspects of the project that need to be completed.	1
2	Time Management	Create a schedule from the task list defining possible deadlines for tasks and the appropriate time to complete them. Ideally in the form of a Gantt chart.	1
3	Interim report	Write the interim report deliverable	14
4	Risk Assessment	Define possible risks to the project and solutions to these problems.	4
5	Additional initial research	Miscellaneous project research.	8
6	Detailed Spec	A more detailed specification of the project, separate from the initial report. For personal use.	6
7	UML Diagrams	Unified Modeling Language Diagram, defining the major class and methods within the program and how they interact with each other.	2
8	Activity Diagrams	Create a diagram showing how the user operates the program which different actions.	3
9	State Diagrams	Show the different states the program can be in while in run time.	1
10	UI Design	Design the user interface, such as the display but also any additional controls.	1
11	Graphic Design / Visualizer Designs	Design a variety of graphical styles to use as visualizers. Make notes on how they act and how to build them.	5
12	FFT Research / Libraries	Gather information on current FFT libraries and their uses.	12
13	OpenTK / Shader research	Gather information about OpenTK and Shaders that are relevant to project	10
14	Concurrency research	Research code implementation that will allow for multiple processes at a time, improving performance.	8

15	Hardware Research	Make documented notes about the Vita's hardware, detailing positives and negatives in relation to the project.	2
16	Basic Visual Response	Provide a basic video/audio response.	11
17	Prototyping	Prototype the implementation of the software, with basic visual response and shaders.	8
18	Prototype Notes	Make notes on tests performed on the prototype.	11
19	Basic FFT Functionality	Use FFT to generate a waveform on screen.	4
20	Change Notes	Notes about changes in the software's design based on the information gathered from the prototype.	2
21	Class and program structure	Build the program based the class designs made previously, plus any additional improvements or features.	4
22	Beats per minute calculations	Investigate the possibility of using an algorithm to calculate BPM.	8
23	Additional debugging and info gathering	Miscellaneous bug testing and additional info that may be relevant.	15
24	Graphics / Shader implementation	Implement the graphical elements, multiple visualizers and so on.	14
25	More testing with FFT	Further testing to see the full extent of FFT.	7
26	Concurrency	Implement multi-processing to try and improve performance.	16
27	Functionality Testing	Test against the activity diagram and feature list to make sure user functions work as intended.	4
28	UI Testing	Test to see if all user interface aspects work.	2
29	Response Testing	Test to see if the visual response is accurate to the audio input, and that it produces the intended effect on the visualizer.	2
30	Device Testing	Test to see whether the software works on the PlayStation Vita system.	1

References

Better Explained, 2012. *And Interactive Guide To The Fourier Transform.* [Online]
Available at: http://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/
[Accessed 2015].

Collecchia, R., 2012. *Numbers & Notes - An Introduction To Musical Signal Processing.* 1st ed. Portland: Perfectly Scientific Press.

Dodge, C. & Jerse, T. A., 1997. *Computer Music.* 2nd ed. New York: Schirmer Books.

Klapuri, A. & Davy, . M., 2007. *Signal Processing Methods for Music Transcription.* Tampere: Springer.

Klapuri, A. & Davy, M., 2007. *Signal Processing Methods for Music Transcription.* [Online]
Available at:
http://books.google.co.uk/books?id=AF30yR41GIAC&pg=PA8&redir_esc=y#v=onepage&q&f=false
[Accessed Janurary 2015].

Wikipedia, n.d. *Sound Intensity.* [Online]
Available at: http://en.wikipedia.org/wiki/Sound_intensity
[Accessed January 2015].