THE CS171 (MUSIC) OSCILLOSCOPE

```
void setup(){
  title = "The CS171 (Music) Oscilloscope";
  name = "(Aaron) Jesse Ashmore";
  studentNum = 16394441;
  date = "07/12/2016";
}
```

Introduction

The aim of this project is to replicate a real-world oscilloscope, and in doing so, use it to display what is commonly called ‘oscilloscope music’. Oscilloscope music is a fascinating application of an X-Y oscilloscope. Through the use of audio signals and the way they are processed, it is possible to ‘draw’ images on an oscilloscope. The left channel’s output is mapped to the X-axis; the right is mapped to the Y-axis. The rest of the ‘magic’ takes place in the songs themselves. As explained in the song “How to Draw Mushrooms on an Oscilloscope”, the combination of a sine wave modulator placed on the left channel, and a cosine modulator on the right allows the coordinates of the drawn signal to be changed. Different frequencies and amplitudes of modulation allow for almost anything to be drawn 2-dimensionally, even 2D representations of 3D objects and scenes.

While drawing on an oscilloscope is no oddity, and has been done long ago with the likes of the game “Pong”, the really remarkable thing is that it *sounds* like music. One would imagine that the sounds used to draw squares, circles or 3D butterflies would be abrasive to the ear, and in no way would it resemble music. However, the artist whose music I’ve used to demonstrate this project, Jerobeam Fenderson, has managed to create songs and indeed an entire album using modulated signals, songs that are – for all intents and purposes – music.

Replicating an oscilloscope required the use of two libraries, ControlP5 *(Andreas Schlegal)* and Minim *(Damien Di Fede / Anderson Mills)*. ControlP5 was used to complement the design of the oscilloscope (created in Photoshop) with an interface. Minim, of course, was used for the audio processing and output. ControlP5 and Minim worked very well together. For example, controlling the gain of the Minim audio output with a ControlP5 knob was simple. As a result, implementing controls and features was a quick and straightforward process.

As previously mentioned, the music used to demonstrate this project was created by Jerobeam Fenderson. No permission has been obtained from the artist as it falls under the Fair Usage policies of research and education.

Overall, the project was a success, with just one main issue that may be tied to properties of Processing or Java: the accuracy of mapping the signal to the screen deteriorates significantly as more and more points are drawn to different (and distant) parts of the screen, this has yet to be resolved.

Specification

The program was designed to replicate an oscilloscope. It reads the current buffer of audio in memory and maps coordinates to the X and Y axes based on the amplitude (and modulation) of the left and right channel respectively. The program contains an interface through which to can select songs, change the gain and colour, pan the output audio, view the audio levels of each channel, and of course watch the visual output of the ‘oscilloscope ‘.

The minimum requirement for this program is a display of resolution 1280x720. It should run on most modern hardware at a respectable framerate of around 30 frames per second. However, on slower machines some breakup in the image and frame drops may be seen, this is very slightly present on the lab machines and laptops of similar hardware. Any higher-end desktop (Intel i5 ~ i7) should be able to achieve 40-60 frames per second without too much difficulty and minimal breakup will be observed. On the same machine, I noticed that Linux Mint ran the program ~20% better. I believe this is down to the utilisation of the first core of the processor. Linux seemed to use it more efficiently, with the Java thread using 100% of the first core. The Java thread on Windows 10 seemed to only use ~75% of the first core.

One other interesting observation is that regardless of the buffer size used, or brawn of the hardware on which it is run, the issue of the inaccurate mapping of coordinates remains. This leads me to believe that it is not a performance issue of the program, or at least not a hardware-based performance issue. If indeed it is a limitation of Java or Processing, perhaps multi-threading the map and draw loop would result in better mapping.

Overview of Code

I highly recommend reading the comments in the program as they provide a more concise description of what is happening alongside the code used.

As previously mentioned, the libraries used in the program are ControlP5 and Minim. These programs worked very well together and aided the quick and easy implementation of features in the program.

The program consists of several methods. Before the setup() method, I declared and initialised the majority of the variables used in the program. The song list (audio files from the data folder) is contained in an array to be used in the ControlP5 dropdown selection list.

In setup() the resolution is set, the images are loaded, and the libraries are initialised. Minim will load the song chosen at a buffer size of 1024. This size was chosen as it seemed to be the 'sweet-spot' in terms of latency and performance. The program draws all of the CP5 buttons in setup(). There is a slider to change the colour of the display, a gain and pan knob to change the audio output respectively, a song list to choose the song desired, and two sliders which function as 'audio level' meters for both channels. Following this there is just some code to tidy up the labels and style of the controls.

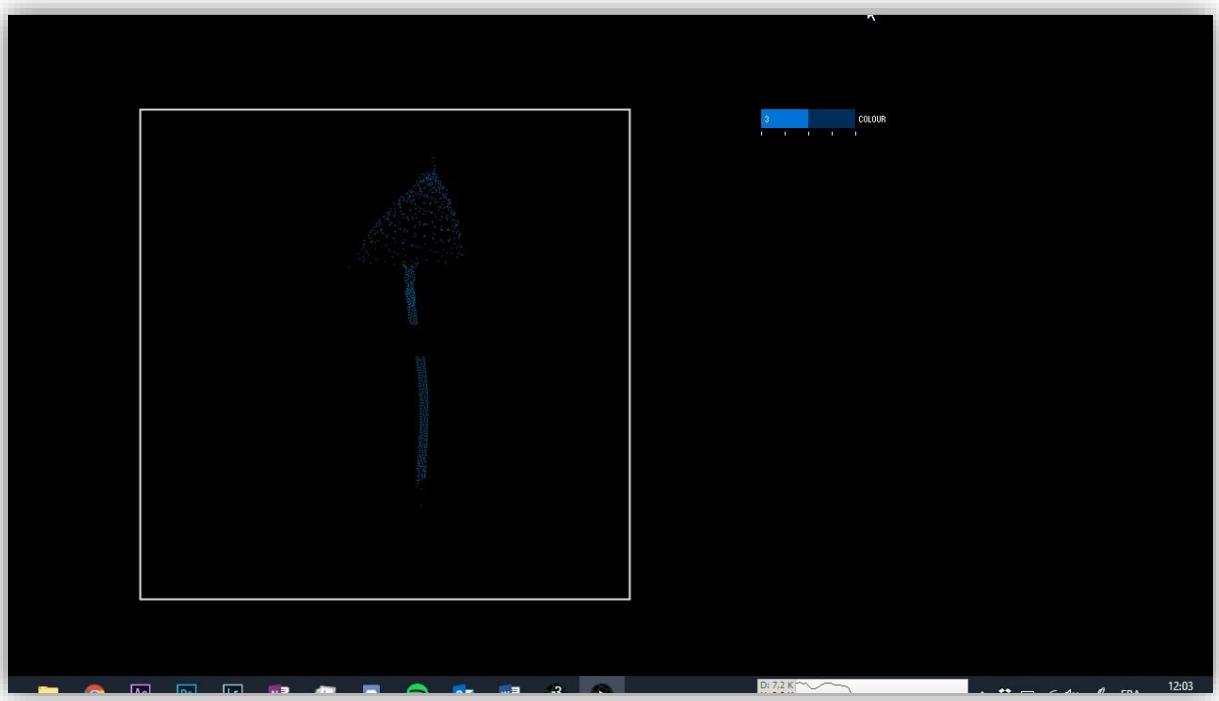
The draw() method is of course where everything takes place. The program begins with an introduction screen: a background created in Photoshop for the project. It waits for the user to press a key before moving on to the main program. The stroke weight is set to 0 and the colour of the stroke is set to the colour of the ellipses, just in case a stroke is drawn. Since the levels sliders are constantly changing in draw(), I had to add some more tidy-up-code in order to prevent the labels being redrawn. The for-loop is where the main function of the program occurs. It begins by reading through the buffer in the memory, retrieving the value of player.get(i) for each channel. It maps these values to the X and Y axes within the constraints of a 240x240 area. These X and Y values are then used to draw ellipses. The X position of each ellipse is also affected by the user-chosen pan value, much like a traditional oscilloscope. The draw() method finishes with some if-statements which control the play, pause and stop buttons. These statements check for mouse position and mouse press.

After draw() we have the final methods for each function of the CP5 controls. The colour values are predefined, selectable via a slider and implemented through switch-statements. Dropdown() pauses the currently-playing song when a new one is chosen. The choice is then taken from the array and ".mp3" is concatenated to this choice so that the player can read the file. Gain() simply takes the knob value and applies it to player.setGain(). Pan() functions the exact same way.

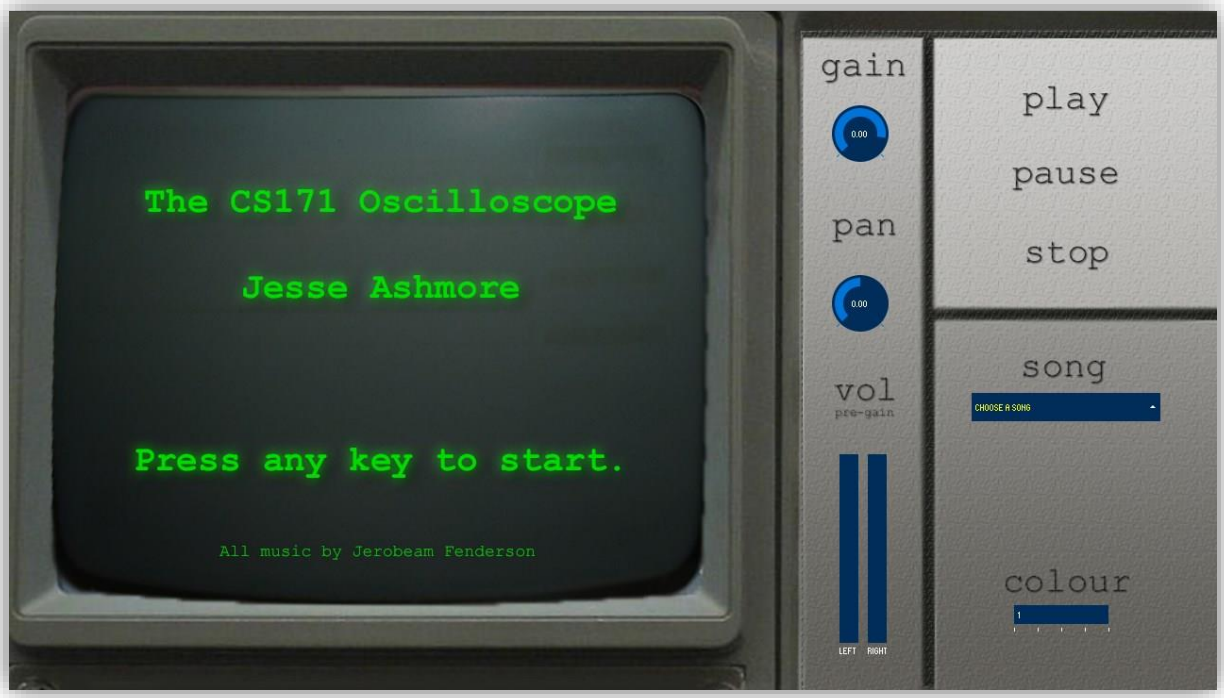
As a final aside, ellipses were chosen over lines to represent the oscilloscope display. This is due to the inaccurate mapping described earlier. While noticeable with ellipses, the loose mapping is not overly distracting or offensive; this is a completely different story with lines. Lines would zig-zag and overlap horribly once inaccuracies manifested themselves, it was a disaster. Thus, ellipses were chosen as the preferred object to draw the values of the audio buffer.

Testing

Here is a very early version of the oscilloscope, all I had was an area for the oscilloscope display and a colour slider. This version of the program was the barebones of everything else and verified that my concept and chosen libraries would function as expected.



Here is the welcome screen.



Here is the program functioning as of 07/12/2016

Testing was smooth throughout; gain and pan both worked from the first implementation. The output was tweaked a bit as I tried to alter buffer sizes and decide between lines vs ellipses. That said, being an audio/visual program it was very easy to test as it was nearly all based on expected output due to reference videos of Jerobeam's music on YouTube.

Conclusion

When I first *saw* the music of Jerobeam Fenderson I was fascinated. All I wanted to do was share the discovery and show others that you could draw images on an oscilloscope. For weeks afterwards I wondered what on earth I would end up doing for my end-of-semester project. Finally, I connected the two ideas and realised that the CS171 module gave me that opportunity to share his ‘visual’ music.

I believe that I achieved what I set out to do: create a near-replicate of an oscilloscope. The functionality of the program is more than satisfactory for the time being and there isn’t much I would or could add, it’s an oscilloscope after all. At some point I would like to add a folder scanning sentinel that will automatically put mp3 files found in the data folder into the array. Aside from this, clearing up or optimising the program further in order to reduce inaccurate mapping is the first thing I’d like to work on.

Overall, I am very pleased with the outcome of the project. It was nice to apply the concepts that I’ve learned throughout the CS171 module in a final project that was both informative and exciting to work on.