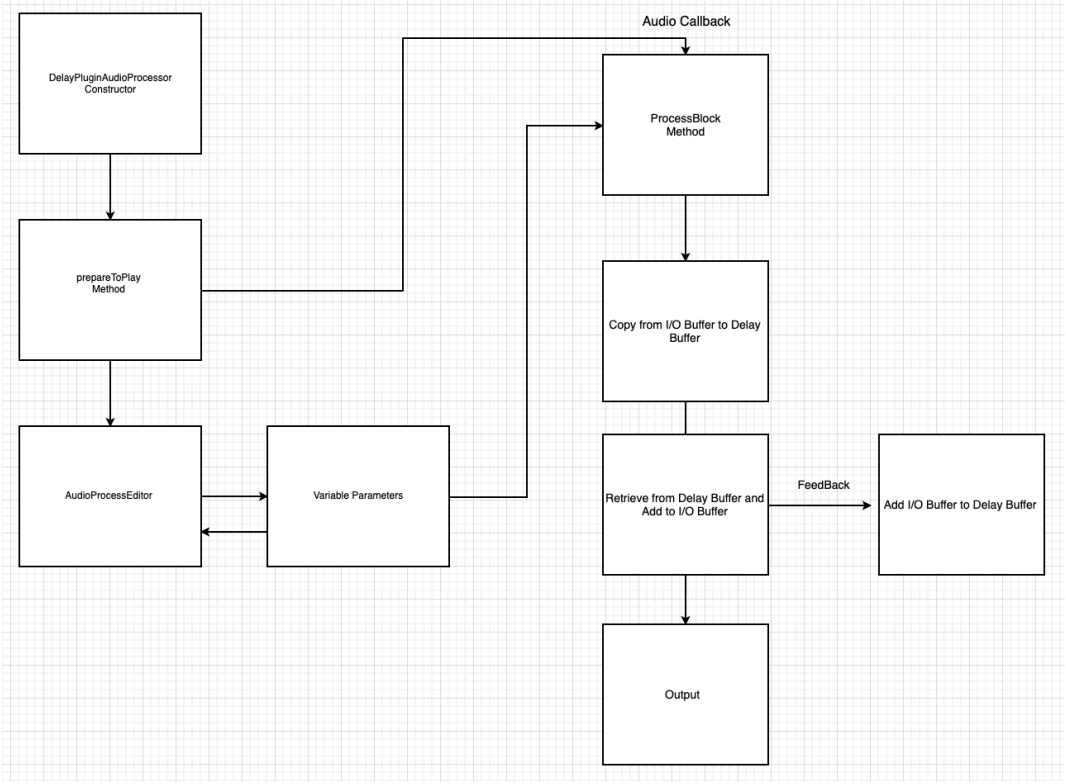
Miles Grossenbacher
C Programming in Music Tech
5/9/19

JUCE Audio Plugin: Delay Effect

The purpose of my final project was to create an audio delay effect through the JUCE framework using the concept of circular buffering. I find the delay effect, particularly feedback delay, to be one of the most creative audio effects when it comes to musicality, so in completing this project, I hoped to understand the basic mechanics of how this effect is created, so that I could further expand on this project while having a foundational understanding from which to work.

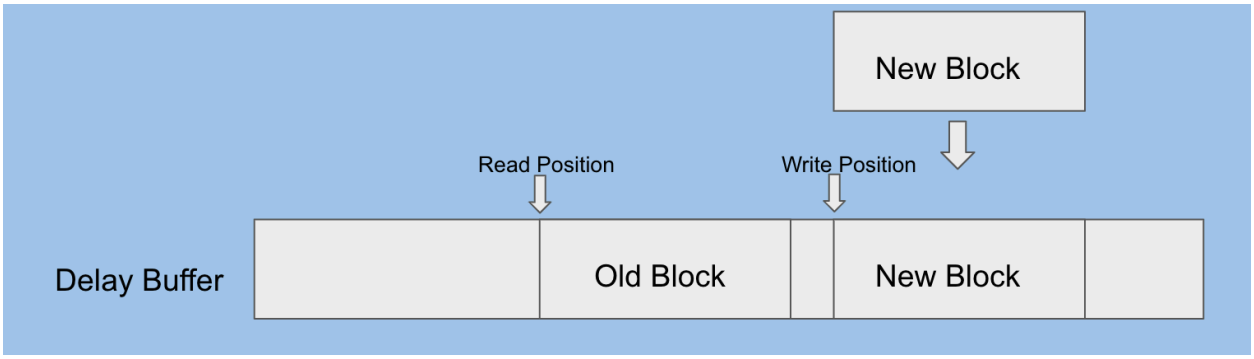
I built my project using the JUCE audio-plugin skeleton, so my project is designed to work inside an audio application host. Specifically, I designed and tested my project with the digital audio workstation, Logic Pro X, on the Mac OS. The JUCE skeleton provides pre-defined functions in which to write code, and the basic flow of the program is seen in the diagram below:



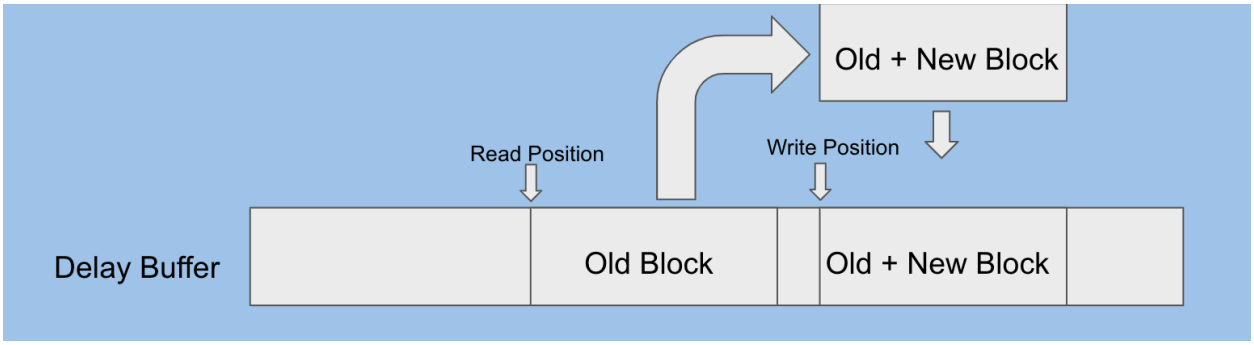
Miles Grossenbacher
C Programming in Music Tech
5/9/19

The functional blocks in which I wrote my code were the prepareToPlay block, The Audio Process Editor, and the Audio Callback. In the *prepareToPlay* method inside the *PluginProcessor.cpp* file, the primary action was that I initialized my delay storage buffer in which I could store up to two seconds worth of audio samples. This served as the circular buffer that I could retrieve blocks of samples from when required. The *PluginEditor.cpp* file was where I created the GUI for my audio plugin. Using the JUCE editor and slider classes, I was able to create interactive, user controls that updated the values of variables that effected the incoming audio signal. In the *DelayPluginAudioProcessorEditor* method I created sliders and knobs to control variable parameters. In the *sliderValueChanged* method, I had the slider variables communicate with the variables from my *PluginProcessor.cpp* file so that they would interact correctly. I was also able to edit the graphics of my plugin within the *paint* method to customize the look of my plugin to my liking.

Finally, the digital signal processing part of my plugin occurred in the *processBlock* method of my *PluginProcessor.cpp* file. In this method, I copied incoming buffers of an audio signal into my delay buffer, fetched old blocks of audio to add to the incoming buffer's signal, and created the feedback portion of my signal by scaling and coping the combined signal back into the delay buffer:



Miles Grossenbacher
C Programming in Music Tech
5/9/19



I then updated the write position in the delay buffer by moving it forward by the value of a block of audio, and recalculated the read position based on how many milliseconds back in time the user wanted to retrieve audio data from. Because I implemented a circular buffer, if a block didn't fit into the remaining space at the top of the buffer, it would wrap around and overwrite audio data that was older than 2 seconds in the past. My source code is thoroughly commented, so the actual process is easier to follow by reading through the function in the source code. For moving audio data from one buffer to another, I used the JUCE methods, copyFromWithRamp and addFromWithRamp which is the standard way of moving data between buffers in JUCE. Also because I worked in JUCE, I only used the JUCE libraries.

The computational complexity in my program comes from copying and adding sound data between the I/O buffer and my delay buffer. Every time I perform an operation, I do it to every sample in an audio block's worth of samples, so my Big O complexity is O(n).

I compiled my program inside the JUCE framework, which can build the audio plugin into a variety of formats. Since I used Logic Pro X, I created an AU plugin, which is the format that Logic works with, but the same source code can be compiled into a VST or VST3 plugin I would assume.

Miles Grossenbacher
C Programming in Music Tech
5/9/19

Citations: I modeled my circular buffer from a tutorial by JUCE developer Joshua Hodge and a tape delay plugin by Daniel Walz: https://www.youtube.com/watch?v=IRFUYGkMV8w&t=824s

https://github.com/ffAudio/ffTapeDelay