ECE484 Audio Signal Processing Final Project Report

Matthew Haussmann
MUCS | V00827158
haussmannm@gmail.coM

ABSTRACT

A Stereo Chorus/Flanger plug-in was implemented in the JUCE C++ framework with delay line effect architecture using linear interpolation. This report introduces the "Matt's Epic Delay" project, goes over deliverables and specifications for the project, walks through the design and its process before moving into the components. Evaluation and testing are discussed, and the prototype plugin is presented.

1. INTRODUCTION

During the fall semester of 2020 at UVic, this project was completed for the class ECE484 Audio Signal Processing instructed by Dr. Peter Driessen. The project was to implement an audio effect that the class had studied during the semester on your own chosen platform. The effect I implemented was a chorus/flanger delay line effect unit as a plug-in for a Digital Audio Workstation (DAW) using the JUCE C++ framework. My initial proposal for the project was to implement a phase vocoder in JUCE, but my focus changed largely because a future project for which implementing a phase vocoder would have largely helped with also changed. The name "Matt's Epic Delay" is a tongue-in-cheek double entendre because of this switch of decision.

JUCE and the Projucer app help audio developers create multi-format audio plug-ins with ease by handling much of the background work and many library implementations of useful functions and packages. It is able to output many plug-in types, including VST, VST3, AU, AAX, or as a standalone program. JUCE was used under a free educational license.

The chorus effect is an effect that uses delay line modulation to create depth and the illusion of multiple signals slightly out of tune with each other, similar to a "choir" of the input signal. A flanger effect uses the same implementation, but different parameter values to produce a sweeping comb filter effect. When used tastefully, these effects create texture and interest on the input signals, and when parameters are set to extreme ranges then they can create heavily distorted and unrecognizable outputs that can be used for sound design or other purposes.

A frequency-shifting vibrato effect can also be created when feedback is set to zero and dry/wet is set to 100% wet.

2. DELIVERABLES AND SPECIFICATIONS

A number of controllable features are specified as the deliverables for the project. The adjustable parameters for controlling the plugin's algorithm include a dry/wet dial for controlling the output mix of the unaffected "dry" input signal and the affected "wet" signal. A dial for controlling the feedback gain coefficient of the delay line tap in included. Included Controllable parameter dials of the delay line modulation Low Frequency Oscillator (LFO) are: The depth AKA the sweep width of the oscillator; the rate at which the LFO oscillates (0.1-20Hz); the left/right LFO phase offset between the left and right stereo channels (0 - 2*pi). As well, a menu box for choice between a Chorus effect and a Flanger effect, which adjust the delay line time settings between relatively longer (1-5ms) and relatively shorter (5-30ms) times, respectively. These parameters are all controllable from the plug-in's Graphical User Interface (GUI).

An implementation of statefulness that works with mainstream DAWs to allow for saving the current parameter settings and coming back to them when reloading the project is included as a feature that utilizes XML files.

As a deliverable as a whole, the format of the project is an Audio Units plug-in and a VST3 plug-in files with an associated Xcode file.

3. DESIGN AND DESIGN PROCESS

The design of the plug-in algorithm took a natural progression. I decided I wanted a usable, real-time product that could be used on multiple platforms. As I have interest in developing audio plugins and in C++ as a programming language, the JUCE framework was decided on as a starting point for the project implementation since it satisfied each of these

requirements. Alternative choices for platforms could have been in MATLAB or on a microcontroller such as Ardiuno or Raspberry Pi, however the prospect of using my own work in a DAW in my future creative musical endeavors was a strong allure which won out.

The implementation of the algorithm was the next step. I began by implementing a gain function to get control of the input and output signal, which eventually morphed into the dry/wet parameter. After that was the implementing the delay line, which led to the feedback control as well. Once the delay line was working and adjustable, an LFO was implemented to modulate it.

The LFO's design needed an implementation of inter-sample interpolation. I decided on Linear Interpolation (LI) for simplicity's sake, which is much better than nearest-neighbour interpolation but can still introduce noise and aliasing to the output signal. LI creates a line between two successive samples, and places the inter-sample fraction on that line and returns the value at the location. There are a number of other interpolation methods available, many of which produce better results than LI. Second-order polynomial interpolation uses three successive samples in its calculation. There are many types of what is known as cubic interpolation, where the simplest type uses the four samples surrounding the interpolation location.[1] For a prototype plug-in, LI was satisfactory.

Once the LFO and its associated linear interpolation were working, a second LFO was implemented with a phase offset parameter for the left and right stereo channels. Descriptions of these component modules are the topic of the next section.

4. COMPONENTS

The program has multiple components, as does the effect algorithm. I will begin with the structure. The PluginEditor component deals with the GUI and is separate from the PluginProcessor component, which is the guts of the plugin and contains the effect algorithm. Both files use implementations of dedicated classes, MattsEpicDelayAudioProcessorEditor and MattsEpicDelayAudioProcessor which inherit from JUCE's AudioProcessorEditor and AudioProcessor classes respectively. The constructor methods of each class of each file set up the parameters and instantiate variables. The editor uses the JUCE Slider class, AudioParameter classes, and a paint method to generate and work with the graphics of the GUI and to link functionality with the parameters.

The processor for this algorithm works mainly with two functions, a prepareToPlay function, and the processBlock function. The prepareToPlay prepares the computer state for playing audio when starting/stopping. It initializes data for the current sample rate of the project, setting parameters including LFOPhase, CircularBufferLength, CircularBufferLeft, CircularBufferLeft, and CircularBufferWriteHead. It also Zeros the new allocated memory to clear any garbage data that might be present.

The circular buffers need to be just that, circular, so when readhead and writehead pointers exceed index ranges then they must be looped back to the beginning or end of the buffers. This is implemented using IF statements

The processBlock is where the input audio signal processing happens. First it gets the number of audio channels (two), clears the buffers, and then begins to loop. Within the FOR loop: It uses two circular buffers with readhead and writehead pointers; calculates signal feedback generates and applies the LFOs with the phase offset, rate, and depth parameters; uses a helper Linear Interpolation function for inter-sample indexing; and calculates dry and wet amounts of signal to output.

The first LFO was created by dividing the variable mLFOPhase, scaled between -1 and 1, by the rate parameter multiplied by the sample rate, and then wrapping it around back to negative one when it reached the top. The output of this was multiplied by 2*Pi and evaluated within a sin() function. This was then subtracted from the smoothed delay time to create the constant warbling and frequency shifting character of the effects.

Once the parameters were mapped and mostly working, it was time to work on the flanging. This involved doubling up the LFO, one for left channel and one for right channel to create the stereo effect with a controllable offset.

The right LFO was created and calculated from the LFOPhase plus the phase offset parameter. Both LFOs are then mapped to appropriate delay time ranges, with the mapping being chosen between the chorus range (5-30ms) and the flanger range (1-5ms). The delay time in samples is calculated separately for left and right channels by multiplying the sample rate with the mapped output phases of the LFOs.

5. PROTOTYPE

The prototype [1] is a fully working plug-in effect unit that can be used in a standard mainstream DAW for real-time audio processing. This is the format of many audio tools today.

[1] Reiss, J. D., & McPherson, A. P. (2015). Chapter 2: Delay Line Effects. In Audio effects: Theory, implementation, and application (pp. 21-56). Boca Raton, FL: CRC Press.

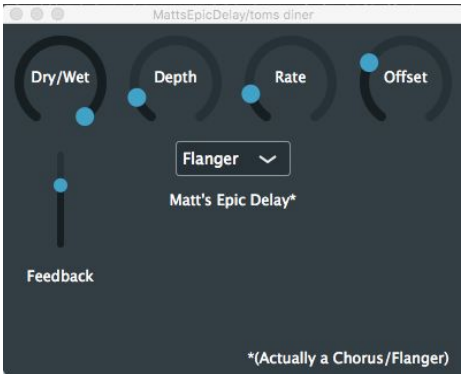

Figure 1. The prototype GUI on light flanger settings.

6. EVALUATION AND TESTING

Of course, testing is a natural part of the development process, so many tests of the modules were performed as they were being coded. However, the end results are where the interest lies. As input signals are manipulated and processed, the results are audible. The question is, are they musically useful sounds? The different settings of the parameters produce different results, and particularly the delay time setting choice between a chorus effect and a flanger effect changes this too. Testing materials included a dry drum loop, a simple guitar phrase, and the first phrase of Suzanne Vega's a cappella song Tom's Diner, which is a classic and canonical testing song for audio development.

Beginning with the chorus setting, turning the dry/wet and feedback dials to medium, the rate and depth to low-mid, and the phase offset at a minimum produced a pleasant spreading of Vega's voice, though there is a subtly noticeable metallic quality to the sound. For the drum loop, turning the dry/wet up to full and increasing the rate a small amount created a similar spread, but an odd-sounding wavering can be heard in the sounds, particularly in the tails created by the medium reverb. This is the result of the LFOs' increased rate; I would suggest a slower LFO rate for a texturally useful application to drums.

For the Vibrato effect, feedback and phase offset values are set to zero, dry/wet is at 100% wet, and depth and rate are the adjustable variables. I set the rate to medium-slow and the depth to medium-low, and Tom's Diner was sounding a little kooky as Vega's pitch was modulated at a constant rate, making the ability to distinguish the centre pitch rather difficult. I suggest a slightly faster rate, and a lower depth for a capella singing. However, even with adjusting the parameters slightly, because it is at a constant rate there is no musically-informed phrasing, where a proper singer often begins their notes as a straight tone, and then adds vibrato once the pitch is more established. Volume is also reduced and increased as the pitch is modulated up and down, so without an implementation of that in the algorithm it will continue to sound unnatural particularly on human singing voices. For the guitar track, I turned the rate and the depth up a small amount, and the result sounded like a proper guitar vibrato pedal. It does not sound like a guitar player's string-bending induced vibrato, but similar facts to the singing voice can be said about musical playing of the guitar, as guitar players do not manually bend every single note they play, especially not at a constant rate.

Moving on to a lighter application of the flanger setting, I set the phase offset to nearly in the middle, set dry/wet to 100% wet, feedback to mid-high, and turned the rate and depth to medium. Tom's Diner is given a lovely stereo spread and pleasing ethereal texture, as well as a sounding somewhat in a pipe. The comb filtering created by the repeated feedback is the cause of this. When applied to the guitar, the pipe effect is less noticeable but the pleasing qualities are preserved. This is a successful example of a very musical application and could be used in a professional production to good effect. It does not matter in a testing situation if chorus or flanger is chosen as a setting because there is no repeated feedback signal. However, in a performance setting, it would be advisable to set it to flanger to reduce the latency, as the 100% wet signal has a shorter delay time than the chorus setting.

As the flanger is a very versatile effect, I turned the depth, rate, and feedback up, and applied it to Tom's Diner, creating a highly affected "electronic" sounding textural flanging effect. This could be applied well in electronic music, where the intent of creating "natural" sounds is discarded in favour of more novel sounds.

Next, by turning down the rate to less than one second, and turning up the depth, to around 75%, an effect often referred to as the "jet sweep" was created on the drum loop. This sounds very similar to the drums heard on Led Zeppelin's song "Kashmir," though the effect is more pronounced than on Kashmir.

Beyond traditional musical applications, I turned many of the parameters up high to listen to the extreme ranges of the effect. Setting to chorus with high feedback, rate, phase offset, but low depth, a spread out, wispy, metallic, and electronic reverb effect was produced largely due to the higher reverb feedback gain coefficient. Since delay is modulated between 5-30ms on the chorus setting, sitting in the middle of that range is around 17ms. Reverb is perceived usually at repeated delay times of less than 10ms, however the drawing out of the signal is the most noticeable feature with these settings.

Lastly, since JUCE has created parameter classes that can be automated within a DAW, I played with this by beginning with a normal rendition of Tom's Diner, but quickly turning every parameter up to almost full, Suzanne's voice became completely unrecognizable as it was mangled by the high depth, rate, and feedback. Listen: The only recognizable musical feature to be preserved was a sense of rhythm, as the input signal was

fed into the mangling flanger and changed the texture at each new note onset according to the rhythm of the melody of Tom's Diner. This extreme setting could only be useful in sound design applications or for humorous effect, but I found it was fun to explore the extremes of the effect unit I created.

Lastly, to test the statefulness implementation, I simply changed the parameters from the default settings and saved the Ableton Live project, then reloaded it. The values I changed remained the same instead of being reset to the default values.

7. CONCLUSION

Summarizing, the strength of this design is definitely the flanger effect with properly adjusted settings for the desired effect, and could be used in a professional setting in the right circumstances. The vibrato settings on the guitar could be applied to other instruments in a real-time performance setting to good effect by choosing the flanger setting to reduce latency. The implementation has some metallic qualities to the sound for the chorus, and as this is usually not as desirable, it is less useful. The extreme ranges of the parameters could possibly also find application. If this was to be released, I would implement more complex interpolation methods and test the differences between the methods to hear the results myself, but as there is no way to test without having two side-by-side implementations, I cannot comment on the difference, and am relying on outside information sources for the suggestions on linear interpolation's comparably audible qualities.

The design satisfied both my vision and the specifications I set, and the algorithm was implemented with success in a manner of a natural progression during the development process. I am happy with how this project turned out.

8. REFERENCES

[1] Reiss, J. D., & McPherson, A. P. (2015). Chapter 2: Delay Line Effects. In *Audio effects: Theory, implementation, and application* (pp. 21-56). Boca Raton, FL: CRC Press.

[2] Zölzer, U. (2011). *DAFX: Digital audio effects*. Chichester: Wiley.

[3] Course Catalog | Kadenze. (n.d.). Retrieved November 30, 2020, from https://www.kadenze.com/courses/intro-to-audio-plugin-development