POLITECNICO
MILANO 1863

Music And Acoustic Engineering

Computer Music - Languages and Systems
Homework #2 (JUCE)
Group: Algorhythmics - A.Y. 2022/2023

NEL-19

Group Members:

Alice Sironi
Cecilia Raho
Stefano Ravasi
Yan Zhuang

https://github.com/Mrugalla/NEL-19

Contents

1 Introduction

The aim of the NEL-19 plugin is to develop a vibrato effect based on re-sampling, using a feed-forward delay to modulate the input signal. This effect can be used in different situation, to modify a simple recorded signal or a guitar during a live exhibition. The user has the possibility to create several types of vibrato texture by selecting from seven modulators with different functionalities, that will be explained in the following paragraphs.

This report is divided into three sections. In the first one there is the graphic representation of the plugin through the block diagram. The second chapter is dedicated to the explanation of the Graphic Unit Interface (GUI): how the creator has distributed the sliders in the interface and what are the elements for. In the last part there is an explanation of how the creator implements the code to obtain these results.

2 Block Diagram

The common representation of a vibrato is very simple: a delay, modulated by an LFO, is applied to the input signal, in order to obtain the final result (see Figure 1).

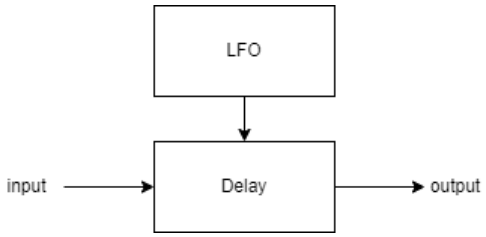

Figure 1: Block Diagram Vibrato

This plugin, however, adds more useful elements for change the parameters of the effect. There is a Feedback, filtered by a Lowpass Filter (called *dump* in the plugin), that is commonly use in effects like Flanger and Phaser to determinate the amount of output signal that falls into the circuit.

In addition, the Delay is modulated by a mix (*Mods Mix*) of different modulators (*Modulators*), that the user can select to create different effects: LFO, as the standard implementation, or something like Perlin Noise, Audio-Rate modulator and so on. To create the output signal, there is a mix between dry and wet signals (*Mix*).

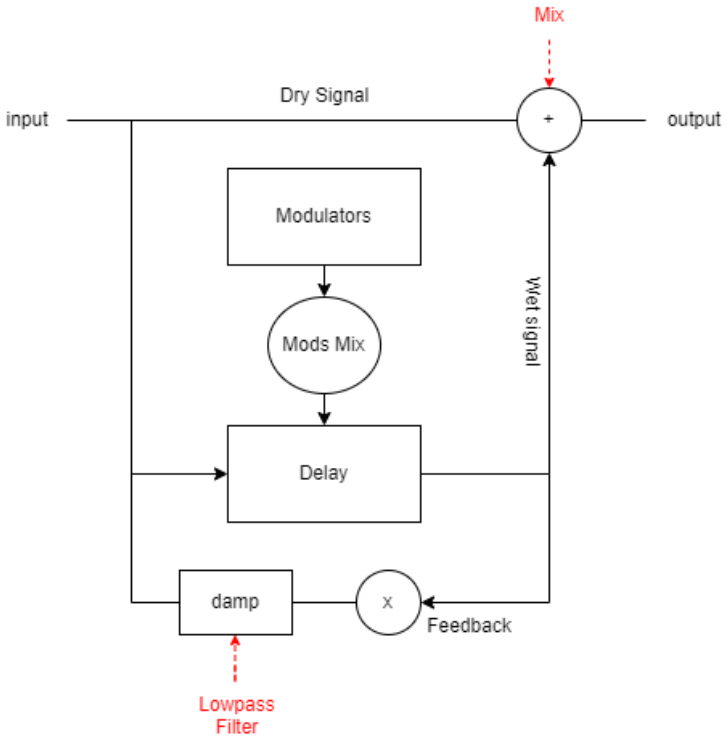

Figure 2: Block Diagram NEL-19

3 Features and GUI

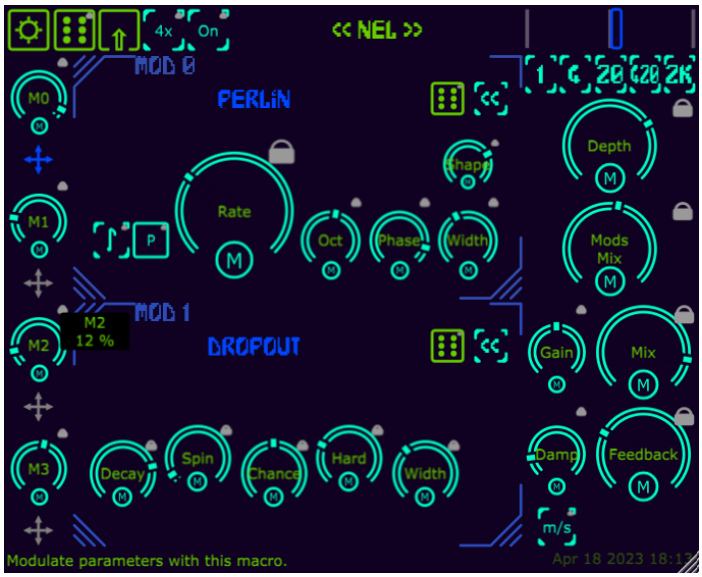

Figure 3: GUI

The user interface area is divided into four sections. The left part contains four Macros, sliders that control creative combinations of effects and parameters. The right part contains the main components of a vibrato effect.

- In the upper part, the user can select different values to resize the internal *Delay*. The values are 1, 4, 20, 420 and 2000 ms.
- The *Depth* modulator expressed as a percentage.
- The *Mods Mix* slider create an interpolation between the selected modulators. Each contribution has different value and their sum is based on a 100-point scale. For example, if the first modulator has a 75% of contribution, the other one has a 25% of presence in the final result.
- The *Gain* is applied only to the wet signal, because some modulators change the volume slightly.
- The *Mix* knob allows the user to mix the vibrato and the dry signal to obtain new effects like Chorus or Flanger. The ratio between the two contribution values is on a 100-point scale: if

there is a 35% of vibrato, the contribution of the dry signal will be 65%.

- The *Damp* slider controls the cutoff frequency of the Lowpass Filter in the Feedback path. The range of values is between 40 Hz and 8 kHz.
- The *Feedback* knob dials the contribution of output signal in the system. It goes from -100% to 100%.
- The *Stereo* configuration button can be set either left/right or middle/side channels.

In the middle of the GUI the user can select different modulators to create several vibrato textures. It is possible to menage two modulators at the same time, changing their components both manually and randomly (with the *Random* button). There are seven choices.

- The *Perlin Noise* modulator uses natural noise to modulate the vibrato. The user can switch between the rates unit (Hz or bpm) with the ♪ button; can create a random or a preset combination of rate, bpm and transport info with the **P** button; can modulate the rate of the noise in Hz (from 1 mHz to 40 Hz) with the corresponding slider; can switch the octave of the noise with *Oct* knob; can apply a phase shift to the signal (from 0° to 720°) with the *Phase* knob; can apply a phase offset to the right channel expressed as a percentage with the *Width* slider; can choose the shape of the Perlin Noise: step, linear and round.
- The *Audio-Rate* modulator uses a midi-note-controlled oscillator to modulate the vibrato. The user can set the evolution of the sound, changing the values of attack, decrease, sustain and release; transpose the oscillator in octave steps (*Oct* slider), in semitone steps (*Semi* slider) or in finetone steps (*Fine* slider); define the modulator's stereo-width (*Width* knob from -180° to 180°) and the oscillator's retune speed (*Legato* knob from 1 to 2000 ms).
- The *Dropout* modulator simulates random pitch dropouts similar to tape artefact. In this modulator, the user can approximate the decay of the dropout (from 10 to 10000 ms); give it a spin from 100 mHz to 40 Hz; the likeliness of new dropouts to appear, expressed in ms; define the smoothness of the dropout (*Hard* knob); define the modulator's stereo-width.

- The *Envelope Follower* modulates the vibrato according to the signal's energy. There are four knobs: the first one controls the modulator's input gain, expressed in dB (from -20 dB to 80 dB); the *Attack* knob and the *Release* knob change, respectively, the attack and the release time in milliseconds (both from 1 ms to 2000 ms); the last one defines the modulator's stereo-width.
- The *Macro* modulator manipulates the vibrato's internal delay.
- The *Pitch Band* modulates the vibrato.
- The *LFO* modulates the vibrato with a classic LFO shapes. The *Rate* slider adjusts the frequency of the LFO in Hz; the *WT* slider creates an interpolation between the LFO's waveform, with a graphic representation on the right part; *Width* adds a phase offset to the right channel; *Phase* adds a phase offset to the LFO; the *free/sync* button switches between free running and temposync LFOs.

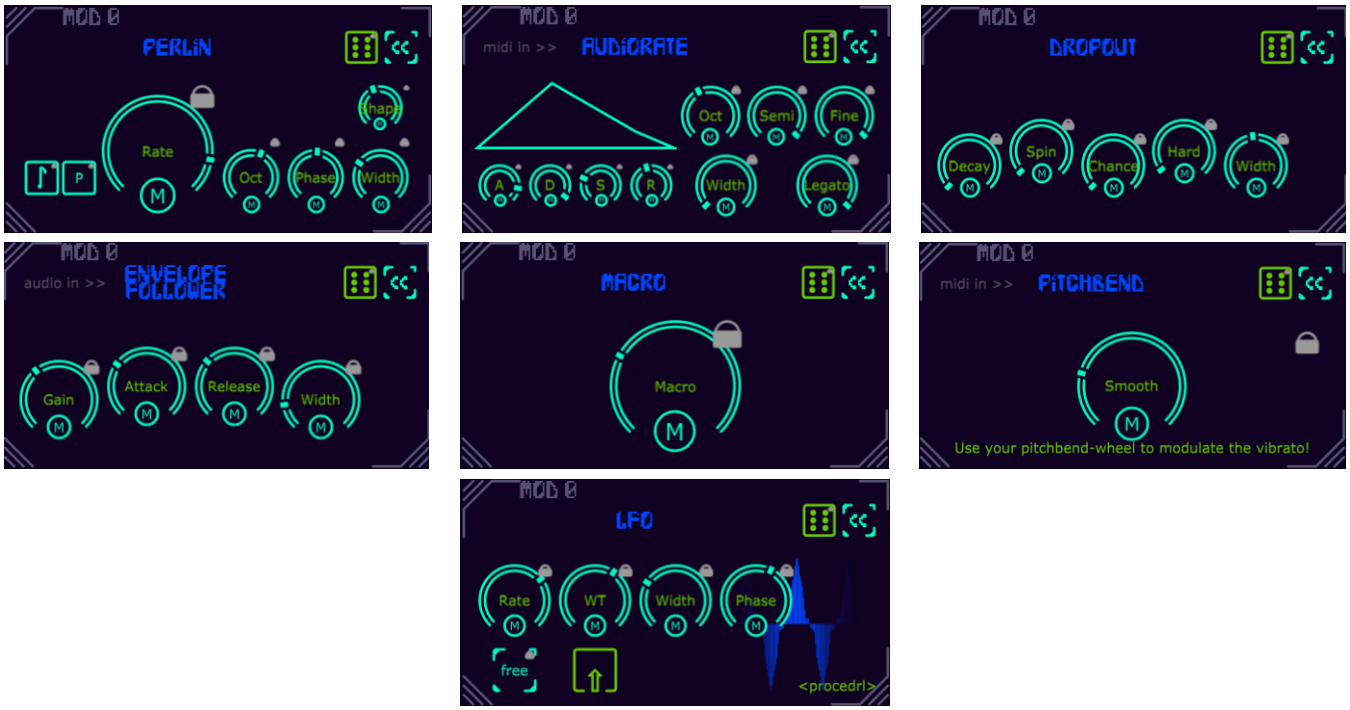

Figure 4: Modulators

In the upper part of the GUI, there are five buttons.

- *Settings*, in which the user can see the buffer size, can change the colors of the elements in the GUI and can enable or disable the help comments.
- The *Random* button selects randomly the combination of two modulators.
- The *Preset Browser*, where the user can choose different presets that have been loaded.
- The *Oversampling* button has two values, 1x is the standard one and 4x ensures a better result.
- The *Lookahead* button aligns the average position of the vibrato with the dry signal.

4 Implementation

This plugin is wrote in C++ language using the JUCE framework. The most important class used to create this plugin is ***Nel19AudioProcessor***, that inherits all the methods from ***AudioProcessor***, a class provided by JUCE. The class is defined in *PluginProcessor.h* file and used in *PluginProcessor.cpp*, recalling and implementing some methods. ***Nel19AudioProcessor*** takes care of most of the tasks used to produce the vibrato effect, importing other classes used to generate the effect. These components can be divided by their *namespace* definition.

4.1 Vibrato

The namespace **vibrato**, defined in *Modulator.h* and *Vibrato.h*, creates the main effect of the plugin. It inherits from ***AudioBuffer*** and recalls the classes we will discuss below, like *WHead*, *Lowpass* and *Smooth*, to generate the effect. In the modulator code there is another important class, defined as ***Modulator*** in which there are several subclasses that implement the modulators seen before (see chapter 3): ***Perlin, AudioRate, Dropout, EnvFol, Macro, PitchBend, LFO***. The ***Perlin*** is the only modulator implemented in a separate file, in order to create the noisy component that characterizes this effect. This class uses also smooth and interpolation to generate the effect.

4.2 Smooth

In the file *Smooth.h*, under the namespace **smooth**, are defined the following classes.

- ***Block*** is a block-based parameter smoother. It works as a buffer.
- ***Lowpass*** creates the Lowpass filter.
- ***Smooth*** uses the previous classes to define the smoothing.

The *Smooth.cpp* code implements the smoothing and applies it to the vibrato effect.

4.3 DryWet

The namespace **drywet** is defined in *DryWetProcessor.h*, that inherits from ***AudioBuffer*** class provided by JUCE. There are implemented two classes.

- ***FFDelay*** creates a delay with a ring buffer and applies it to the wet signal (defined in *WHead.h*).
- ***Processor*** combines dry and wet signals, making equal the loudness curves and smoothing the mixed parameter values, to create the output audio.

4.4 ModSys

The namespace **modSys6** is used to defined the parameters of modulation (modulators, macros and vibrato's main components). It is contained in *ModSys.h* and the main class is ***ModSys***, that inherits from ***AudioProcessor***. Another important class is ***Param*** that inherits from ***AudioProcessorParameter***. It takes the parameters from the classes defined in this file and adds them to ***AudioProcessor***.

4.5 Oversampling

The namespace **oversampling** can be found in four different files: *Oversampling.h, Filter.h, ConvolutionFilter.h and IIRFilter.h.* Each class is based on ***AudioBuffer***. Start from the Oversampling

code:

- ***Processor*** recalls two classes, *ConvolutionFilter* and *LowkeyChebyshevFilter* (from *ConvolutionFilter.h* and *IIRFilter.h*, respectively), and uses their parameters to implement the Convolution filter and the Chebyshev filter.
- ***OversamplerWithShelf*** applies the process of oversampling.

In the ConvolutionFilter code there are three classes:

- ***ImpulseResponse***, as we can guess from the name, menages the impulse response of the system, using the Nyquist Theorem.
- ***Convolution*** implements the process of convolution between the impulse response and the input signal.
- The ***ConvolutionFilter*** class applies filter to the operation described above to extract specific features from input datas.

The classes defined in the IIRFilter code are the following:

- ***IIR*** implements the IIR filter with its components.
- ***MakeChebyshev*** and ***LowkeyChebyshevFilter*** creates the Chebyshev filter and applies it to the signal.

4.6 Interpolation

The namespace **interpolation**, defined in *Interpolation.h*, implements four types of interpolation used selectively for the implementation of different effects. For example, the *lerp* interpolation is used in Dream Arp, Psychosis and Shoegaze, the *spline* interpolation in Flanger and Broken Tape, and so on. All the defined effects are grouped in the *preset* folder.

5 Conclusions

The NEL-19 plugin implements a vibrato effect that can be modified through seven different modulators. It is a *work in progress*, the creator adds new code components, on a weekly basis, to improve its functioning. We analyse one of the latest version of the code, to which improvements could be made:

- NEL-19 works only on Windows. Its use could be extended to other Operating System, like Mac or Linux.
- The creator defines *presets* for several effects that the user can use to generate the effect, but there isn't the possibility to select them during the performance. The user must change the values programmatically.
- In *Outtakes.h* code the creator tries to implement an expensive method to draw 3D grid over the components, but it doesn't work. It is just a graphical improvement.