# Computer Music - Languages and Systems
# Homework 2
# FLANGELVS: a flanger effect plugin

Group 6 - GELVS:

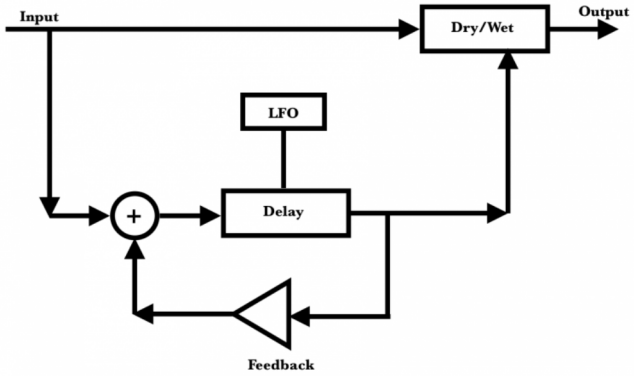
Del Moro Samuele, di Clerico Letizia, Negroni Viola, Perego Gabriele, Roncuzzi Enrico.

## 1 Goal

Implement a flanger effect plugin with feedback in Juce, choosing the parameters that should be controlled by the user using the GUI.

## 2 Concept

The Flanger effect is based on the idea of duplicating and retarding the input signal through simple delays.
The time interval in which the channels are delayed is shorter (5-20 milliseconds) compared to similar sound effects, like chorus or phaser .
Other than feeding the original signal with a delayed version of it, this plugin also includes a feedback control, which sends a scaled copy of the delay output back to the input, causing the sound to continuously repeat over time.



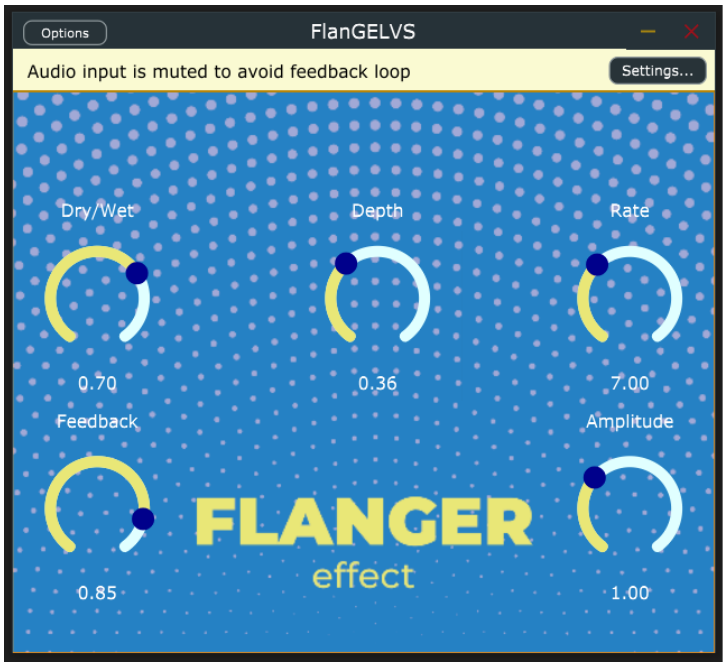

By means of the user interface it is possible to modify some parameters related to the effect: besides the amount of feedback, the user can also act on the overall amount of effect ("Dry/Wet" slider) and on 3 parameters that control the low-frequency oscillator (LFO) through which the delay is handled, "Depth", "Amplitude" and "Rate".

In order to make the effect successful, the delay should be allowed to change smoothly over time: for this reason, it has been implemented as a fractional delay, thus allowing non-integer delay values, by means of a linear interpolating function.
A more detailed explanation of this mechanism can be found in the next chapter, along with an overall explanation of the code.

# 3 Method

## 3.1 Overview of the GUI

A rather simple and yet effective interface has been implemented in order to allow the user to customize the effect.
As shown in Fig. 1, the 5 chosen parameters can be controlled and modified through five "horseshoe shaped" sliders. Click and drag the dot to the left to lower the value of the selected parameter, and drag to the right to increase it.
The full building procedure, in terms of code, can be found in the PluginEditor files.




## 3.2 Code implementation

A quick overview of the variables and parameters employed, other than the linear interpolating function, is provided by the PluginProcessor header, while the C++ file contains the whole operating code:

Firstly, both the parameters of the knob and the effect parameters have been set up:

As regards the knobs we have:

- Dry/Wet: controls the effectiveness of the flanger effect and goes from a minimum of 0.0 (basically no effect applied), to a maximum of 1.0, default value is 0.5
- Depth: makes the effect more/less clearly discernible (min: 0.0, max: 1.0, default: 0.5)
- Rate: controls the speed of the LFO, thus the rate of the oscillation (min: 0.1, max: 20.0, default: 0.5)
- Feedback: handles the amount of feedback (min: 0.0, max: 0.99, default: 0.5)
- Amplitude: sets the amplitude of the LFO sinusoid (min: 0.01, max: 3, default: 1)

While among the parameters needed to implement the flanger we find: the phase of the LFO, the delay time (which is a float number), differentiated variables for the two channels (left and right) since we work in parallel on both, "placeholders" for when it comes to read from the input signal and when it comes to copy portions of it on the buffer employed to create the delay line.

In the "prepareToPlay" section, the initial delay time and the initial phase of the LFO are initialized (respectively to 1 and 0), and both the circular buffers (left and right) for the delay line are set up to a sample length equivalent to the maximum chosen delay (2 seconds): since the length of the delay changes with the phase of the LFO, the buffer is preallocated to be large enough to accommodate the maximum amount of delay.

The processBlock is where the audio input ("buffer") is acquired and the flanger is effectively implemented.
While iterating through each sample of the buffer, a sinusoidal low-frequency oscillator is generated, whose key parameters we here recollect to be editable by the user. The LFO is the core element of a flanger: its characteristic sound comes from the motion of regularly spaced notches (points of destructive interference between the signal and the LFO wave) in the frequency response, and for this reason it is critical that the length of the delay changes over time.

So indeed, the delay time is varied using a low-frequency oscillator. This is also why the domain of the LFO sinusoid has been remapped into a different domain, to make it suitable for operating on the delay time and making it varying among reasonable values.

The effect is then actually implemented by means of the previously defined circular buffers.
The actual length of delay at any time is controlled by the distance between the read pointer and write pointer in the buffers: while the latter moves at a constant speed, advancing one sample in the buffer for each input sample, the former moves progressively slower, increasing the delay.

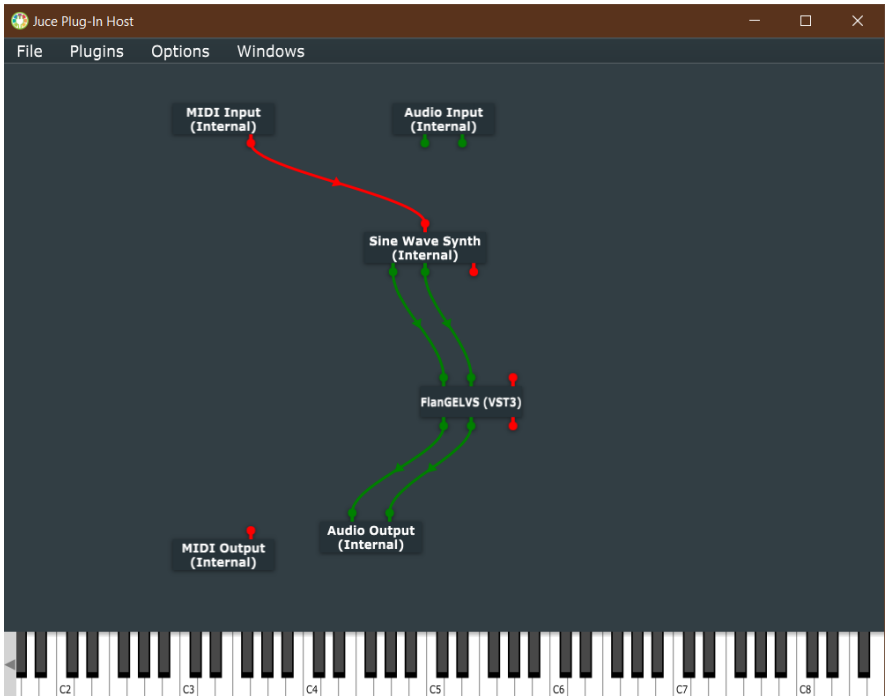
As mentioned above, in order to allow the delay to change smoothly, a linear interpolating function has been implemented with the aim of performing a fractional delay.
The interpolator basically draws a line between neighbouring samples and returns the appropriate value on that line: the fractional part of the delay (readHeadRemainderFloat) determines how far to go along this "line" between samples.
Inside the function, the fractional delay, as been defined as:

```
/*==============================================================================*/
/* Function: Linear Interpolation  */
float linear_interp(float sample_x, float sample_x1, float inPhase) {
    return (1 - inPhase) * sample_x + inPhase * sample_x1;
}
/*==============================================================================*/
```

where delta is the fractional part of the delay.
The eventual amount of feedback is then updated, along with the write placeholder/pointer.
Finally, the left and right output samples are generated, combining the original samples and the processed one through the Dry/Wet editable parameter (a sort of gain value for the delay).

# 4 Results

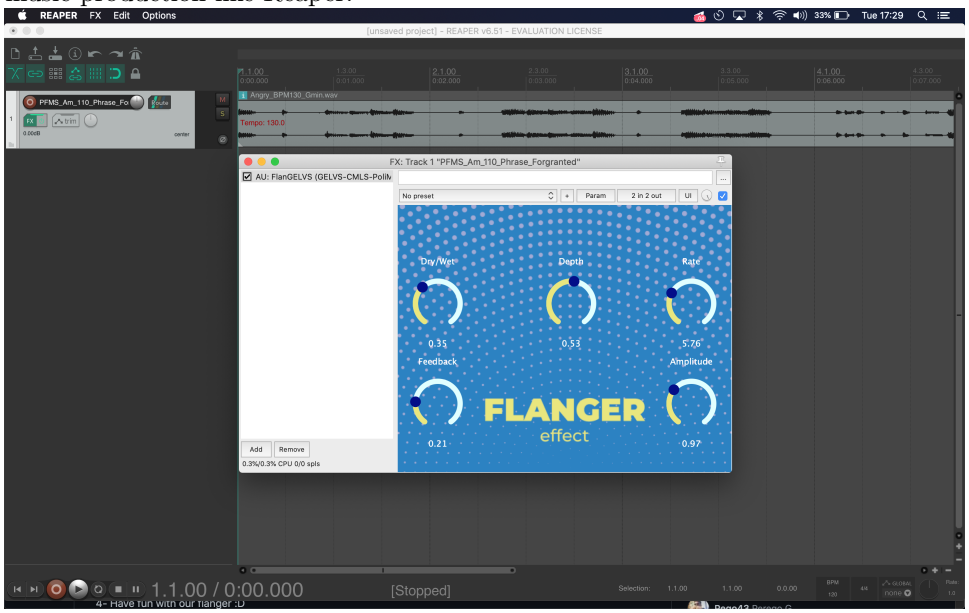
Summarizing, a flanger effect plugin with feedback has been implemented.
The plugin has been developed in Juce through Projucer and can be built with Visual Studio (Windows) or XCode (MacOS).
To assess the effectiveness of the plugin it can be either tested directly through the Juce Plug-In Host or simulating the input from an instrument with a VST3.




The current version has a clean user interface that provides 5 slider with which as many parameters can be modified also in real-time:

- Dry/Wet: tells "how much" effect apply
- Depth: makes the effect more/less discernible
- Rate: controls the speed of the LFO
- Feedback: handles the amount of feedback
- Amplitude: sets the amplitude of the LFO, it increases the depth value trying to reproduce a "clipping" effect.

This plugin can be used as well through more complex applications than AudioPluginHost, as software for music production like Reaper:



# 5 Conclusions and further improvements

Much of the effort behind this project lies directly on the comprehension and the mechanism behind a flanger.

On one hand it is more complicated than simple delay lines, by being time-variant unlike the standard delay, while on the other is similar to a phaser (or a chorus) since in both these effect the superposition of the signal and the LFO basically results in a comb filter.
Despite being aware of the existence of the Juce modules "dsp", it has not been employed in order to keep the task still challenging. The implementation of a flanger, then, requires some determined characteristics and yet it is highly "customizable", given the consistent amount of parameters that can be set as editable.

Among the possible future improvements which could be added, there is for sure the possibility of enlarging this parameters set, allowing for example to choose the waveform of the LFO or to invert the phase by building an Inverted Mode (trading places between the peaks and the notches in the frequency response of the comb filter).
Also, it is possible to add a slider for control the phaseOffset of the LFO sinusoid. Furthermore, as regards the interpolation to perform fractional delay, it is true that cubic interpolation requires more computational effort, but it could also give better results with respect to the linear one.