FloatFX

FX made easy for singers, instrumentalists and coding musicians.

Alberto Colucci, Davide Rizzotti, Guido Elli, Madhav Gopi

May 2024

Project github link: `https://github.com/polimi-cmls-2024/FloatFX`

1 Introduction

Singers, coding musicians, instrumentalists, sound technicians over the world face major challenges when having to control the tone of their sound/instruments in realtime. FloatFX adds an exciting flavour to live performances by enabling musicians/technicians to control various fx parameters like filters, distortion etc, with just simple motions of the hand. This would save them from the trouble of looking for knobs on their pedals in the dark, having to communicate between the stage and the console area, and plus, it also makes for a cool trick for the audience. Finger-pinch locks avoid accidental parameter changes while playing.

The effects processing and the GUI are implemented in 'JUCE' and 'Processing'. An Arduino board, paired with an accelerometer, is attached to the hand to capture movements. To demonstrate the viability of this product in a live music scene, we have made a live SuperCollider music performance, without using any external audio files. The tools, the approaches and the implementation are discussed in detail in the following sections.

2 Live Music in SuperCollider

2.1 SynthDef

All sounds for this live performance have been made using the inbuilt functions in SuperCollider, without the use of any external audio files. A variety of sounds (kick, hi-hat, electronic synths, arpeggiators) have been implemented as explained below:

a. Sound generation using deterministic UGens like SinOsc, LFTri, Saw and other noise generators like PinkNoise and WhiteNoise.

b. These sounds are modulated using envelopes, setting the appropriate parameters. They are used both for amplitude and frequency modulation. In the case of the kick drum, envelopes have been applied to both amplitude and frequency to shape the tone. For the synths, they are majorly used for controlling attack, sustain and release levels. Env is specified with the appropriate frequencies, times and curves for each use case, and at times, also using in connection with the Standard Shape Envelope Creation Methods (Env.perc, Env.linen).

c. For the electronic synth sounds, a detune has also been applied, which is achieved using the following formula:

$$detuned\_signal = original\_frequency * 2^{(}((detune\_value)/12) * [-1, 1] * number\_of\_voices)$$

The [-1, 1] range is applied to detune positively or negatively from the original frequency and achieved using the sum3rand function. As this contains a number of voices, a panner UGen namely Splay is used to spread the array of channels across the stereo field.

d. Further sound shaping is done using filter UGens such as LPF, HPF, RLPF.

e. Almost all of the adjustable parameters have been declared as arguments, eg: \freq.kr(1000). This enables to adjust these parameters during the live performance.

f. These instruments are saved using SynthDef namely kick, hats, saw, sin, tri, pul.

2.2 ProxySpace

Real-time music performance in SuperCollider is implemented using ProxySpace. It is a powerful feature that allows for flexible and dynamic creation, manipulation, and routing of audio signals. It is part of the JITLib (Just In Time programming library) and enables a high degree of real-time interaction with sound processes by leveraging the dynamic nature of proxies.

It helps us synchronise playbacks with the same tempo and starting points of various instruments, fade times, dynamic modification of synth parameters, all while groups of audio playback and process management are taken care of by SuperCollider. It can be initalised with a simple single line command as follows:

$$p = ProxySpace(s, clock : TempoClock(100/60)).fadeTime_{(}3).quant_{(}4).push;$$

After this command is run, all global variables further declared will be a NodeProxy object, meaning, we can leverage the functions mentioned above instantly without any further configuration.

2.3 Patterns

Patterns in SuperCollider are used to generate sequences of values or events over time. Various patterns have been used for the live performance implementation, most importantly Pbind, which helps us change values dynamically with just giving pairs of variable name and values, which is similar to the .set function. Other patterns such as Pseq, Pxrand, Pwhite etc. have been used to select notes, change duration of notes, configure attack, release times and more.

Each instrument can be played, modified (parametrically by changing the values mentioned in the Pbind value pairs or by modifying within the SynthDef itself) or stopped individually. Due to the randomness of patterns like Pexprand, each playback would yield a different sonic experience, even without changing any parameters.

2.4 Connection to JUCE framework

To route the audio from SuperCollider to JUCE, VB-CABLE Virtual Audio Device (`https://vb-audio.com/Cable/`) has been used.

3 Physical and electronic implementation

3.1 User interface

The human interface for this device is a glove, provided with an accelerometer and metal tips on the thumb and index fingers.
The hand rotation changes up to two selectable parameters, that can be locked with a fingertip pinch.

3.2 Modules

3.2.1 Accelerometer

The accelerometer is mounted on a PCB from DF-Robot. It uses the MMA7361 IC. This kind of sensor is a capacitive micromachined accelerometer. Internally, there are multiple membranes and their relative movement alters the devices' capacity. A variation in the output voltage seen at the three X,Y and Z pins. The voltage can oscillate in a range between 0 and 3,3 volts.
The MMA7361 IC can detect accelerations of $\pm$1.5Gs or $\pm$6Gs, based on the mode it is set to by grounding a pin. For this experiment, it was set to $\pm$1.5Gs sensibility. Using this resolution, an acceleration of 1G leads to a voltage variation of 800mV.

The voltage variation follows the acceleration direction, sign included. Arduino board

3.2.2 Arduino

The Arduino board used is a MKR WiFi 1010. It has a 12-bit DAC, but the Arduino environment uses 10-bit resolution by default, giving 1023 steps. This leads to a sensibility of 4,9mV for each increment. When resting, the accelerometer detects 1G on the positive Z axis, leading to an output voltage of 2,45V. The ADC is supposed to read 501 / 1023. The other two axis are not being subjected to any acceleration, leading to an ADC value of 338 / 1023.

3.3 Algorithm

At startup, there is a calibration phase where the hypothetical values of 501 and 338 are overwritten by the real module values, adapting to the user's hand resting position and manufacturing variability of the accelerometer.
Every axis voltage is read continuously and mathematically low pass filtered by means of a moving average with a window size of 10 samples. Three 'deltas' are computed to know how big is the rotation with respect to the resting position during the calibration phase.

3.3.1 Set & lock

When the user wants to change a parameter, he has to rotate his hand and pinch his fingertips to lock the value in place, then he can go back to the resting position and set another parameter for the other axis. Unlocking the parameters is done by repositioning to the previous position and pinching again. This kind of behaviour allows a musician to play without worrying about involountary parameter changes. If the rotation affects more than one axis over the set threshold, a decision is made, based on which axis is subjected to the biggest variation. This avoids changing two parameters at once if the hand is slightly tilted into another direction.

3.3.2 Control linearization

The output values are linked to the accelerometer ones by means of a map. This approach gives total freedom and allows to strongly linearize the control behaviour. Since the accelerometer is subjected to gravity, the vertical component is the prominent one, which changes as $\sin(\theta)$.
With linear value spacing this would lead to a higher sensibility until 30 degrees of rotation, then it would loose it quickly.
By mapping the values to a sinusoidal function in the interval between the threshold and the maximum readable delta, the control behaves smoothly and predictably.

4 FloatFX overview

FloatFX features three different effects - Equalization, Distortion and Delay - with multiple parameters. It features a gain slider to adjust the overall volume and a mapping mechanism to control the parameters using hand movements. The GUI is shown in Figure 1

Figure 1: FloatFX GUI

4.1 Effects

4.1.1 Equalization

The Equalization part of the plugin allows the user to modify the frequency spectrum of the input sound by using the following parameters:

- Frequency Cutoff Slider: This can be used to modify the cutoff frequency of the filter, which is ranging from 20 Hz to 20000 Hz to cover all the audible frequencies.

- Q factor: This can be used to modify the Q factor of the filter. It can creates interesting resonant effects used in combination with the frequency cutoff.

Moreover, we allow the user to switch between three different types of filters:

- Low Pass filter
- High Pass filter
- Band Pass filter

4.1.2 Delay

The delay panel is simple yet powerful to create an interesting and trippy effect on the output sound. It features two main sliders:

- Feedback slider: This allows the user to modify how much feedback hw wants to apply to the original sound.
- Delay time slider: This can be used to set the time in milliseconds between the repetitions of the original sound

4.1.3 Distortion

The distortion panel allows the user to enrich the input sound by applying non linear transformations to it. It features six different sliders:

- Drive slider: this is the main slider, which allows to control how much distortion we want to apply on the input sound
- Dry/Wet slider: this is used to control on how much of the input signal we want to apply the distortion on. A value of 0.5 means that half of the signal is unchanged, and half of the signal is distorted.
- Low pass filter and High pass filter sliders: This filters apply to the wet distorted signal so that we can control its range of frequencies.
- Volume slider: controls the volume of the distorted signal
- Anger: This slider controls how crunchy and scratched the output distorted sound is.

4.2 Controlling the parameters

The user can map every parameter to the X axis or Y axis movements of the accelerometer. To do this it is simply necessary to press on either the "MAP X" or "MAP Y" buttons and then press on the grey button near the parameter we want to control. Once a parameter is mapped, it will assume the same color of the relative axis, as shown in Figure 2. After the mapping is done, the user can control the parameters with the movements of his hand giving an unique feeling to the control of music.

Figure 2: Mapping the controls

4.3 Technical Notes

4.3.1 Code Structure

Alongside with the usual *PluginProcessor.h/cpp* and *PluginEditor.h/cpp* files, we decided to decouple the effects in their respective header files. The computation of the spectrum for the processing part is done aswell in separated files.

4.3.2 Modules

In the github repository it is shown the list of modules require to compile and build FloatFX. The *juce_serialport* (`https://github.com/cpr2323/juce_serialport`) module is used to handle the connection with Arduino and receiving it's messages.

5 Spectrum visualizer in Processing

A spectrum visualizer has been implemented using Processing in a Java environment.

5.1 OSC Communication

An array of floating point values, that represent the magnitude of the spectrum beans, is transmitted through OSC messages from the plugin to Processing. The

message is decoded and the visualizer is drawn. The libraries "OscP5" and "NetP5" are used to implement the OSC receiver over the local network.

5.2 Spectrum bars

A series of bars is drawn according to the values received. The height represent the magnitude of the spectral bean and the horizontal position represents the frequency. These values are displayed in a logarithmic scale and the width of the bars changes accordingly.

5.3 Particle system

At every drawing iteration a particle system is created at the top of each bar. The particles are subjected to a force pointing downwards, plus some random values, in order to have a fading trace of the spectral envelope.

5.4 Customization

There are some parameters that can be changed, such as:

- number of spectral beans (related to the FFT size in the plugin)
- minimum value in dB in order to draw a bar
- correction factor to reduce low frequencies and enhance high frequencies
- particles' settings (number of particles, force and lifespan)
- smoothness (up and down)
- OSC settings (IP address and port)
- colors (background, bars and particles)