POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Computer Music : Language and Systems

Project : Music Instrument Interface for Guitar

Djavan Borius, Dario Sorce, Salvatore Pantusa, Gianmarco Parente & Julien Boidevaix

May 27 2024

Table of Contents

Introduction

As of today, music production has reached its greatest accessibility levels yet. With the ever-increasing performances of our personal computers and the popularization of DAWs and audio software, professionals and amateurs musicians are playing and producing music more than ever. With the help of this technical evolution and the impact of Internet and social medias, new ways of music-making keeps emerging. As musicians and guitar players, we wanted to bring accessible new ways of playing our instrument in computer-assisted music production, so we decided to make a *music instrument interface* for guitar. Not unlike the MIDI Process that is pretty common in music production with keyboards, our goal was to code a way to transform in real-time the audio signal of an electric guitar plugged in a computer into a completely different sounds that could mimic instruments or synthesizers. As a guitar behave quite differently than a keyboard while playing it, it opens some musical possibilities and could inspire musicians to experiment with it while sounding totally differently, or use its various effect to bring new types of synthesized sounds in their recordings or live performances. This report will explains how we built the different components of our system (Interaction system unit, Computer Music unit, Graphical feedback unit), and how we can gather all of them in order to have our own and special sound for guitar.

1 Description of the Project

The project requires the combination of several aspects.

First we need to collect the sound of the guitar. The *Interaction System Unit* is composed of a guitar connected to the computer by an audio interface.

Then, in our *Computer System Unit*, the signal is processed through several stages. In order to change the sound of the guitar we need a proper SynthDef plugin that plays the same note but with a different timbre. This means that we need to collect the fundamental frequency f_0 and the power amplitude A_0 of the note played by the guitar. This is done via the Fast Fourier Transform (FFT) implemented in JUCE (See Here). After, thanks to the OSCsender function we can send f_0 and A_0 to SuperCollider. The SynthDef defined in SuperCollider can then play the sound we want when a guitar note is played. To enhance the possibilities of making a new and original sound we put the sound of the SynthDef through 5 different plugin effects (See Here) :

- Reverb
- Space Echo effect
- Chorus effect
- Distortion

Finally a *Graphical User Interface* (GUI) is implemented in SuperCollider to control the initial values of the plugin effects. The parameters that controls the effects are changed after we launch the graphical unit. For that, we coded in Processing a bouncing square program which sends instructions to SuperCollider to changes the effects' parameters (See Here).

The aim of the project is summarized in the following scheme:

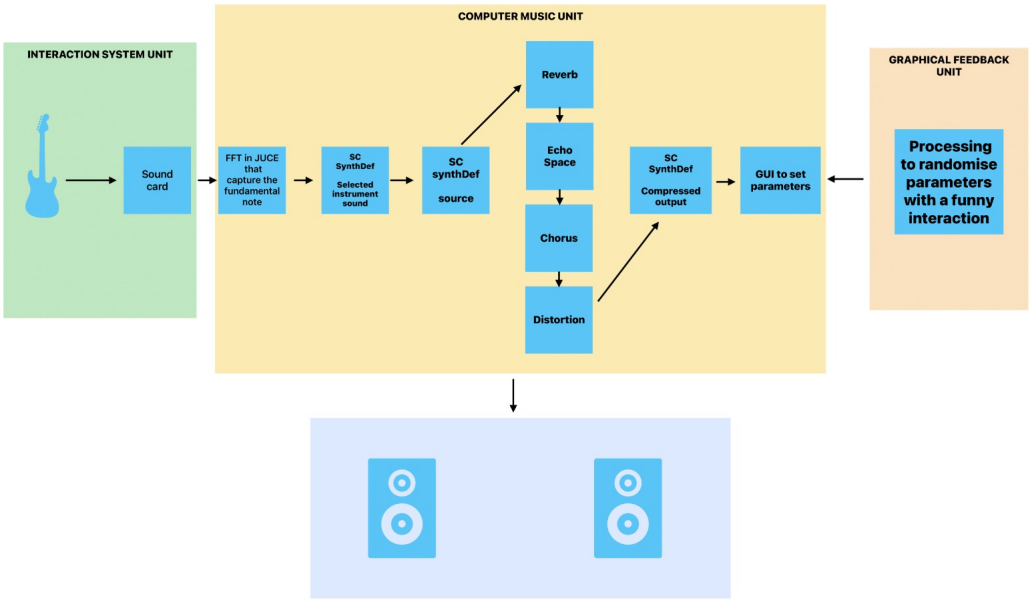

Figure 1: Scheme of the project

2 Connecting the guitar to the computer and JUCE

For collecting the guitar's signal we are using an audio interface connected through USB ports to a computer. The type or brand of audio interface used isn't important, in our case we tested and used two different kinds of audio interface that can be found easily for a fair price. The audio interface we used were the Scarlett focusrite 3^{th} generation (See Here) and the M-Audio M-Track DUO (See Here). The guitar is connected by a standardized male to male jack cord to the interface, which is connected to the computer by USB ports. The audio interface hardware process the signal coming in at a sampling frequency $F_s = 48\ 000$ Hz (in the focusrite hardware we can change the sampling frequency but we choose to work at F_s). We setup JUCE to consider the audio input as the audio interface used, and we can make the Time Musical note recognition algorithm work.

3 Real Time Musical note recognition

In order to play different instruments or sounds with our guitar, we need first to recognize the note that is being played. While it is being far from easy, we managed to get it working. But the difficulties we encountered are proof that it's still an active area of research. Since this project aims to do note recognition in real time and we want to have a working prototype that could be expanded further in the future, we only kept the basic elements and recognize one note played at the time (Monophonic note recognition).

3.1 The Fast Fourier Transform of a note

A note is characterized by a fundamental frequency in the audio bandwith 20 Hz to 20 kHz. But since we play an instrument (in our case a guitar) a note is the combination of several frequencies in addition to the fundamental one. We can analyse them by looking at the

spectrum of the signal thanks to the FFT (Fast Fourier Transform). As we can see in figure 2 several peaks compose the spectrum, we have the fundamental at $f_0 = 98$ Hz, the harmonics and a background noise at high frequencies.

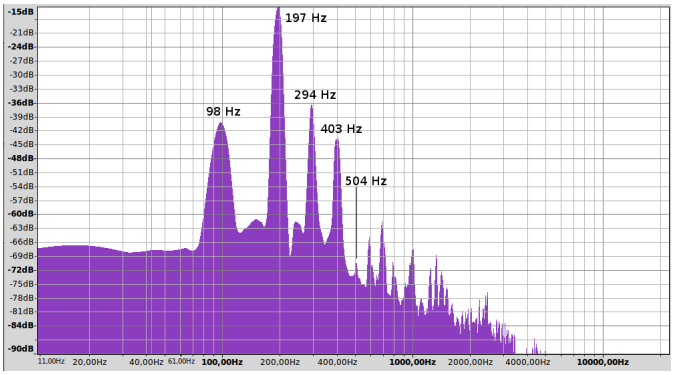

Figure 2: Spectrum of the G string of a classic Guitar [4]

Also, from the figure 2 we see that if a harmonic is played then this harmonic will have a higher amplitude. In consequence in order to extract the frequency played by the guitar we can apply this function :

$$f = \max_i T[i] \times \Delta f \tag{1}$$

Where T is the absolute value of the array computed by the FFT (since the FFT gives us an array of complex values $T = \text{abs}(FFT)$, $\Delta f = \frac{F_s}{N}$ ($F_s = 48000$ Hz is the sampling frequency of our audio card, and N is the size of the FFT) is the frequency resolution, i.e every index in the FFT array is spaced by Δf. We can collect the maximum amplitude of the FFT $\max(T)$ that can defines the intensity at which a note is played. Finally we can send these two information to SuperCollider with the JUCE function `juce::OSCsender`.

3.2 Compute real time FFT

As stated previously, the main difficulty in this project is to work with a signal in real-time. This means that the FTT must be adapted accordingly. To maximize the efficiency of our note recognition process we need a low Δf meaning a longer size for the result of the FFT. When usually the size $N = 1024$ we have 4096 samples for our Fourier transform.

Furthermore, normally we are given a complete signal that we separate in several frames on which we apply the Fourier Transform, as in figure 3. But since we do real time recognition we don't have a complete signal. Thus, we need to have a circular buffer, also known as "first-in first-out" queue (FIFO). Before we can do any FFT processing, we first need to collect enough samples to fill up the next FFT frame. So we wait for the FIFO to be full of input data from the signal. Process the FFT on this current frame, find the fundamental frequency of this frame and simply overwrite the previous contents of the FIFO to fill up the next frame, and so on (more details : [3]).

The performances of our real time recognition are quite good since for whistle the tuner give us fa^6# the is equal to 1479 Hz whereas ou FFT finds 1364 Hz. That is 7% of error. For lower notes like a do^5 at 523 Hz we find 517 Hz, so less that's 2% error.

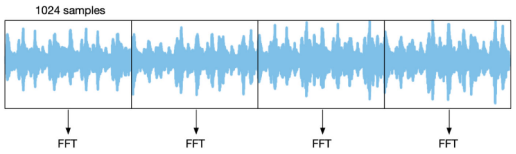

Figure 3: Waveform signal and Windowing [3]

4 Our Plugins

Our project combines real-time audio input, digital signal processing, and an intuitive graphical user interface to create a flexible audio processing system. Each of the five plugins (Reverb, Chorus, Space Echo, Distortion, Compressor) provides effects that can be manipulated dynamically, offering a broad palette of sounds for creative exploration. The integration of SuperCollider and Processing allows for complex sound design and real-time interaction, making this project a good tool for musicians and sound engineers alike.

4.1 Plugin 1: Reverb

Purpose: Adds depth and space to the sound, simulating the effect of sound reflections in a physical space. Technical Details:

- Pre-delay: Initial delay before the reverb effect starts.
- Decay Time: Duration for the reverb to fade out.
- Damp: Controls the high-frequency damping of the reverb tail.
- Comb and All-pass Filters: Used to create the characteristic reverb sound by simulating multiple reflections.

Implementation: The reverb is implemented by combining Comb and All-pass filters. The input signal is first passed through two Comb filters that simulate multiple reflections. Subsequently, the signal is further processed by a series of four All-pass filters, creating additional diffusion and density. Finally, the processed signal is multiplied by the damping factor and sent to the output.

4.2 Plugin 2: Chorus

Purpose: Thickens the sound by simulating multiple voices or instruments playing in unison with slight variations. Technical Details:

- LFO (Low-Frequency Oscillator): Modulates the delay time, creating a detuned effect.
- Depth and Rate: Control the intensity and speed of the modulation.
- Blend: Mixes the dry and wet signals.

Implementation: The chorus uses an LFO to modulate the delay times of two separate delay signals. These delayed signals are then mixed with the original signal based on the blend parameter. Optionally, the mixed signal can be further processed by a band-pass filter or an all-pass filter.

4.3 Plugin 3: Space Echo

Purpose: Creates a vintage tape echo effect with a distinct character and modulation. Technical Details:

- Echo: Controls the delay time.
- Intensity: Controls the feedback level.
- Repeat Rate: Adjusts the speed of the repeats.

Implementation: The space echo is implemented using a Comb filter that introduces a variable delay and feedback to create the echo effect. The delay time and feedback intensity can be modulated to achieve variations in the effect.

4.4 Plugin 4: Distortion

Purpose: Adds harmonic distortion and saturation, enhancing the sound's warmth and aggressiveness. Technical Details:

- Drive: Adjusts the amount of distortion.
- Tone: Controls the frequency content of the distorted signal.
- Level: Sets the output level of the effect.

Implementation: The input signal is first passed through a high-pass filter to remove unwanted low frequencies. Next, the signal is amplified based on the drive parameter and then clipped to create the distortion. The distorted signal is then filtered again to control the frequency content and sent to the output.

4.5 Output

A final compressor is applied to the combined output of all other plugins. The processed signals from these effects are combined and sent to a bus. The combined signal is then routed through a compressor defined in the out SynthDef. This compressor applies dynamic range compression to the entire output, ensuring that the final audio signal remains balanced and within desired amplitude limits. The purpose of this setup is to apply a final stage of processing that smooths out the overall output, preventing any peaks and ensuring a cohesive sound.

5 GUI Description for Audio Effects Control Panel

The graphical user interface (GUI) is designed to provide intuitive control over the various audio effects. Each section of the GUI corresponds to one of the effects, allowing users to adjust parameters through knobs and sliders. The design follows a clean, organized layout with distinct color schemes for each effect section.

Reverb :

- Predelay Time Knob: Adjusts the reverb predelay time, range from 0.08 to 0.3 seconds.
- Decay Time Knob: Controls the reverb decay time, range from 0.3 to 10 seconds.
- Damp Knob: Modulates the damping effect, range from 0 to 1.

- Level Slider: Sets the reverb mix level, range from 0.0001 to 1.

Chorus :

- Depth Knob: Adjusts the chorus depth, range from 0 to 0.02.
- Blend Knob: Controls the blend between dry and wet signals, range from 0 to 1.
- Rate Knob: Modulates the chorus rate, range from 0.05 to 10 Hz.
- Level Slider: Sets the chorus mix level, range from 0.0001 to 1.

Space Echo :

- Echo Knob: Adjusts the echo time, range from 0.1 to 2 seconds.
- Rate Knob: Controls the repeat rate, range from 0.1 to 5 Hz.
- Level Slider: Sets the echo mix level, range from 0.0001 to 1.

Distorsion :

- Drive Knob: Adjusts the distortion drive, range from 0 to 5.
- Tone Knob: Controls the tone of the distortion, range from 0 to 5.
- Level Slider: Sets the distortion mix level, range from 0.0001 to 0.80.

Figure 4: GUI

5.1 Functionality

Each control is linked to a corresponding parameter in the synthesizer's signal chain. Adjustments made via the knobs and sliders are immediately reflected in the audio output, allowing for real-time tweaking of the effects.

The GUI's color themes are matched with the squares in the Processing sketch, which move across the screen and change direction when they hit the edges. Each square corresponds to an audio effect and sends OSC (Open Sound Control) messages to update effect parameters when a square hits the boundary. The colors used in both the GUI and the Processing sketch are respectively Electric Lime, Blizzard Blue, Atomic Tangerine, Fuzzy Wuzzy.

6 Processing

In the processing part we wanted an artistic and interactive program. The main idea of this segment is to enhance our imagination and our artistic skills. In order to make so, we have n moving squares of different colors that controls the n effects (in our project we have $n = 4$ effects) we developed in SuperCollider. Once they hit the border of the window `Processing.exe` will send the information (through OSCmessage) that the n^{th} square hit the wall and it will change randomly the parameters of the n^{th} effect associated. The squares association is depicted in the figure 5

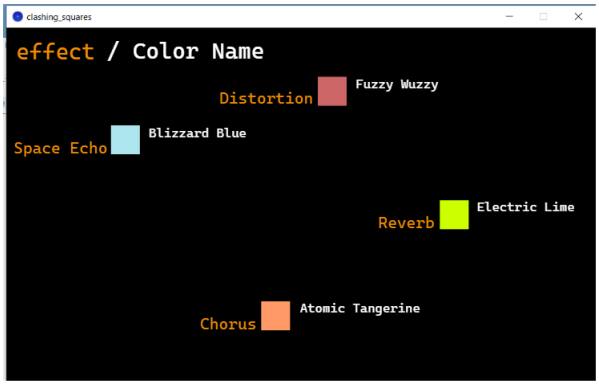

Figure 5: Colored Squares associated to one effect

Conclusion

While we are satisfied that we answered to the problematics of this project and code a functionning application, we still think there is room for improvement. As stated earlier, this type of real time application is still an area of research, with a lot of discussion around it to improve the results.

One major issue that we'd like to improve is the issue about the latency between input and output. As the communication between Juce and SuperCollider is a quirky part of the code, we think that we could improve the lag, and get a more responsive system in the future. While its depending on the amount of information Juce has to send to SuperCollider, we get a latency of approximatively 500ms between the moment a note is played on the guitar and the sound we get on the speaker.

This is mainly due because the FFT have to process in real time a lot of information. Since we chose a high number of samples to reduce potentials errors on deducting the frequency of our signal, a lot more of calculations is made, which cause delay. Knowing this, we think that with a few more time and experiments, we could fix this and get a better result. But in the meantime, this prototype fulfills our expectations.

References

[1] https://docs.juce.com/master/tutorial_spectrum_analyser.html

[2] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://zestedesavoir.com/tutoriels/pdf/3013/reconnaissance-de-notes-de-musique.pdf

[3] https://audiodev.blog/fft-processing/

[4] https://zestedesavoir.com/tutoriels/1836/physique-de-la-corde-de-guitare/

[5] https://focusrite.com/products/scarlett-2i2

[6] https://m-audio.com/m-track-duo