Politecnico di Milano

Music and Acoustic Engineering

Computer Music - Homework #1 (Super Collider)

Group: DelayLama - A.Y.: 2021-2022

∞ctaver

April 4, 2022

Group members:

Ahmed Said

Riccardo Di Bella

Juan Braun

Sofia Parrinelli

Lea Bian

Contents

1 Problem description and introduction

The aim of the project is to develop an *octaver* using the *SuperCollider* programming language. An octaver is an audio effect which mixes the input signal with a synthesised signal whose musical tone is one (or more) octave lower or higher than the original. One common way to use an octaver is to feed it a signal coming from an analogue instrument, such as a guitar or a bass. Our implementation was developed and tested exactly with this use case in mind, though it could also be used with other kind of signals (such as a human voice).

2 Features and GUI

The effect allows to mix into the final output the desired amount from these three signals:

- Dry input signal
- One octave above signal
- One octave below signal

The octaver can operates in two modes: *monophonic* or *polyphonic*.

Additionally, three knobs allows the user to control a chorus effect, which is applied only on the octave-modified signals.

The following is an in-depth description of the knobs available in the GUI and how they affect the sounds produced by the effect.

2.1 GUI

The user interface is composed of seven knobs and a switch button. As can be seen in Figure 1, the controllable parameters are:

- Octave up/Octave down: this controls the mix of the two octaves in the output of the effect. If the value of the knob is zero only the lower octave will present and, vice versa, when the knob is at its maximum only the higher one will be present. Values in between allow the user to "mix" the two signals.

- Dry/Wet: this controls the amount of original dry signal that is let through the effect unaltered. If the knob is at minimum only the original signal will be present, while if the knob is at maximum only the synthesised signals will be present.

- Monophonic/Polyphonic: this button allows the user to select which algorithm will be employed to synthesise the octave-modified signals. When the switch is off the effect acts in *Monophonic* mode, delivering a more "vintage" sound which mimics the one produced by classic analogue octaver pedals. This mode, however, is not well suited for playing chords. The *Polyphonic* mode offers a less interesting and more "pitch-shifting"-like digital sound, though it is able to handle chords.

- LPF Cutoff: when the *monophonic* mode is selected the signal is filtered by a low pass filter after the output of the octaver. This knob allows the user to select the cutoff frequency in the $200Hz - 8kHz$ range.

- Volume: this controls the overall volume of the output.

- Chorus Dry/Wet: this controls the mix of the chorus effect with the output of the octaver. If the value of the knob is zero, only the direct output of the octaver is present and when the value of the knob is maximum, only the output of the chorus is present.

- Chorus Depth: This controls the *depth* of the chorus, which is the amplitude of the oscillator which modulates the time-delay of the chorus.

- Chorus Rate: This controls the frequency of the time-delay modulator, thus altering the speed at which the time-delay changes over time.

Figure 1: Graphical User Interface

3 Implementation

The following is a basic diagram of the software architecture:

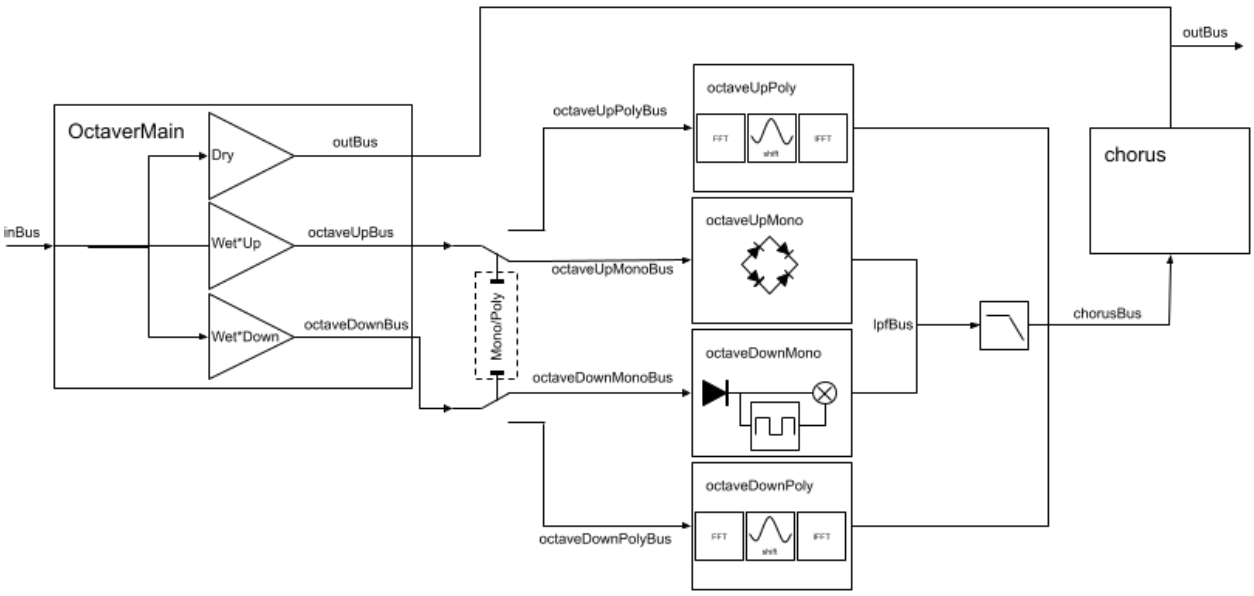


Figure 2: Block diagram of implemented octaver

3.1 SynthDefs

3.1.1 OctaverMain

This is the *SynthDef* which handles all the routing, distributing the signal coming from its input bus across all the audio busses which will be read by the effect *SynthDefs*. The routing is done according to a series of arguments, which are linked to the knob exposed in the GUI.

3.1.2 OctaveUpMonophonic

This *SynthDef* is used to receive the input signal and synthetise a signal an octave above (i.e. with a frequency that is double that of the original signal) and route it to the output bus. For the monophonic mode, the synthesis technique was based on the one employed by the first analog octaver effects, whose design was very simple. To double the frequency of the signal, full-wave rectification was performed. The Figure 3 shows the several steps performed in order to obtain the higher octave signal.

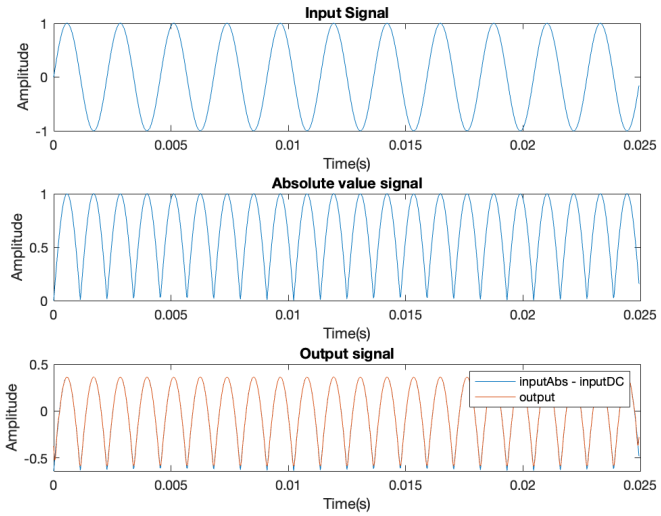


Figure 3: Step by step processing in octave up filter

In the digital domain, this simply equates to taking the absolute value of the signal. After

this, we remove the DC component introduced by the previous operation and then apply a low-pass filter to smooth out the high frequency components, introduced by the abrupt changes which appear at the end of every cycle. Obviously, the signal obtained in this way doesn't maintain the same timbre of the original, as taking the absolute value is a non-linear operation.

3.1.3 OctaveDownMonophonic

Always following the approach used by analog octavers, in this case we perform half-wave rectification of the input signal. This half rectified signal triggers a Flip Flop that changes its output every period of the original signal, resulting in a square wave of period twice as long as the original signal. The output of the Flip Flop is multiplied with the half rectified signal to obtain the lower octave signal. Figure 4 shows the several steps performed on the signal until the final lower octave signal is obtained.

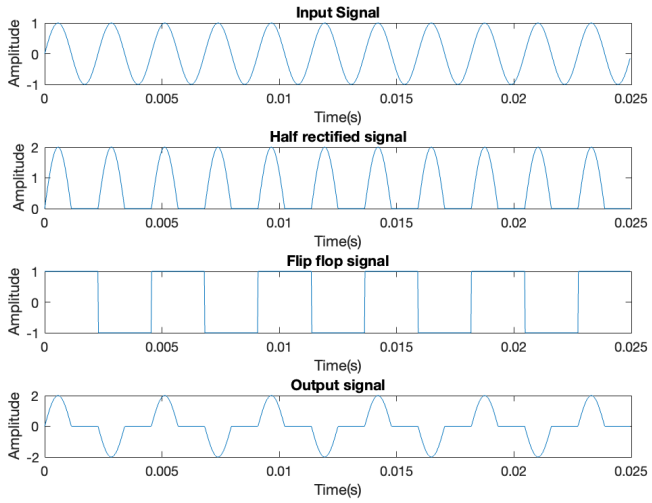


Figure 4: Step by step processing in octave down filter

3.1.4 OctaveUpPolyphonic

After implementing the analog-type octaver, we decided to implement a phase-vocoder based octaver, taking advantage of the `PV_ChainUGen` class in *SuperCollider*. To implement this, we compute the

FFT of the input signal and then apply the method `PV_MagShift` in order to stretch the magnitude of the frequency bins by a factor of 2.

3.1.5 OctaveDownPolyphonic

The idea is the same as discussed above, with the difference that now the magnitudes of the bins are stretched (or better, compressed) by a factor of 1/2.

3.1.6 Chorus

As a proof of concept we added a chorus effect that is applied to the signal after the octaver stage, as can be seen in Figure 2.

The traditional chorus effect is obtained by creating multiple copies of the original signal and modulating their delay time with a low frequency oscillator. These timing differences create the perception of a slight and gradual shift of the pitch of the signal. In *SuperCollider* this can be implemented mainly with the aid of two *UGen*:

- *DelayC*, which implements a simple delay line
- *LFPar*, a low-frequency parabolic oscillator which creates the delay-modulating signal, fed to *DelayC*

3.2 Buses

The implementation uses the following busses:

- inBus: this bus is used to route the input signal to the `OctaverMain` synth;
- octaveUpMonoBus: this bus is connected to the higher octave output bus of `OctaverMain` when the *monophonic* mode is engaged, and routes the signal to the input of `OctaverUpMono` synth;

- octaveDownMonoBus: this bus is connected to the lower octave output bus of `OctaverMain` when the *monophonic* mode is engaged, and routes the signal to the input of `OctaverDownMono` synth;

- octaveUpPolyBus: this bus is connected to the higher octave output bus of `OctaverMain` when the *polyphonic* mode is engaged. It routes the signal to the input of `OctaverUpPoly` synth;

- octaveDownPolyBus: this bus is connected to the lower octave output bus of `OctaverMain` when the *polyphonic* mode is engaged. It routes the signal to the input of `OctaverDownPoly` synth;

- lpfBus: this bus receives the outputs of `OctaverUpMono` and `OctaverDownMono` synths and routes them to the input of the `lowPass` synth;

- chorusBus: this bus receives the outputs of `OctaverUpPoly`, `OctaverDownPoly` and `lowPass` synths and routes them to the input of the `chorus` synth;

- outBus: this bus receives the dry signal output of `OctaverMain` and the output of `chorus` and sends them to the output channel of the audio device;

4 Results and final considerations

The current implementation gave satisfying results when working with a live electric guitar input signal, though there's certainly room for improvements concerning some aspects. In particular, the main problems we were faced with are:

- for the *monophonic* mode, the quality of the result depends a lot on the harmonic content of the input signal, thus it is affected by many factors which are difficult to control when dealing with an electric guitar input (e.g. pickup position, where the note is played on the fretboard...).

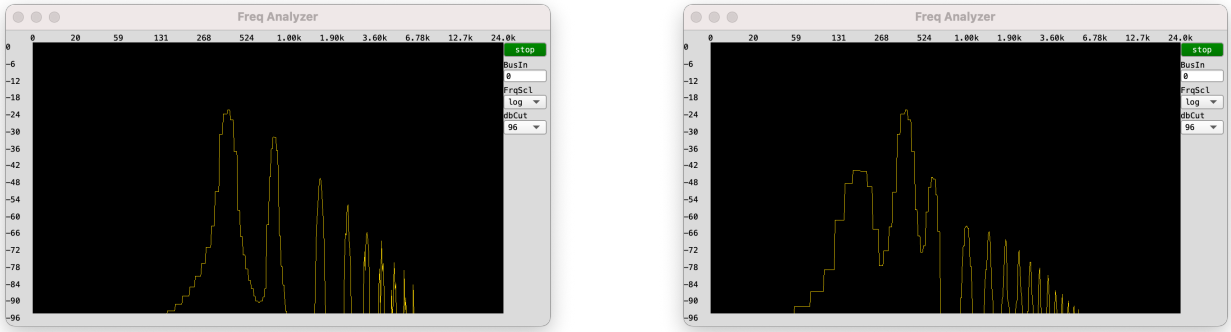
- For the *polyphonic* mode, the phase vocoder approach involves some costly operations which

can make the latency excessive for real-time applications, as it happened during our testing. As usual, the latency can be reduced by altering the parameters used to perform the FFT, though there is always a trade-off between performance and quality of the result.

In the next section we report the results we obtained by performing a spectral analysis on the signals obtained through the two approaches. For simplicity, these results show what happens when the input signal is a pure sinusoidal tone. Of course, when dealing with a signal coming from a real instrument the behaviour will be much more complex.

4.1 Spectral analysis: analog-inspired octaver

As mentioned before, this solution introduces distortion in the synthesised signals due to the non-linear behaviour of the applied operations. By looking at Figures 3 and 4, we can see that both output signals contain high frequency components due to the sharp edges of the obtained waves.

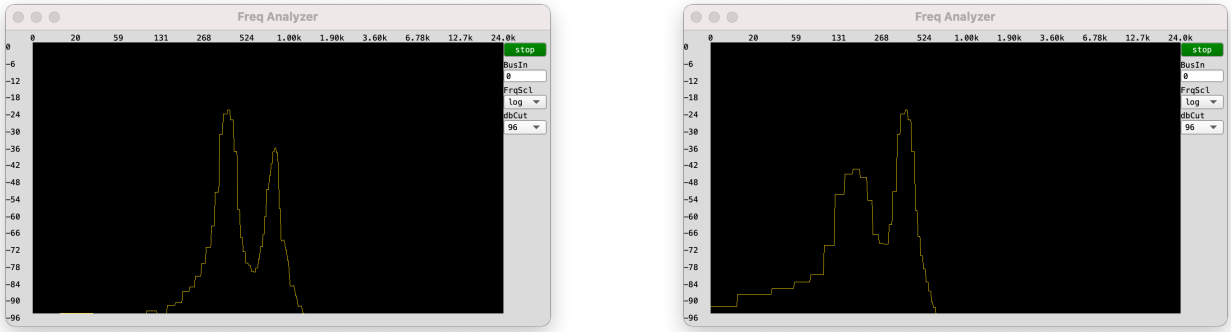


(a) Higher octave

(b) Lower octave

Figure 5: Spectrum obtained through the analog-inspired approach, using a 440 Hz sine wave as input signal

To minimize this effect, that generates an unpleasant distortion effect, we have added a user-controllable low pass filter with a cutoff frequency between $200Hz$ and $8kHz$ at the output of the monophonic octaver, as can be seen in Figure 2.



(a) Higher octave

(b) Lower octave

Figure 6: Spectrum obtained through the phase-vocoder approach, using a 440 Hz sine wave as input signal

4.2 Spectral analysis: Phase-vocoder octaver

As we discussed before, this implementation makes use of the phase-vocoder *UGens* provided by *SuperCollider*, which are used to apply transformations directly to the frequency-domain representation of the signal.

Figure 6 clearly shows that the spectrum obtained in this way is much cleaner than the one obtained using the analog-inspired approach. This generates an effect that is better suited for working with polyphonic material and when we don't want to alter too much the frequency content of the original sound.