CMLS Homework 2 – Subtractive Synthesis

Samuele Bosi, Marco Ferrè, Philip Michael Grasselli, Simone Mariani

May 22, 2020

Contents

1 Introduction

In this homework, our main task was to implement a synthesizer based on **subtractive synthesis**: when a tymbrically rich sound (usually referring to the quantity of harmonics) is involved into this procedure, you obtain an altered one, thanks to filters which attenuate defined partials.
In fact, this routine is based on two components to muster up:

- a waveform generator;
- a filter.

A GUI (Graphic User Interface) is also included, in order to both respect the request of the homework, and to have a more practical and clearer experience in understanding subtracting synthesis.

2 The Synthesizer

The following is a concise guideline on the development of the synthesizer. For a complete reference, we remind to its repository: `https://github.com/marcoferre/CMLS-HW2`.

2.1 GUI

Our plug-in, in a essential fashion, sticks to a classic sound chain of a synthesizer based on subtractive synthesis. In further detail, the visible and manageable components are thus represented in Figure 2.1:

- an **oscillator** with three different kinds of waveform available (sine wave, saw-tooth, square wave);
- a low-pass resonant IIR (Infinite Impulse Response) **filter**;
- an ADSR (Attack, Decay, Sustain, Release) **envelope generator** for the output signal.

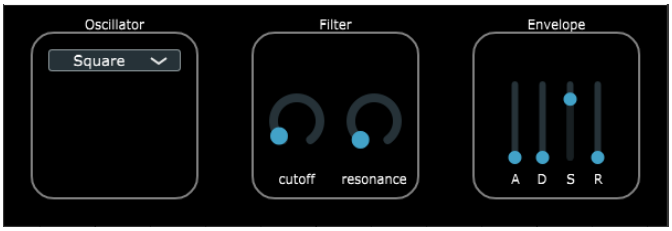


Figure 1: A screenshot of the Graphical User Interface

2.2 The Oscillator

To develop the oscillator section, the `maximilian` library is the easiest and most powerful way to implement multi-voice oscillator in the JUCE environment. In particular, our oscillator involves the presence of three different wave forms (sine wave, saw-tooth, square-wave), with an up-to-five voices overlap, so that this produces polyphonic sound. Thus, there's the chance to produce either *Pad* or *Lead* sounds.

2.3 The Filter

We focused on using a IIR filter, and this is fully implemented into JUCE in the class DSP in the following way:

```
dsp::IIR::Coefficients<float>::makeLowPass (double sampleRate, NumericType frequency,
    NumericType Q)
```

The `makeLowPass` method, in particular, returns a IIR filter with the coefficients adapted to create a **resonant low-pass filter**. In the GUI, you have its control through the frequency and resonance knobs. The choice of using a IIR filter, instead of a FIR one, is for is very light computational cost, and to take advantage of the instabilities that emerge at high resonance values.

2.4 The Envelope Generator

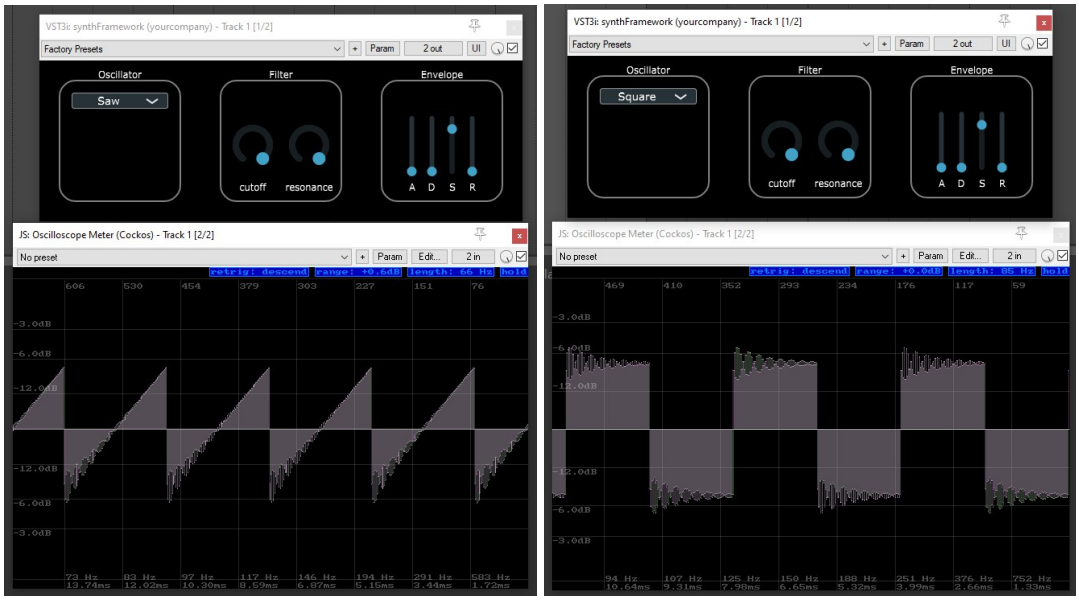
By using the same `maximilian` library mentioned previously, we added a standard envelope generator based on the **ADSR procedure**, and applied on the volume level at the output, in order to render the sound timbre more pleasantly listenable.

3 Linking GUI and Processor

The links between the GUI and the processor were implemented by applying the class `ValueTree`. Moreover, this class lets the GUI parameters be exposed to any possible DAW, to external controllers or automation.

4 Conclusions

The final result, built as a VST3 and tested in a DAW (Reaper), allows you to set up and appreciate a catchier and more useful sound. Please note that there are means of overcoming the lack of harmonic parts due to the absence of an envelope, such as considering external automation solutions, thanks to its explicit parameters.
As measured by an oscilloscope, the output signal shows up a pretty accurate waveform: the instability given by the IIR filter is appreciated whilst high values of resonance are contemplated, and with the typical oscillation close to the jump points, as shown in Figure 4.



(a) A saw-tooth wave

(b) A square wave

Figure 2: A couple of screenshots taken from the synthesizer's GUI