https://www.overleaf.com/project/626d940bd740b008a47e98f8

Politecnico di Milano

Music and Acoustic Engineering

Computer Music - Homework #2 (JUCE)

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

Fractasizer

May 11, 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 a MIDI synthesizer based on fractal systems using C++ and the JUCE framework. Many fractals can be generated by applying an iterative process, for example by repeated computations of a formula. The Mandelbrot set, for example, is the set of complex numbers c for which the function

$$f_c(z) = z^2 + c \tag{1}$$

doesn't diverge to infinity when iterated from $z = 0$. This set of points, when represented on the plane, creates a fractal image. These types of fractals, in particular, are known as *escape-time fractals*.

The idea of the project is to use the coordinates of the points, yielded by iterative applications of these formulas, to control the parameters of an oscillator.

In particular, the user will be able to select the desired fractal succession and the "starting point" c (not to be confused with the initial value of z) used in the computation, within a range of possible values.

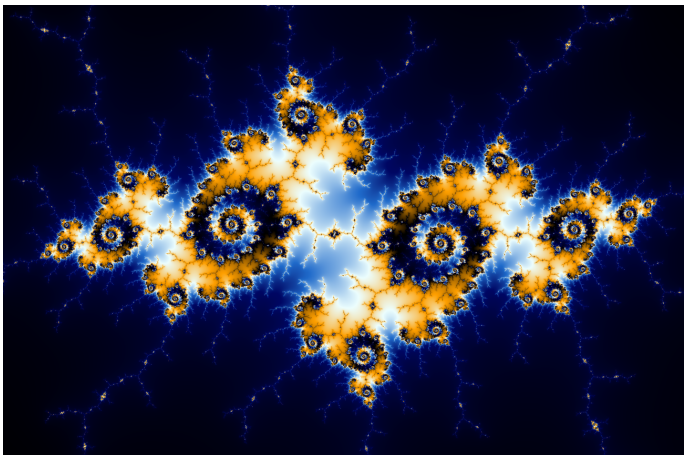

Figure 1: Mandelbrot set

2 Chosen fractals

For our project, three fractals were considered as possible choices to control the synthesizer: the *Mandelbrot* fractal, the *Burning Ship* and the *Tricorn* fractal.

2.1 Burning Ship fractal

The Burning Ship fractal is generated by iterating the function:

$$f_c(z) = (|\mathrm{Re}(z)| + i|\mathrm{Im}(z)|)^2 + c \tag{2}$$

starting from $z = 0$.

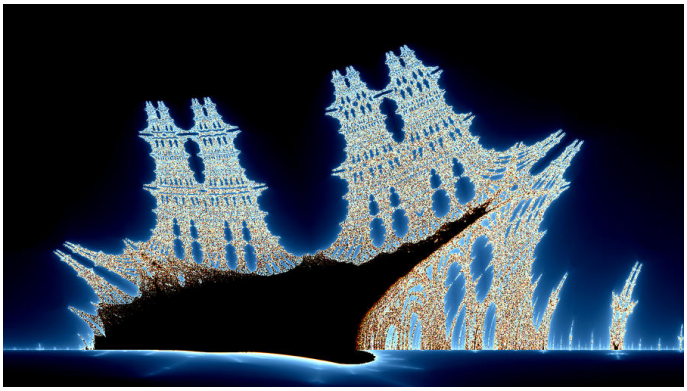

Figure 2: Burning Ship set

The difference between this calculation and that for the Mandelbrot set is that the real and imaginary components are set to their respective absolute values before squaring at each iteration.

2.2 Tricorn

The Tricorn, sometimes called the Mandelbar set, is a fractal defined in a similar way to the Mandelbrot set, but using the complex conjugate of z, so the formula becomes:

$$f_c(z) = \overline{z}^2 + c \tag{3}$$

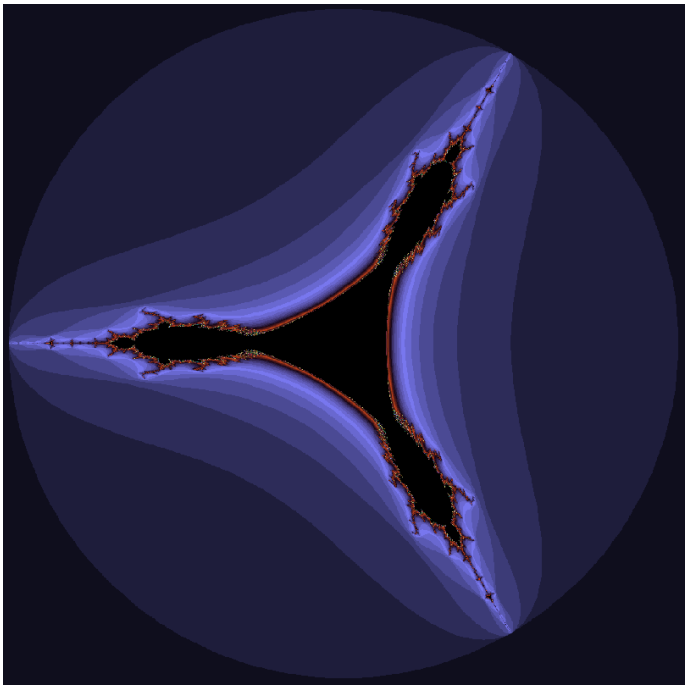

Figure 3: Tricorn set

2.3 Additive Synthesis - controlled parameters

As the final sound is generated by adding multiple sinusoidal components (having different fundamental frequencies), the plugin falls within the category of *additive synthesizers*. Depending on the relations between the frequencies of the components (also known as *partials*) we can obtain harmonic or inharmonic sounds. An even more broader definition of additive synthesis also allows more complex waveforms as invividual components, instead of just pure sine waves. Our implementation consider this broader definition, as it allows the user to choose a different waveform for each partial.

Some parameters of each of these components will be controlled by the values of the points obtained by iterating the fractal succession. In particular, we considered a total of 4 partials, so only the first 4 values of the fractal succession will be computed, starting from a specific user-provided point. In particular:

- the real part (x coordinate) of the points will control the frequencies of the partials;
- the imaginary part (y coordinate) of the points will control the rate of a tremolo effect applied on the partials.

The high level block diagram of the voice generation mechanism is shown in 4.

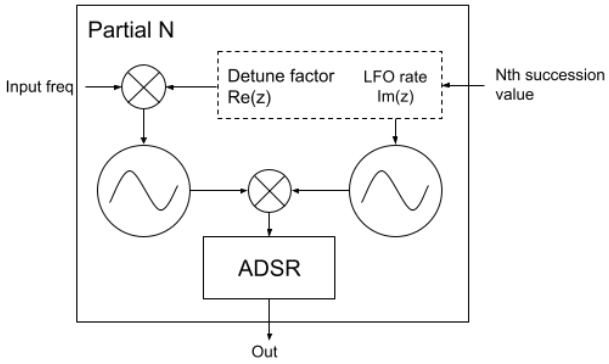

Figure 4: Voice generation

The four sound components have different amplitudes in order to give a greater relevance to the pitch. The first component, the fundamental, has an amplitude scaling factor of $\frac{1}{2}$. The remaining components have a gain weight equal to, respectively, $\frac{1}{4}$, $\frac{1}{6}$ and $\frac{1}{8}$. This is shown in figure 5.

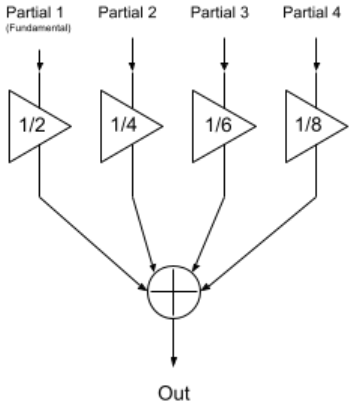

Figure 5: Additive synthesis

3 Features and GUI

The user interface area is divided into two main sections: the upper part, containing the controls for the individual 4 partials generated, and the lower part, dedicated to the fractal controls.

The upper part is further subdivided into 4 areas, one for each partial. For each partial the user can control and ADSR envelope, a combo box to choose the waveform and can visualize the resulting partial waveform in real-time.

In the lower section it is possible to select, from a combo box, the fractal that will be used as well as the starting point for the generated succession. This can be done in two ways:

- by clicking on a specific point inside a 2D input plane;
- by setting two separate sliders (one for x and the other for the y component of the point).

The following is a more thorough description of all the controls and the components available in the GUI:

- Waveform Combo Boxes: allows the user choose the partial waveform between:
  - sine wave;
  - saw wave;
  - square wave.
- ADSR knobs: 4 knobs used to control the amplitude envelope of the partial: one for the attack, one for the sustain, one for the decay and one for the release.
- Wave visualization areas: a component that allows the user to see the real-time evolution of the the generated partial.

- Fractals Combo Box: used to change the computed fractal succession between:
  - Mandelbrot;
  - Burning Ship;
  - Tricorn.
- Input plane: a plane where the user can click to select the starting point for the fractal succession computation.
- X slider: used to select the x coordinate of the starting point.
- Y slider: used to select the y coordinate of the starting point.

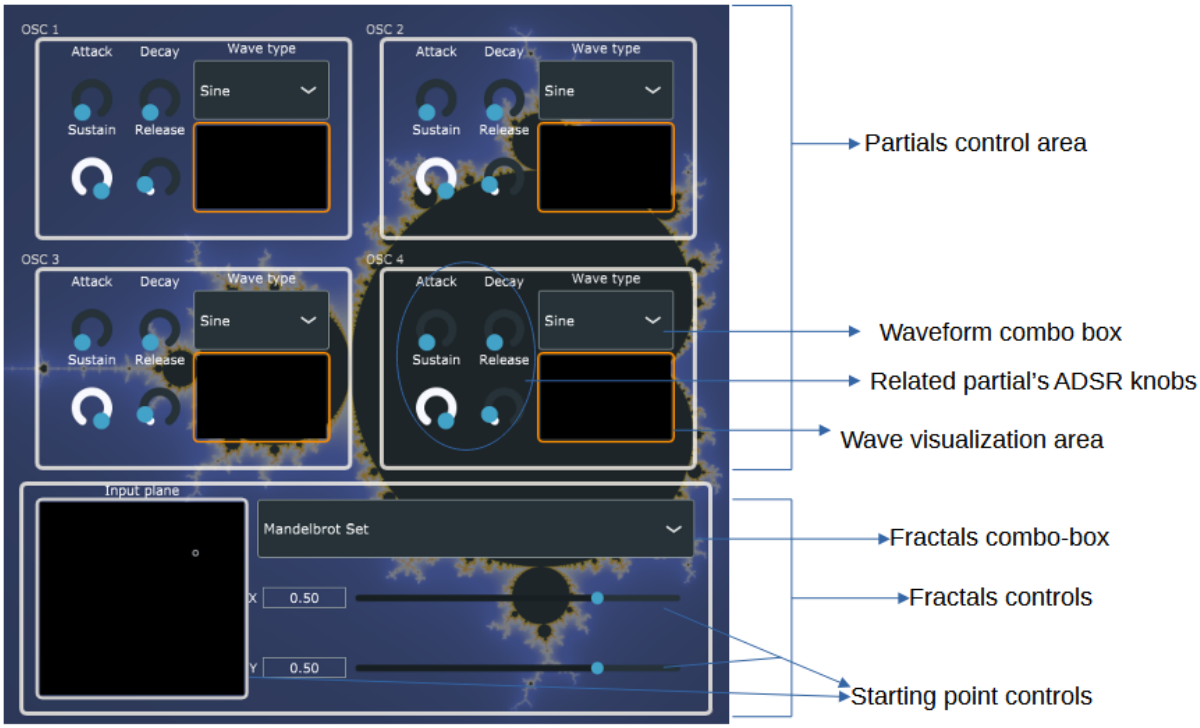

Figure 6: Graphical User Interface

4 Implementation

4.1 The JUCE *Synthesiser* class

For the implementation of the MIDI synthesiser we exploited the `Synthesiser` class provided by JUCE, which takes care of most the tasks related to handling the incoming MIDI messages. A `Synthesiser` object must be provided with one or more `SynthesiserSound` and `SynthesiserVoice` objects. The `SynthesiserSound` determines which sound must be associated to which incoming MIDI note numbers; in our case we used the same sound for all incoming MIDI notes. The number of `SynthesiserVoice` objects available determines the maximum number of polyphonies (i.e. the maximum number of notes that can be played at the same time), which has been set to 10 for our project.

4.2 The *SynthVoice* class

Our custom `SynthVoice` class, which extends the abstract JUCE `SynthesiserVoice` class, contains most of the core logic of the program, as it defines the actions to perform every time a new note has been started, stopped or a new audio output block must be generated. For each partial, this class contains:

- a dedicated `ProcessorChain`, which is made up of an `Oscillator`, a `Gain` and a `Panner` object.
- another `Oscillator` object which implements a LFO.
- an `ADSR` object (with related parameters), which controls the ADSR envelope of the waveform.

Since the LFO has a relatively low frequency compared to the frequencies of the generated audio signals, for performance reasons its next value is computed only once each 100 samples of the audio buffer.

To generate the final output, these are the main steps carried out in the `renderNextBlock`

function:

1. a local temporary buffer is set up for storing the output of each partial;
2. we retrieve a sub-portion of the buffer;
3. the sub-portion of the buffer is processed using the `ProcessorChain` of the corresponding partial;
4. the ADSR envelope is applied to the result stored in the buffer;
5. if enough samples have been processed (this is checked with a periodically re-setted counter), the LFO is applied;
6. the result of each local buffer is added to the final output audio buffer;
7. finally, we check if all the partials have finished playing (by checking their corresponding ADSR objects), and clear the current note if needed.

4.3 The *PluginProcessor* class

Inside our `PluginProcessor` we mainly set up the `Synthesiser` object, an `AudioProcessorValueTreeState` object, which contains all the parameters that can be controlled through the GUI, and the wave visualiser components. The `PluginProcessor` also takes care of re-computing the first points of the fractal succession every time the user selects a different fractal function or chooses a new starting point. The computed values are then sent to all the voices of the `Synthesiser` to update their corresponding parameters.

4.4 The *InputPlane* class

The `InputPlane` class extends the JUCE `Component` class, which is the base class extended by all the objects which can be displayed in the user-interface. Its main function is to display a rectangular

area in which the user can select a starting point for the fractal by means of a mouse click. The `InputPlane` is always linked to a pair of `Slider` components, which serve as alternative way to input the x and y coordinates of the starting point, other than to actually connect the input plane to the `PluginProcessor` parameters. The `mouseDown` function performs the range mapping from the local screen space coordinates of the displayed rectangle to the pre-defined allowed range for the coordinates of the starting point. The `paint` function performs the inverse mapping to display a circle in the position of the selected point.

The range of possible values is limited to $[-1, +1]$ both for the x and the y coordinate. This is to prevent the user from selecting starting points which can make the fractal succession quickly diverge and generate unpleasant results (or even prevent the program from working completely).

Additionally, a check is performed to prevent the user from selecting the pair $(x = 0, y = 0)$ as input point, as this would make all the values in the fractal succession equal to zero, thus leading to invalid frequencies for the partial components.

4.5 Fractal to parameters mapping

With each iteration of the fractal function the real and imaginary part of the point are computed, as the domain is complex.

The real part is directly used as a *detuning factor* for the partials, simply by multiplying it by the fundamental frequency, which depends on the incoming MIDI note number. The frequency of the first partial is left unchanged, as it defines the pitch of the note played. This means that the real part of the first computed point in the succession is actually ignored.

The imaginary part is used to control the rate (i.e. the frequency) of an LFO which controls a tremolo effect on the partials. In, particular the mapping is performed as follows:

$$r_i = \frac{|\mathrm{Im}(z_i)|}{\sum_{k=1}^{4} |\mathrm{Im}(z_k)|} \cdot 10 \qquad i = 1, 2, 3, 4 \tag{4}$$

Where r_i is the rate for the i-th partial, and z_i is the i-th value of the fractal succession. Note that, in general, we always skip the first value z_0, since for all the considered fractal it always holds $z_0 = 0$.

5 Results and final considerations

5.1 Possible improvements

Some controls commonly found in other MIDI synthesizers were not implemented due to the attention given to other aspects of the project. For example, an individual gain and pan control could be developed for each partial, as the current processor chain of each component already includes the required elements.

Though the current implementation already creates an interesting and usually pleasant sound, the result largely depends on the choice of the starting point, with even closely spaced points giving very different sounding results. This happens both because it's difficult to study the behaviour of the fractal function and because by controlling directly the frequency of the partials we alter the resulting timbre significantly. A more rigorous study of the fractal behaviour of further experimentation could lead to a better mapping between the fractal and the oscillator parameters. Another idea is to add a filter to the processor chain of each partial and use the fractal points to control its parameters (either directly or by controlling an LFO linked to one of the filter parameters). This alternative filtering idea, however, is of interest only when using partial waveforms which are not simple sine waves, and it's more akin to the framework of *subtractive synthesis* rather than additive synthesis.