COMPUTER MUSIC - LANGUAGES AND SYSTEMS

HOMEWORK N.1 - ASSIGNMENT 4

Granular synthesis for foley sounds

Professor: **Fabio Antonacci**

Academic year: 2022-2023

Group 7 - Lemonjuice:
Guglielmo Fratticioli (Matr: 214322)
Chiara Lunghi (Matr: 233195)
Alessandra Moro (Matr: 102283)
Elia Pirrello (Matr: 223050)

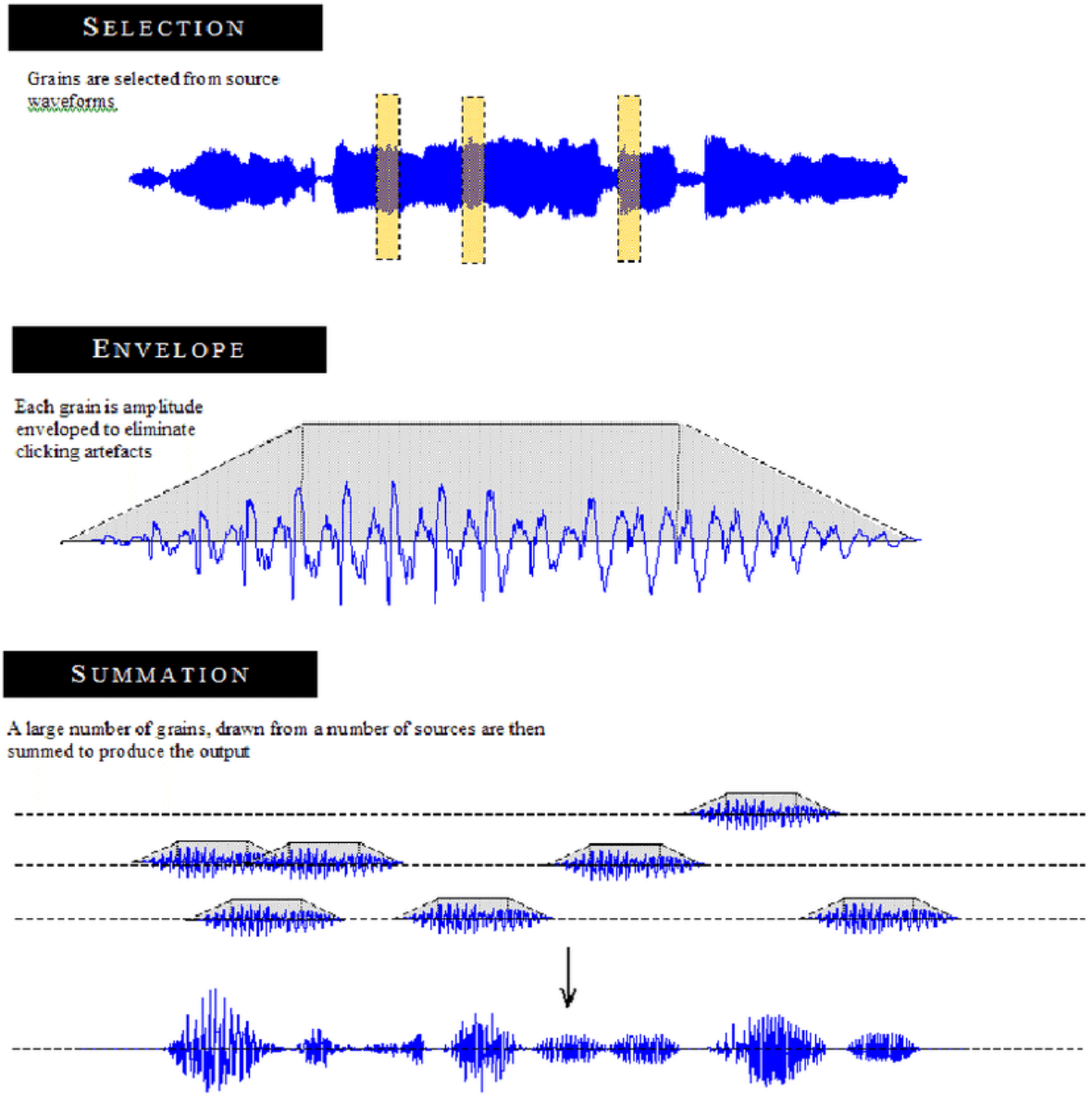
This report explores the use of granular synthesis for sound design and presents a Supercollider implementation of a granular synthesizer. The report covers the basic principles of granular synthesis, its applications in foley and soundscapes, and remarks on sound design. It also includes a detailed description of the graphical user interface and model architecture of the Supercollider implementation, as well as code snippets. Finally, example results are presented, showcasing the versatility of granular synthesis.

Contents

1 Introduction

Granular synthesis is a sound synthesis method that involves dividing a sound into small pieces or ”grains,” which can be manipulated independently and played back at a certain rate. The higher the playback rate is, the higher chance there will be of grain superimposition, resulting in a ”blurry” and ”cloudy” version of the source sound.

The synthesis process is composed by three main steps:



1.1 Applications

Granular synthesis is a powerful tool for creating realistic and complex foley sounds, such as footsteps or ambience sounds. A certain degree of casuality could be exploited in the grain selection to obtain non-repetitive results. This approach can also be used to generate immersive soundscapes from completely different audio sources.

1.2 Remarks on Sound Design

Although Granular synthesis may gives decent results in a naif implementation, its true expressive power is unleashed when the source signal is already a mix of different sources and the output is further processed with **audio effects** (such as reverbs) or actual **audio edits** (e.g. volume and pitch automations).

Varying creatively the envelope shape and the pitch of the single grains could also be an optimum way to alter the result.

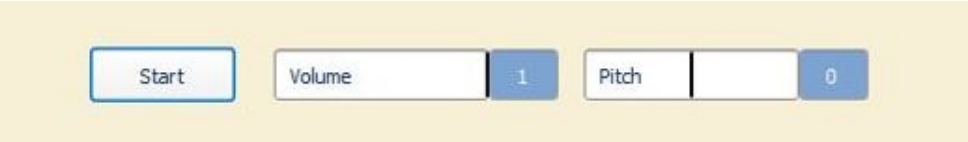
2 Supercollider implementation

The SuperCollider environment is strongly focused on live performances, facilitating the creation and management of audio flow. While the creation of the graphical interface may be more challenging compared to languages like JUCE, it still allows for the development of simple control surfaces.
We have developed three **.scd** files containing the UI builder, the *SynthDef* and a main file.
To start the synthesizer is sufficient to load the **main.scd** into supercollider and run the whole code at once.

2.1 Graphical User Interface

Defined in **gui.scd**, it is composed by three main modules that controls various parameters of the main *SynthDef* (grainSynth) with sliders, buttons and knobs.

2.1.1 Output module



- ”Start” button toggle playback of the synthesizer
- ”Volume” slider changes the output level
- ”Pitch” slider shifts the output pitch by semitones

2.1.2 Samples module

The two source sounds can be controlled by:

- A relative button, to choose the sample
- A ”Volume” slider
- A ”Pitch” slider

2.1.3 Granular parameters

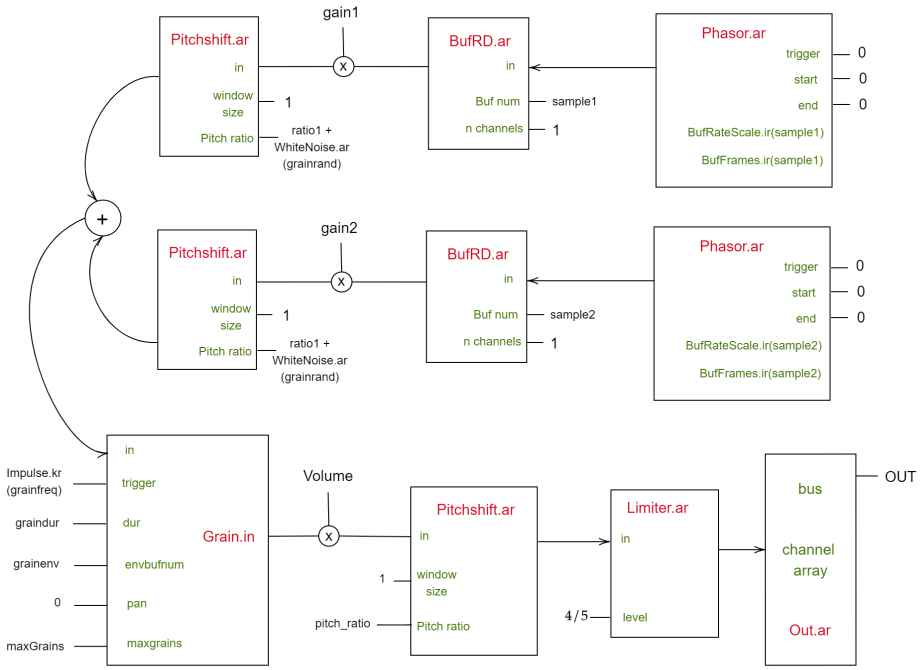
- ”Attack”, ”Sustain”, and ”Release” knobs define a triangular envelope applied to the grains amplitude
- ”Frequency” knob sets the grain rate in Hz
- ”Duration” sets the time duration of each grain in seconds
- ”Granular Drift” adds randomness to the pitch of each grain

2.1.4 Recording

- ’Record’ starts the recording of the output
- ’StopRec’ stops the recording and save the audio in ’.wav’ format into the project directory

2.2 Model Architecture

The audio generation is completely defined inside a `SynthDef` in the `core.scd` file. `out.ar()` function implements the following block Diagram :



The granulation is performed by the Supercollider's `Grain.In` function that receives as input a mix of two sample buffer readers.

To improve the randomness and obtain more creative results, we added pitch shifter effects to the buffer reader, controlled by the "Granular Drift" parameter and knob.

Moreover, to the final output we added a limiter (to prevent the signal from clipping) and a pitch shifter effect.

3 Conclusions

This group work has taught us the fundamentals of this kind of synthesis despite the fact that we didn't implement explicitly the sample Granulation algorithm. In the end the `GrainIn` function has resulted to be not as tweakable as we thought.

`GrainIn` processes the audio in the input bus in a frame by frame fashion, so we couldn't actually randomize the grains starting position inside the whole sample as other known granular synths are capable of.

However the integration of pitch shifters and a discrete number of audio samples to choose from has result in a good timbric variation in the results.

3.1 Example results

Combining different samples, at different pitch, and interacting with the synth parameters during the recording, we created five different sounds as examples of what a granular synthesizer is capable of:

- Foot steps
- Hail on the window in a stormy night
- Helicopter noise
- Creepy sounds of a Horror Factory
- Sound of a Mining Cart traveling in a rocky mine

4 Code Snippets

- In Supercollider language the GrainIn function requires the following syntax :

```
GrainIn.ar(
    nchannels: 2,
    trigger: Impulse.kr(freq: grainfreq),
    dur: graindur,
    in: ... ,
    maxGrains: maxGrains,
    envbufnum: grainenv)
```

- grainenv is the envelope applied to the grains and is defined by :

```
winenv = Env(levels: [0, sustain, 0], times: [attack, release]);
grainenv = Buffer.sendCollection(s, winenv.discretize, 1);
```

  that is a triangular envelope whose vertex is at 'sustain' level

- the input of the GrainIn function is a mix of two signals :

```
 PitchShift.ar(
    in: gain1*BufRd.ar(1, sample1,
                Phasor.ar(0, BufRateScale.ir(sample1),
                0, BufFrames.ir(sample1),0)),
    windowSize: 1,
    pitchRatio: ratio1 + WhiteNoise.ar(grainrand) )
+
PitchShift.ar(
    in: gain2*BufRd.ar(1, sample2, ...
```