POLITECNICO
MILANO 1863

Granular Synthesizer

ACTaM Project

A.Y. 2021/22

Gargiulo Antonino Manuele	(Person code: 10829418, ID: 990594)
Morena Edoardo	(Person code: 10865449, ID: 996003)
Orsatti Alessandro	(Person code: 10680665, ID: 994757)
Perego Niccolò	(Person code: 10628782, ID: 992023)

6 September 2022

1 Introduction

The aim of the project was to realise a creative application involving sound.

We focused on the development of an easy-to-use Granular Synthesizer, enriching it with multiple functionalities like the possibility to chain it with some effects, to play it using both only the mouse or with mappable MIDI controls and allowing to save and reuse some great synth configurations.

2 Granular Synthesis

2.1 An historical view

Granular synthesis was first discussed as an outgrowth of quantum physics, when researchers such as Dennis Gabor examined the concept of reducing sound to its most basic building blocks, called *sound quanta*, in the late 1940's.
Gabor developed an interesting rotating tape recorder head that was able to break down the captured material into small bits for time and pitch manipulation.

Even before computers became capable of performing the large number of calculations required to generate the necessary massive amounts of data, Iannis Xenakis both discussed and used these atomic-level sound quanta, often exploiting an enveloped simple waveform like a sine wave, or simply a bell-shaped sound impulse called *wavelet*, to create statistically-shaped *density clouds*, as shown in his book "Formalized Music".

By the time computers were ready to provide significant control to the process, leaders in the granular synthesis technique emerged in Curtis Roads and Barry Truax. Truax was able to use his PDP minicomputer to control the grain production and results in real time.
The most notable musical composition realized using granular synthesis is "Riverrun" (the first word of James Joyce's "Finnegans Wake").

Of seeming greater interest in recent decades is the granulation of real-world sampled sound. This technique was used with fascinating results in Paul Lan-

sky's "Idle Chatter" series, circa 2011, along with several other techniques. Many DAWs, plug-ins, apps, and built-in or add-on objects for synthesis languages provide easy access to the technique both for creative and needed purposes.

2.2 Performing granular synthesis

The term "granular synthesis" is rather unspecific, as it can cover all systems that utilize the concept of granulation. **Granulation** is a specific process in which an audio sample is broken down into atomomic elements, called **grains**. A grain may have a variable length between 20 and 100 ms, where the lower bound is the integration time of the human ear (which let us differentiate sounds) while the upper bound is the limit between the granulation of a sound and a looping issue.

Generally, the original sample is split into a series of smaller ones. This original series, in chronological order, is called the **graintable**. Theoretically, if grains were played in this order at the speed of the original sample, the output would present no changes. However we can have a lot of control over audio grains. Each one can be individually manipulated, as can the overall arrangement of them.
In fact, we can play grains using different orders, individual grains can be looped, grains can be layered, omitted and skipped over. This semi-randomic process is what gives granular synthesis its distinctive feel.

After creating a new sequence of grains, volume cross-fades will be applied to blend from one gain to the next in a process called **smoothing**. The shape and length of the cross-fades have important roles in determining the tone of the resulting sound.

3 Functionalities and Interactions

The whole project is visually based on *Bootstrap 5.0* to offer a responsive experience.

The user is firstly presented various possibilities to select the sample to be fed to the synthesizer. In fact, a dedicated area can be clicked to open a finder to

select an audio file or can receive a sample drag-and-dropped on it. Lastly, the possibility to use one of the two pre-loaded default audio samples is present.
After this process, the main page is shown.

The first object to analyse is the **waveform visualizer**, realized with the library *Wavesurfer*. The user can interact with it by selecting the point around which the granular synthesis will occur. It is also possible to move the selected point in the waveform to change the sound dynamically.
The last clicked position remains saved on the waveform and will be used in case the synth is played through a MIDI keyboard.

Under the waveform, the main block of **parameter knobs** is shown. This dials are the ones that act on the main characteristics of the synthesizer. In particular:

- Density: time density of the played grains;
- Spread: area around the waveform pointer in which the grains to be played are created;
- Pitch: perceived frequency of the played grains;
- Attack: envelope attack of each single grain;
- Release: envelope release of each single grain;
- Volume: master volume of the synthesizer.

A particular combination of values can be saved on a data base through the "Save preset" button. All the sets of parameters created by the users, along with 4 presets (designed by us) can be applied to the synthesizer picking them from a dedicated `select` element.

4 parametric effects can be dynamically toggled and operated via the GUI. In particular, the **effects and customizable values** are:

- Delay
  - Feedback: the amount of signal fed back to be delayed;

  - Time: actual time delay;

- Reverb
  - Decay: seconds of decay of the reverb;
- Distortion
  - Amount: quantity of distortion applied to the grains;
- LPF
  - Cutoff: frequency at which the filter begins to attenuate the signal;
  - Resonance: emphasis or suppression of portions of the signal above or below the defined cutoff frequency.

3 **green buttons** grant the user the possibility to: reverse the current audio sample, with consequent update of the waveform, change the current audio sample and open the dynamic MIDI control assignment area.

When shown, this last block allows the possibility to link whatever MIDI controller to one or more of the main parameters of the synthesizer.

MIDI devices are managed dynamically during the execution. They can be used to play the synthesizer using the MIDI keyboard. The pitch of the grains is modulated with respect to what key is pressed. Both this and the MIDI velocity are handled in the *Granular* library.

4 Development

The project has been realised with HTML, JavaScript and CSS, exploiting various available libraries.

4.1 Granular and granular_module

The main engine of our Granular Synthesizer is the *granular-js* library, by Philipp Fromme. We locally included it in our project, in the `Granular.js` file,

in order to document it and modify it to our needs.

The `Granular` class manages the behaviour of the whole synthesis process. Actually, the `AudioContext`, the audio buffer of the loaded sample and the gain of the synth themselves are attributes of the class, used once a `Granular` object is instantiated. Another important parameter is the actual `state` of the synth, which stores the current `attack`, `release`, `pitch`, `spread` and `density` values. Lastly, an array of `Voice` objects (with respective ids), from the homonymous class, is contained in the Granular object.

The `Voice` class implements the `play` method, which is in charge of starting the synthesis process. The actual creation of the grains is done via the `createGrain` method, which is recursively called thanks to a `setTimeout` function. While creating and playing the grains, the following conditions are honored according to the choices of the user:

- the right amount of time between one grain and the other (density);
- the right centre position in the audio buffer (where the user has clicked). The correct normalized position is received from `script.js`, where it is evaluated thanks to the `updateCursorPosition` and `normalizeTime` methods when the waveform is clicked;
- the proper area around the waveform pointer in which the grains to be played are randomly chosen (spread);
- the right envelope of the single grain (attack and release);
- the proper pitch of the grain, also taking into account the note the user has played on the MIDI keyboard if one is used;
- the volume, also considering the velocity if a MIDI keyboard is used.

After properly loading an audio buffer, the `startVoice` method on a `Granular` object will create a `Voice`, insert it in the `voices` array and call the `play` function on it, starting the actual sound.

When a `Voice` needs to be stopped the function `stopVoice` is called with the `Voice`'s `id` as an argument. This function also deals with the removal of the stopped `Voice` object from the `voices` array.

In the `granular_module` script a `Granular` object is instantiated with the current `AudioContext`. Also an `Effects` object is created to manage the parametric effects to which the synth can be chained to.
Some utility functions are then defined and exported to be used in `script.js` to manage the state of the `Granular` object and the whole `Effects` object.

4.2 Audio effects: routing and typologies

All the audio effects in our project are realized exploiting the *p5.js* library. The routing follows this diagram:

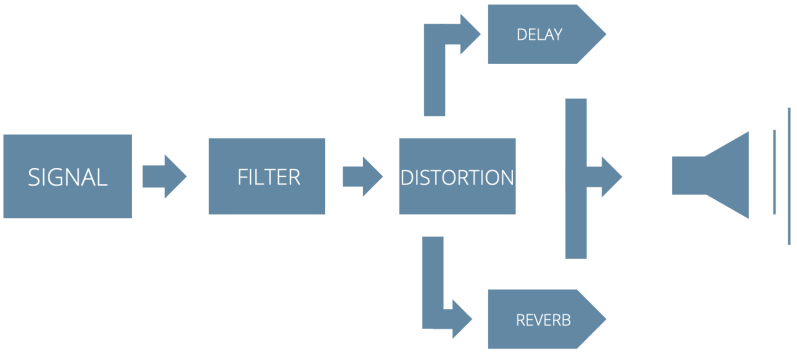

Figure 1: Effects routing

The signal is first of all filtered with a *Low Pass Filter* with parametric `cutoff` frequency and `resonance`, which output is fed into a *Distortion* block, with variable `amout` of effect. Then, both a *Reverb*, controlled in `decay` time, and a *Delay*, with parametric `feedback` amount and `delay` time, receive the audio from the *Distortion* effect and elaborate it. Finally, their output is summed and routed to the destination.

Every effect is hadled with the relative *p5* object and can be switched off with a proper toggle button. For the `Distortion` and `Filter` objects, this means just selecting the respective source signal to be only sent on a dry path to bypass the

effect. However, new `Reverb` and `Delay` objects need to be created and chained correctly each time the user wants to turn those effects on.

The parameters of each effect are directly linked to the knobs present in the GUI, which trigger the update of the values both in the `Effects` object `currentState` and in the single *p5* ones.

4.3 JQuery Knobs

We exploited the *JQuery Knobs* library to create all the dials of the main page. It allows to develop canvas based elements, with no png or jpg sprites, that handle touch, mouse, mousewheel and keyboard events.

HTML elements are of the `input` type and are characterized by the class `knob`. The other classes assigned to these elements are used to style them and also serve a classification purpose in JavaScript. Each one of this elements has some special parameters which set their height, width, min, max and default values.

```
<input id="density-knob" class="knob knob-font-big par-knob
    animated" default="0.5" data-min="0" data-max="1" data-width="
    140" data-height="140" />
```

Each dial is then initialized in `script.js`. Every knob is firstly selected through *JQuery* and then shaped with the method `knob`, which sets its main style and function properties, interpreting both the data fields specified in HTML and JS:

```
$('.knob').each(function () {
    var $this = $(this);
    $this.knob({
        'step': 0.01,
        'angleArc': 270,
        'angleOffset': -135,
        'lineCap': 'round',
        'fgColor': '#33383a',
    });
});
```

Properties that are specific to a certain class of knobs (or even to single ones), e.g. the change event, are then set using the `trigger` method with the `configure` setting:

```
$this.trigger(
    'configure',
    {
        'change': function (v) {
            updateGranParValue(elementId, v);
        },
        'release': function (v) {
            updateGranParValue(elementId, v);
        },
    }
);
```

Both the `change` and `release` properties need to be set to grant the possibility to update the knob value both with clicking and scrolling. Depending on the purpose of the dial, different exposed methods from `granular_module` are used to update the `Granular` object `state` and the `Effect` object, which compose the model of our application.

Parameter knobs are given the `animated` class, used to recall them when the application stars or when a set of parameters is selected by the user, in order to make the visualized data swing to the correct one. This is done through the `animateToValue` and `animateToDefaultValue` methods.

4.4 Firebase

Firestore is a non-relational data base offered by the *Firebase* library. In our application it is used to have persistency both on default presets and on the sets of parameters created by the users. For each saved entry, a document is present in one of the 3 *Firestore* collections:

- `presets`: which contains all 4 presets chosen by us (plus the default one);
- `user_presets`: which contains all sets of parameters saved by the users;

- `preset_num`: which contains the number of saved user sets.

The first time the main page is loaded, `populatePresetList` is used to insert in the `preset-select` HTML element all the combinations of values contained in the `presets` collection. Then, the same is done for `user_presets`, using `populatePresetListUser`.

To keep track of the up to date number of sets created by the users and to correctly name new ones (without overrinding the contemporary creation of other sets by other users) a counter is stored in the `preset_num` collection.
This value is also used to update only the changing part of the list of sets, each time a new sample is loaded (or the entire web page is refreshed).

When a user wants to save its own set of values, the listener of the click event of the `save-preset` button requests the current number of sets on the DB using `getCurrNumOfPresets`. Then, the `Granular` state is gathered with the exposed method `getState` from `granular_module`, and then a new document is created and saved in the `user_presets` collection. Following this process, the list of sets in the dedicated select is updated.

5 Conclusion

The achieved results fills the specifications we planned for the project and allowed us to acquire important knowledge on the synthesis topic, and also to further develop our skills in programming, following a newly found creative mindset.

Further work on the project might include:

- allowing the user to save each preset with a specific name;
- using an authentication service to offer a unique experience to each user;
- adding the possibility to operate effect parameters with MIDI controls.