CMLS Homework 3 - Group 9

10724182 10743181 10766268 10768481

https://github.com/mttbernardini/cmls-hw3

1 Introduction

The aim of this document is to describe the realization of our digital equalizer, SpacEq. We decided to implement a one-third-octave graphic equalizer, with 30 parallel band-pass filters.

In order to do that we used SuperCollider, and we set up the possibility for the user to load an audio file and modify the gain on the single bands and the panning of three macro-sections in a quadraphonic system. In section 2 the implementation process is described in detail.

Section 3 instead describes the Graphical User Interface of our program, implemented in Processing. It uses OSC communication protocol to interact with SuperCollider.

2 Graphic Equalizer

The graphic equalizer is a tool for precisely adjusting the gain of multiple frequency regions. Structurally, it is simply a set of filters, each with a fixed center frequency. The only user control is the amount of boost or cut of each frequency band.

The basic unit of the graphic equalizer is the band. A band is a region in frequency defined by a center frequency ω_c and a bandwidth B or quality factor Q. These three terms are related by $Q = \omega_c/B$. The gain G of each band is controllable by the user. Typical gains range from –12 dB to +12 dB ($0.25 < G < 4$), with 0 dB ($G = 1$) meaning no change or flat. The center frequency ω_c and bandwidth B are not user-adjustable.

Usually the bands in a graphic equalizer are distributed logarithmically in frequency to match human perception. The number of bands is determined by their spacing and the requirement to cover the entire audible spectrum.

One-third-octave designs have 30 bands ranging from 25 Hz to 20 kHz. These frequencies, are standardized by the International Organization for Standardization (ISO) [3]. The calculation of center frequencies f_c is computed keeping in mind that $f_c = f_0 \cdot \sqrt[12]{2^n}$ is the frequency of the n-th semitone from the reference frequency f_0.

In our case, we choose $n = 4$ to represent one-third-octave center frequencies, so that

$$f_{c,i} = f_0 \cdot R^i = f_0 \cdot \sqrt[12]{2^{i \times 4}}$$

is the center frequency of the i-th band. R is a fixed ratio between each band. Bandwidth is chosen consequently setting the reciprocal of the Q factor, meaning:

$$Q^{-1} = \frac{B}{\omega_c} = \sqrt{R} - \frac{1}{\sqrt{R}}$$

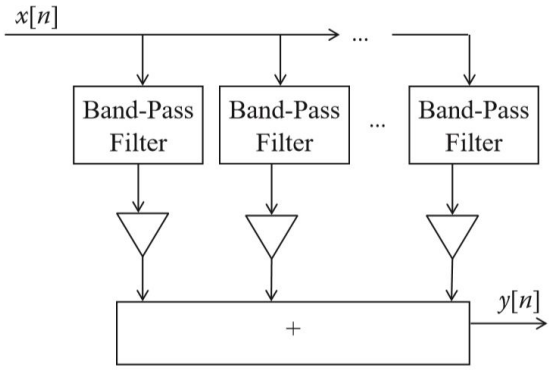

Figure 1: A diagram of a graphic equalizer, implemented with band-pass filters placed in parallel, with N bands of control

2.1 Equalizer Architecture

Graphic equalizers can be implemented as a cascade of peaking/notch filters. This method, however, can bring to undesired effects, such as a sum of the nonlinear phase responses of each IIR filter, or an audible delay in case of many FIR filters in series.

For this reasons, we decided to implement SpacEq as a set of 30 parallel *Band-pass filters* (see figure 1).

2.2 Quadraphonic spatialization

With a spatiomorphological approach [4] in the back of our heads, we decided to conjugate the possibility of acting jointly on spectral and spatial characteristics of sound.

Therefore, our equalizer has an internal quadraphonic spatialization function which allows the user to spatialize finely-filtered sections of the spectrum.

In particular, previously filtered sounds are grouped in three macro sections for low, middle and high frequencies, and sent to the four output channels. Panning is adjusted for each one of them by their respective $x - y$ plane control.

As explained before, details of the spectrum can be finely selected through the graphic equalizer. Supercollider provides an easy way to implement a four channel equal power pan, which takes the square root of the linear scaling factor for a smooth panning. Thanks to these features, SpacEq can be used for sound design as well as for multichannel and acousmatic compositions.

2.3 Controls

SpacEq allows the user to control the gain of 30 filters and the position of three groups of bands in a two-dimensional space. In addition to this, SuperCollider sends to the GUI a feedback on output volume values, in order to be displayed by 30 separate meters. This is achieved thanks to OSC protocol, as explained in detail in the following section.

3 User Interaction

Control of spacEq is achieved through OSC Communication between SuperCollider and a Graphical User Interface built in Processing [1].

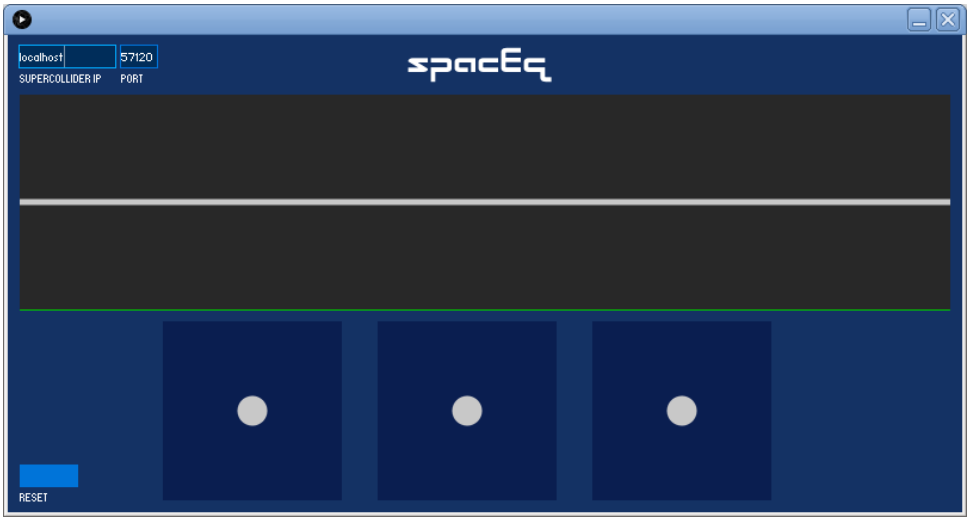

Figure 2: GUI: initial conditions

3.1 OSC Endpoints

The SuperCollider `SynthDef` provides two kinds of controls: *gains* and *pannings*. The first one is adjustable for each band b through the endpoint `/eq/gain/`b, which receives as argument a value ranging from -12 to $+12$, expressed in dB. The latter is adjustable for each group of bands g through the endpoint `/eq/pan/`g, which receives as argument a pair of values x and y, both ranging from -1 to $+1$.

The Processing GUI provides the endpoints `/gui/volumes/`b for each band b. Each of them receives a value ranging from $-\infty$ to 0, representing the volume of the b-th band, expressed in dB.

3.2 Graphical User Interface

The GUI is composed by N vertical sliders to control the gain of each EQ band. Each slider also provides a visual feedback, in form of a meter, of the current volume level of the respective controlled band.

On the lower part, each panning control is related to a group of frequency bands, three in our case, representing low, mid and high ranges. Controls are displayed as circles that can be moved from the user inside squares, where each vertex represent a speaker of a 2D system.

In the left down corner of the window a reset button is provided, which sets back sliders and panning controls to the default values.

Finally, two text boxes are provided to insert address and port of the remote SuperCollider application to be controlled.

In figure 2 the initial condition of the window is displayed. Figures 3 and 4 respectively show an example of interaction by adjusting a gain slider or a pan control.

3.3 Implementation details

We employed the library ControlP5 for the realization of text boxes and labels, while for the sliders and the panning controls we preferred to implement our own classes, in order to have more flexible behaviors for our needs.

In fact, we needed the possibility to drag the position of the sliders with a single mouse click by moving the cursor along the spectrum. This allows the user to "draw" curves representing

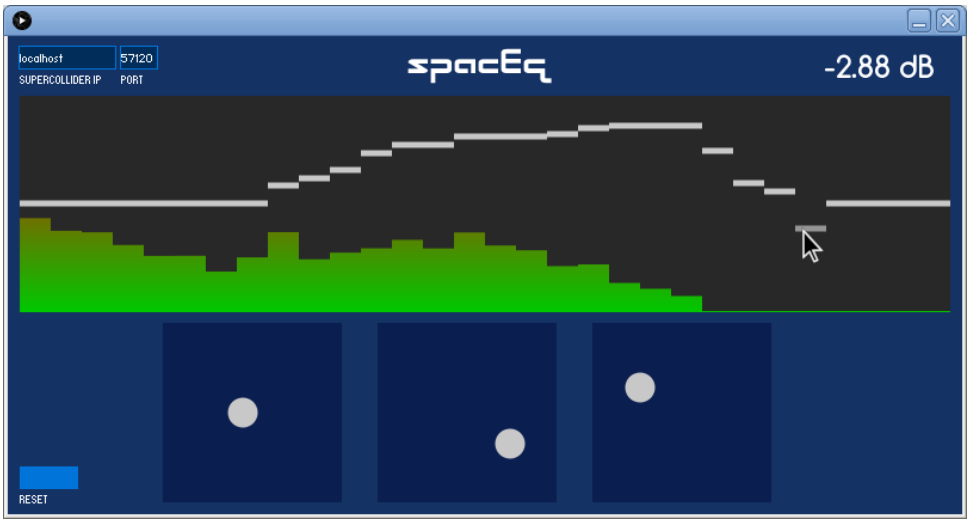

Figure 3: GUI: regulating a gain slider

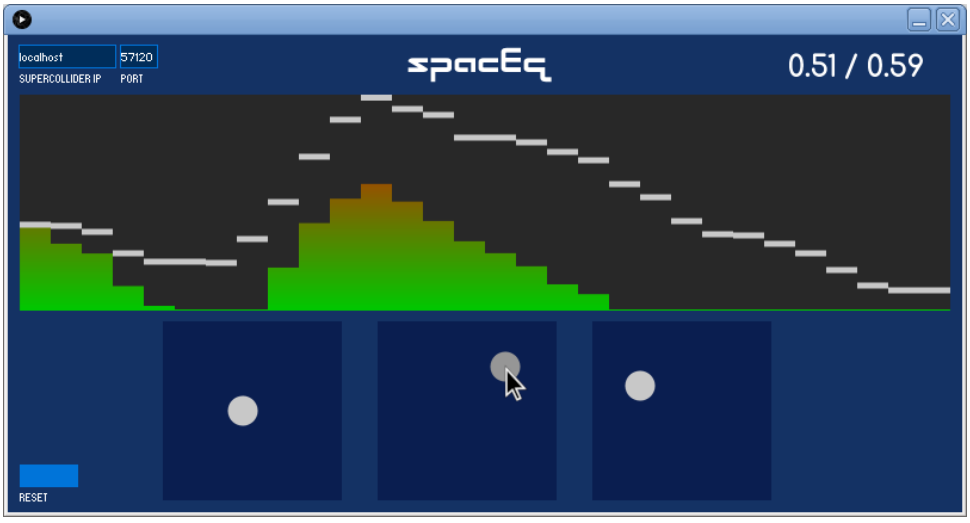

Figure 4: GUI: regulating a pan 2D slider

the desired trend and give artistic nuances to their audio. Clearly this is meant for a creative approach, but the equalizer can also be used in the classic way to do precision mixing operations.

We embraced the object-oriented approach of Processing by following inheritance and polymorphism patterns to realize our custom components.

Firstly, we implemented the base abstract class `UIElement`, which provides the common interface of each drawable component and allows to generalize collections of different components.

Then, for the implementation of the sliders (`EQSlider`) and the panning controls (`PanSlider`) we extracted the common logic between the slider thumb and the circles inside the squares, which can be thought as a 2D slider, in the class `SliderValue`. This class handles the conversion between cursor coordinates and values, and implements the boundary logic of the containers.

For the layout, we implemented three classes, each one extending the previous one, `UIGroup`, `UIStack` and `UIFlexbox`. The first one allowed us to group elements inside of a root object, on which we can call the `draw()` method which delegates the drawing of the children elements. The second one allowed us to lay elements in a row or a column, by determining automatically the position of the children elements. The last one, finally, allowed us to dynamically compute the size of the children elements based on the size of the container, in a flexible manner.

We further subclassed `UIFlexbox` in `Mixer`, which simply takes care of drawing a gradient as background of the meters, by calling the function `makeVerticalGradient`. Meters are actually implemented by hiding the portion of the gradient which is not relevant whith a black rectangle. We resorted to draw a single gradient instead of a gradient for each separate slider, which could allow us to introduce some spacing between the sliders, because of degraded frame-rate and performances in having to repeat so many cycles to draw the gradients.

References

[1] Processing reference. `https://processing.org/reference.html`.

[2] Supercollider reference. `https://doc.sccode.org/`.

[3] Joshua D Reiss and Andrew McPherson. *Audio effects: theory, implementation and application*. CRC Press, 2014.

[4] Denis Smalley. Space-form and the acousmatic image. *Organised sound*, 12(1):35–58, 2007.