Assignment 3: SuperCollider and OSC Protocol

Group 9: 10804339,10796569,10528650,10622373

May 29, 2021

https://github.com/FrancescoBorgna/ImageChordifier

1 Introduction

The goal of this project is to develop an innovative digital musical instrument and based on an additive synthesis synthesizer, as well as a unique interface to control it. The user uploads an image, which is displayed; the user then moves the cursor on the image. Each time a different pixel is selected, a chord based on the color and tonality of that particular pixel is generated and played back. A use case for this instruments would be, for example, an interactive website experience, perhaps with a "cover size" background image; or an artistic installation at a particular venue.

2 SuperCollider - SynthDef

The synthesizer used to produce sound through additive synthesis is defined using a `SynthDef` object to which are passed various arguments. Mainly, there are frequency, amplitude, reverb dry/wet ratio, ADSR parameters, chord selection and slide duration. Also some useful variables are initialised and an array containing all the offset values used to build chords from its root note.

The core of our synthesizer is found inside the `Mix.fill()` method. This method will mix 5 single channels containing a synth for each note of a chord iteratively. For each iteration we have: the iteration index, a temporary array used to store the current chord, an additive factor, the envelope used for frequency sliding. To pick the current chord that needs to be built a `Select` object is used. That's because UGen variables (like chordIndex, an OutputProxy variable) are used, which can't be used as normal Integers for indexing. The same happens when the offset is retrieved. To go from a chord to the other, an envelope is built and applied to the frequency of every note. The current note (in midi notation) is then stored for the next chord. Note: a global variable is used to keep track of each note's index in the previous notes array also to preserve the Integer type.

The output of the mixed synths is then fed into a second order resonant low pass filer and into a simple reverb effect. An ADSR envelope is specified and triggered with a specific argument.

Lastly the signal is fed to the output stereo channels and the volume envelope applied.

3 SuperCollider - OSC Reciever

With an `OSCdef` object we allow the software to receive OSC messages from the GUI at any time. An argument is automatically set to be an array of the received objects. Some supporting variables are declared. Every time a message is received the additive synthesizer's arguments are changed and the envelopes (ADSR and frequency sliding) are triggered. Every message contains (OSC address excluded):

- Reverb dry/wet ratio
- Midi root note
- Chord index
- Peak of the envelope
- Attack time
- Decay time
- Sustain level
- Release time
- Frequency LPF cutoff
- Slide duration

How these values are selected by the user will be discussed in the next section.

4 Processing - Value selection

Every time the mouse is pressed inside the boundaries of an image the RGB color of the selected pixel is retrieved; hue, saturation and brightness values are extracted from it.

Now the association of these values to sound/music values is going to be explained, but take in mind that many other methods of color-music association could have been applied being this a totally subjective interpretation.

4.1 Hue

Hue consists in a set of values that represent the color spectrum, often this range of colors is represented with a wheel, because the values wrap around (red = 0 or 256). This wrapping around has been associated to the wrapping around of the musical notes. In addition, blue has been mapped to the lower note of our notes array (A2 = midi note 45) and from here the note's pitch goes up to our last note (Ab3 = 56), associated to light blue. This is why, with a simple algorithm, hue values are, firstly wrapped around blue (170), and secondly mapped to an octave of 12 notes.

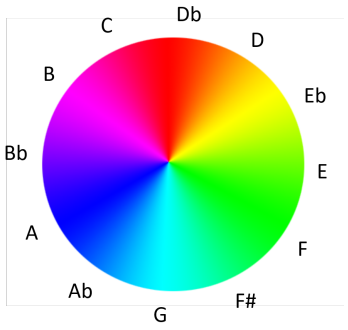


Figure 1: Color-notes association

4.2 Saturation

Saturation is a measure of colour intensity. As saturation increases the color appears to be more pure; as it decreases the color appears more washed out and dull. To give the sense of washing out the sound reverb has been chosen. Therefore, saturation values have been mapped to the reverb dry/wet ratio: when saturation goes to 0 (gray/washed out color), reverb wetness goes to 1 (washed out sound); when saturation goes to 1 (pure/vivid color), reverb wetness goes to 0 (pure sound).

4.3 Brightness

Brightness is the relative lightness or darkness of a particular color, from black (0, no brightness) to white (1, full brightness). Color brightness has been associated to the auditory "brightness" or "mood" that certain chords can give. More on how this chords have been selected and mapped in the correspondent sections.

5 Chord Selection

The authors decided that having a pool of chords to draw from, rather than generating them from scratch as a function of the image parameter, would yield to more predictable and pleasing musical results.

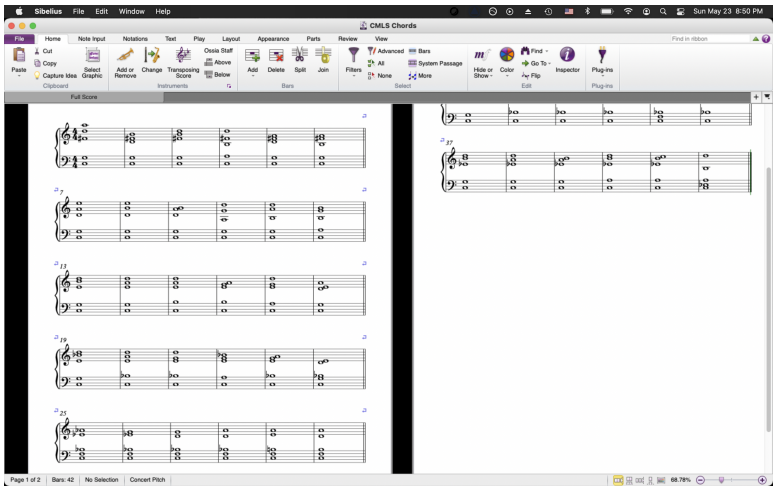
A set of 42 chords of five notes each, ordered, subjectively, by brightness, was created. The chord families are Lydian, Major Seventh, Suspended Fourth, Dominant Seven, Minor Seven, and Minor Sixth. All of the chordal structures are then stored in a matrix; each chord is a vector of five

values to be added to the MIDI note number representing the root. This approach ensures ease of transposition.

A second array has been created containing the nomenclature of each chord (e.g. Cmaj7add9, C7b9b13, etc); the indexes of this array match the row index of the chord matrix. This array is utilized in the GUI, to show the user what chord is in fact being reproduced.

6 Chord converter utility

The set of chords was initially compiled using standard musical notation using the AVID Software Sibelius.



Sibelius can export musical notation files in the MIDI format, as well as in the MusicXLM format, the latter being an open format widely used as an interchange format between different music notation software.

An auxiliary Phyton script was written, which parses the XLM data from the MusicXLM file, and outputs a bi-dimentional array correctly formatted for Supercollider. This solution allows for more flexibility should the need arise to revise the chord set, or to outsource its creation to a professional music composer.

7 Sigmoid distribution of the Chords

Since the values of brightness of the pixels from average images uploaded by the users very rarely approach the extreme values (<0.2 and >0.8), the authors have found it effective to apply a sigmoid function to the normalized brightness value. In other words, better use is made of the the pool of chords as more resolution is available towards the center of the range (0.5). A graph of the function is below:

8 Communication in Processing and Supercollider: Oscmessages, net address

OSC stands for Open Sound Control, and consists in a protocol for networking between computers, synths and various multimedia devices. For instance, it allows a software to communicate with a hardware synth, whenever the latter supports OSC.

One of the most important, or rather most useful difference, is that OSC allows to send any type of messages at high resolution to any address. Differently, the MIDI protocol has its own specific messages, like note On, not Off, pitch, etc., with low resolution.

OSC communication between programs is often done by sending messages from one application to another. SuperCollider has OSC support in its core, as it is based in a client-server model for communicating between the client and the server-side.

On the other hand, Processing supports OSC communication via the oscP5 library which is one of the standard libraries that Processing is shipped.

8.1 Supercollider

SuperCollider is an engine for sound synthesis and algorithmic music composition. In SuperCollider the communication is done by creating a NetAddr of the target application and creating an OSCFunc to listen to another application.

To establish the communication with another application (Processing in our case), you need to know on which port that application is listening. We can create a network with NetAddr with the listener.

SuperCollider will listen to messages at the defined port and address. In a typical configuration server and client are on the same machine and they communicate through the localhost (127.0.0.1), but SuperCollider can be reconfigured to accommodate scenarios where client and server are on different machines.

```
NetAddr("127.0.0.1",57120);
```

8.2 Processing

Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. It is a Java-based language. In order to establish a communication with Supercollider, we have to import oscP5, which is an OSC implementation for the programming environment processing.

```
import oscP5.*;
```

We also import some network libraries for processing which supports UDP, TCP and Multicast.

```
import netP5.*;
import java.io.File;
import controlP5.*;
```

As said before, OSC messages are sent over a network, so we must define where we want to send our message. We define the following things:

IP Address – The IP address of the device where our message is sent. Usually the localhost IP address is 127.0.0.1

Port – The port number where we are sending our message. This could basically any number, but a lot of ports are reserved for other purposes.

In this case:

```
NetAddress myRemoteLocation;
...
myRemoteLocation = new NetAddress("127.0.0.1",57120);
```

myRemoteLocation is a NetAddress that takes the 2 parameters (ip address and port number). *myRemoteLocation* is used as parameter in oscP5.send() when sending OSC packets to another computer, device or application (Supercollider).

```
OscMessage myMessage = new OscMessage("/color");
```

It creates a OscMessage object: the first argument is the OSC Address Pattern, i.e. an OSC string beginning with '/', that specifies the receiver of an OSC packet. After that we add three float numbers to OSC our messages in orde to add some parameters such as saturation, brightness:

```
myMessage.add(hue);
myMessage.add(sat);
myMessage.add(bri);
```

At the end we send our OSC message to our address:

```
oscP5.send(myMessage, myRemoteLocation);
```

9 GUI

In order to give to the user an easy way to interact with our synth, we built a minimal user interface comprising the following controls:

- Image Selector

- Image Display
- Current Note
- Current Chord
- ADSR Controller
- Cutoff Filter Knob
- Sliding Factor Controller

All knobs and buttons belong to the class *ControlP5*. Each one of these controllers is linked to their relative variable which is then sent to Supercollider through OSC messages.

The following table sum up the characteristic of each controller:

Knob	Control	Range	Variable
StartVal	Starting Value of Env	0-1	SVal
attKnob	Attack	0-3 sec	att
decKnob	Decay	0-5 sec	dec
susKnob	Sustain	0-1 sec	sus
relKnob	Release	0-5 sec	rel
freqCutKnob	freqCut	0-10000 Hz	freqCut
slideKnob	Sliding factor	0-3 sec	slideDur

Table 1: GUI Controllers

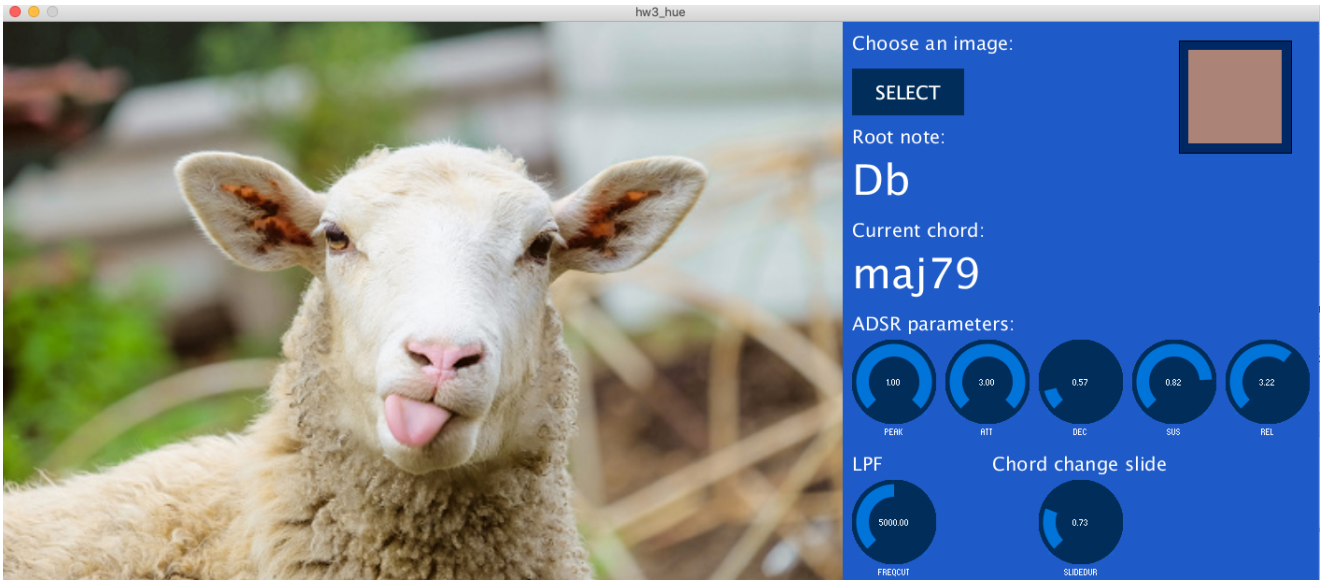


Figure 2: GUI view

10 Improvements and further developments

Improvements and areas of further study related to this project include:

- expanding the pool of chords, and/or coming up with an even more structured approach to their organization by brightness;
- including passing chords between the various selections;
- investigating the use of machine learning to come up with an even broader range of possible chords/intervallic combinations;
- devising machine learning tecniques to associate colors and chords;
- devising machine learning tecniques improve the chord progressions;
- applying the software to a live video stream: the average hue, brightness and tone of each frame are computed; new sounds are generated based on the video;
- porting the code to a web-friendly framework.