CMLS - HW3 Report

10494722
10800940
10502570
10719249

May 29, 2021

Contents

Chapter 1

HW3 - Drum Kit

1.1 Homework

The aim of Assingnment 1 is to create a set of drum sounds, with a minimum of 3 and an interface to control them. It is possible to implement non traditional drum sounds. Is possible to exploit the interactions with MIDI or OSC communication protocols. The GitHub repository of the code is available at this link:

`https://github.com/DavideTommy/Drum_Kit-CMLS_HW3.git`

The repository is already public. Click on link above to reach.

1.2 Idea

In order to achive our homework we need to create a set of sounds and we did it using supercollider SynthDef. After that we needed to create a GUI and we decided to use TouchOscEditor to prepare it. We downloaded the .apk of android app and we uploaded our template on our mobile phones. The final step consists in creating listeners able to receive special key words coming from button pressed and call the related Synth function to play the desired sound.

1.3 Implementation

1.3.1 GUI - TouchOSC

For the GUI part of our project, we decided to exploit the capabilities of the TouchOSC, a modular control surface in the form of an app for Android and iOS. The creation of this environment was made possible by the TouchOSC

Editor, a tool that allowed us to create many different graphic elements, like labels, linear and rotary encoders, all useful to set a drum-kit-like interface which the user can find intuitive and aesthetically pleasing. From the editor, it is possible to move, resize and change colors for each element. Our idea for the GUI was to create a visual environment based on the actual appearance of a real drum set, as seen from above. Each element corresponds to a circle that responds to tap from the user's finger. For each percussion there is the relative controller for the gain. For each graphic element is possible to set a range of value that he can assume and a particular keyword.

When an object is pressed on the screen of our Android device, the TouchOSC sends the keyword through the socket connection and the Super-Collider server catch it activating the relative SynthDef and so producing the desired sound. Gain sliders have the same behaviour, but they act directly on the out gain of each sound.

To give the user further control of the audio output, we implemented faders that acted on gain and effects. At the top left we have located two master controls through two sliders, one for the room reverb and one for the general panning, both being set to zero at the start.

When choosing a higher value for the panning, the user will perceive the sounds from the different elements of the kit according to their visual location. Immediately below, we implemented a metronome, with a toggle for its activation and a slider to set its BPM. In the center we have the drum-kit, rendered graphically with circles of different colors and sizes: the three yellow ones are the battery cymbal, while the others are the drums. At the bottom left there is a minimal legend which shows the names of each instrument. On the right there are other controls: At the top there is the master volume, which is adjustable thanks to a knob; under it there are other rotary knobs, each one for adjusting the volume of its corresponding drum or cymbal.

1.3.2 Sound Processor

The core of the sound processing is Super Collider. We need to define a SynthDef for each sound we wanna obtain. For each sound is possible to set the frequency and the amplitude of the sound we want to obtain. Is also possible to add different sounds in the same SynthDef.

At the end of the Systhesis section we have the series of listeners to call our function as soon as we receive the proper message from the graphic interface.

The first of the two blocks of code of the Supercollider file includes the library of sounds for our drum kit. We chose to use sounds that mimicked a real drum set with a kick, a snare, a hi-hat, three toms and two cymbals. We procedeed with a list of `SynthDef` functions which allow us to create sound and add it permanently to the server without playing them, the last part being saved for the interaction between SuperCollider and TouchOSC.

Our implementation is based largely on subtractive synthesis, a sound building technique in which partials of an audio signal (usually rich in an harmonics) are attenuated by filtering in frequency to alter the timbre of the sound.

For example, the kick and snare sounds, very different in nature, were built using a low pass filter (`LPF` function) and a band pass filter (`BPF` function) respectively, with the aim to cut and save specific frequencies (for instance, for the kick we threw away high frequency to obtain a severe sound).

We also used different kind of noise to obtain different sounds. For instance, `PinkNoise`, softer than `WhiteNoise`, was used in the crash cymbal while `Clicknoise` fitted well with the hi-hat. Being our goal to implement a drumkit, we also applied the effect of envelopes (in time) and created them with the function `EnvGen`. In this way, applying a ADSR (Attack-Decay-Sustain-Release) regulation, we obtained a percussive effect.

In each block of `Synthdef` we also set some important parameters; among these the ones related to reverb and panning of sounds.

For the first one, we used the function `FreeVerb`, putting as `mul` the parameter `ampRev` (amplitude of reverbation), which will be controlled by the user interface.

For the second one, besides using the function `Pan2`, the key to control panning was the product between the parameters `pan`, which is set depending on the position of the object in the interface, and `panning`, set to 0 and changed by the user with its slider. Another important aspect concerning envelopes is the `decayAction` parameter of the `EnvGen` UGen.

When set at a value of 2, when the envelope is finished playing, a message is sent for the synth to be deallocated or 'freed'. Our first implementations of the program ended with an error message from the Post window caused by insufficient memory allocation. The problem was the continuous creation of synths triggered by the interface buttons which were not deallocated. Including an envelope that freed the synth after 5 seconds turned out to be a satisfactory solution.

1.3.3 Listeners

In the second part of the code, we built the connection with TouchOSC. Using the OSC communication protocol, SuperCollider can receive messages from the app each time a user interacts with it. For sound output, TouchOSC sends a message every time the user presses one of the buttons corresponding to an instrument. This message includes a special keyword, used to properly route the message to the listener, and the payload (the value) stored in. (`msg[1]`. Each time the listener is triggered, a new synth is created with the right name and parameters. The gains, pan and reverb work the same way, their values in SuperCollider being set by the value from the sliders. We made the code so that the function will post in the Post window of SuperCollider the current value for a matter of further precision. In addition to the instrument data, the general values such as the master values, or those relating to the metronome bpm are also taken.

1.4 Conclusions

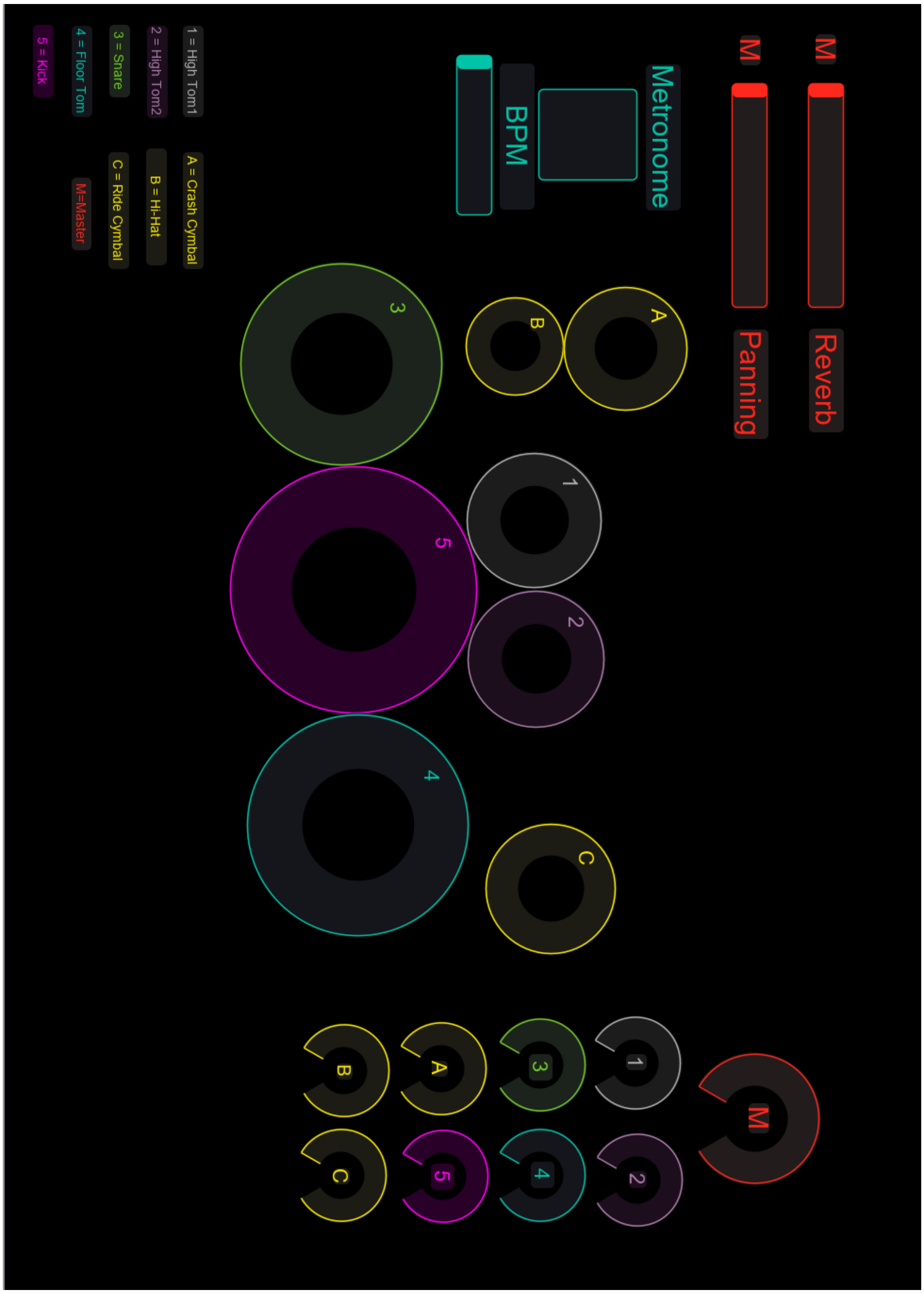
1.4.1 Features

With further implementations is possible to substitute our sounds with pre sampled sounds, transforming our Drum-Kit in a functional effect box that can be used, for example, as musical effect generator for Streamers or DeeJay, instead of a MIDI box.

We think this should be very useful in order to remotely control not only sounds, but also equalizers, sliders, mixers etc. That should be really useful in applications like sound tuning and acoustic regulation of an arena, or a theater.

1.4.2 Graphic Screenshots

The following is a screenshot of the graphical interface as seen from the TouchOSC app.



M
M
Reverb
Panning
Metronome
BPM
A
B
3
1
5
2
4
C
1
2
3
4
A
5
B
C
M
1 = High Tom1
2 = High Tom2
3 = Snare
4 = Floor Tom
5 = Kick
A = Crash Cymbal
B = Hi-Hat
C = Ride Cymbal
M=Master