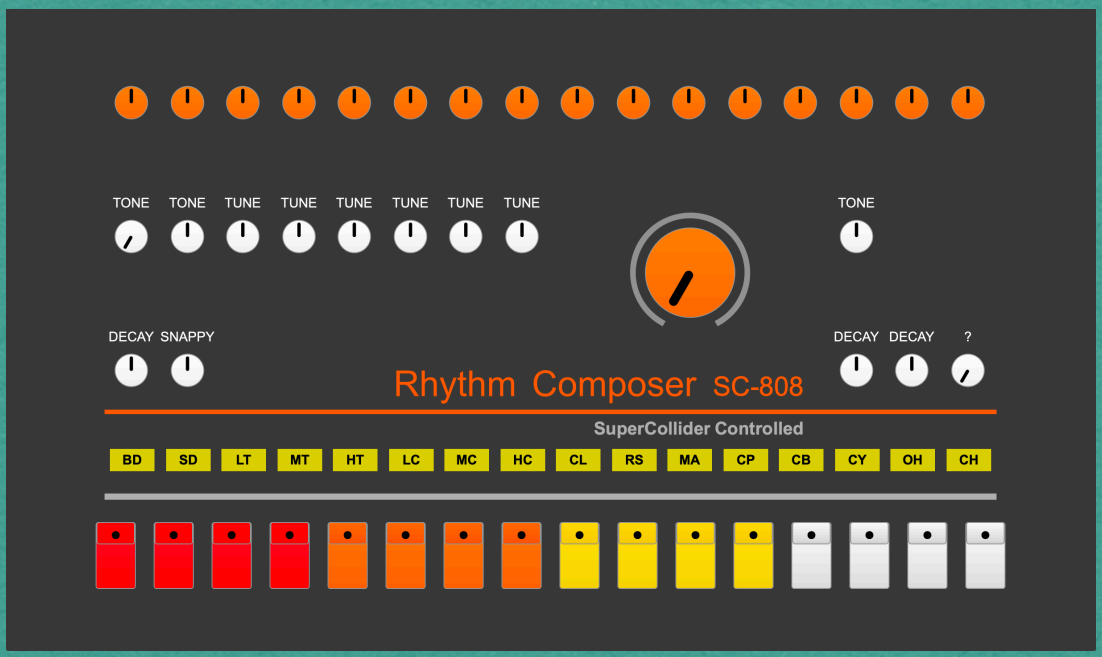
808

User Manual



Yoshinosuke Horiuchi

Table of Contents

Introduction

SC-808 is running on SuperCollider which is for sound programming and composition for composers, sound designers, and researchers and is used a programming language called sclang to coding. This 808 is not a sample-bases, is synthesized, and emulates the sound (not strict modeling) based on the original circuit. Therefore, you can modify the code and the sound easily and flexibility.

Background

This project was started during Lockdown. I'd like to explore the music with Drum Machine, but I had no money enough to get that because I have been an unemployed due to mental illness and Covid-19.
However, I had time to get start something. The decision of my passion was to create Drum Machine by software.

While a month, I learned and researched about SuperCollider and original circuit and built an 808. This is the culmination of which I've learned.

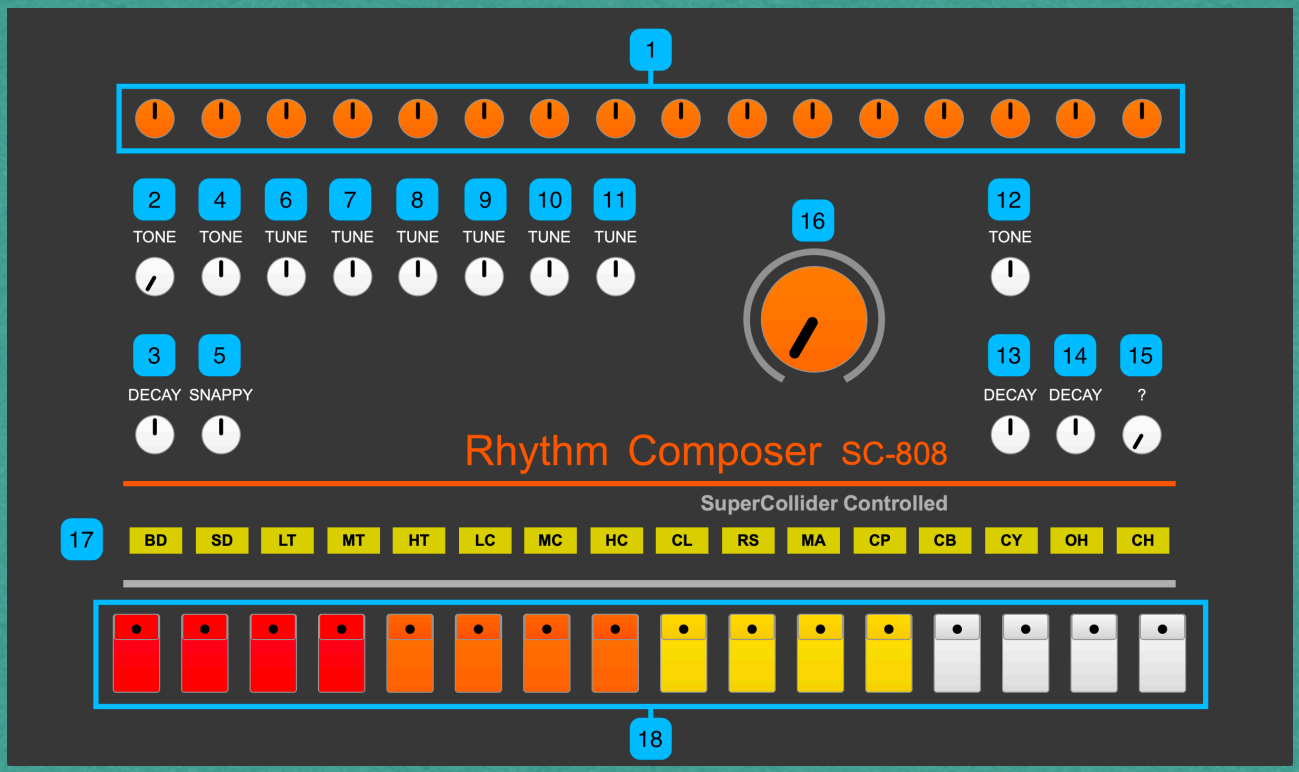
I guessed there are many people who have much desire for precious music experience like me. Additionally, we are confronting not expecting social situations.
I decided to give you 808 for FREE.

I'm so happy to distribute it to everyone.
Thank you so much for finding and exploring the 808.

Code by
Yoshinosuke Horiuchi
Donate

Overview

SC-808 has two sections, GUI and TextEditor. Text Editor is literally for code. Text Editor and Code sections mention in this part but a more specific detail is later. This Overview Part, mainly, focusing the GUI features such as each knobs and buttons.



1 Volume Level Knobs - All of the instruments' volume are controlled there, the order of each knob is correspond to the instrument name label. (See the instrument name label)

2 Bass Drum Tone

3 Bass Drum Decay

4 Snare Tone

5 Snare Decay

6 Low Tom Tune

7 Mid Tom Tune

8 High Tom Tune

9 Low Conga Tune

10 Mid Conga Tune

11 High Conga Tune

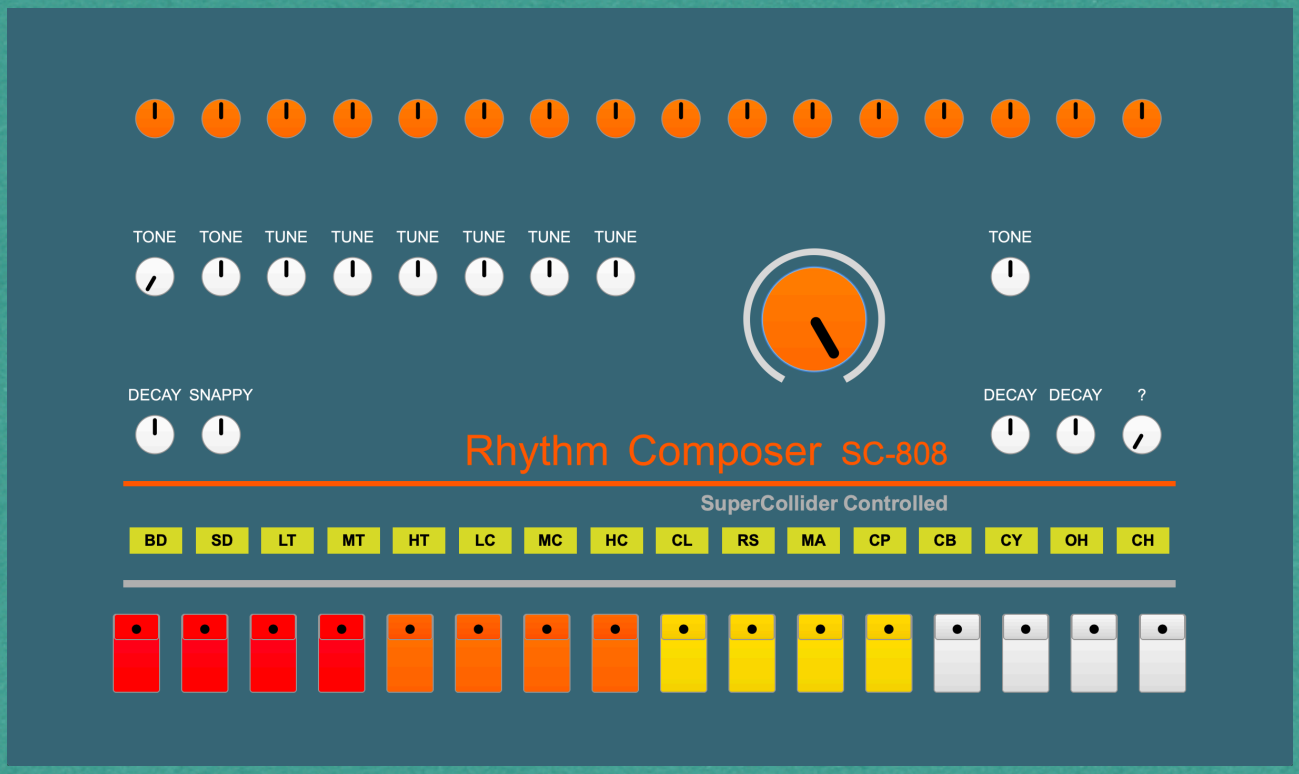
12 Cymbal Tone

13 Cymbal Decay

14 Open Hihat Decay

16 Background Color Knob

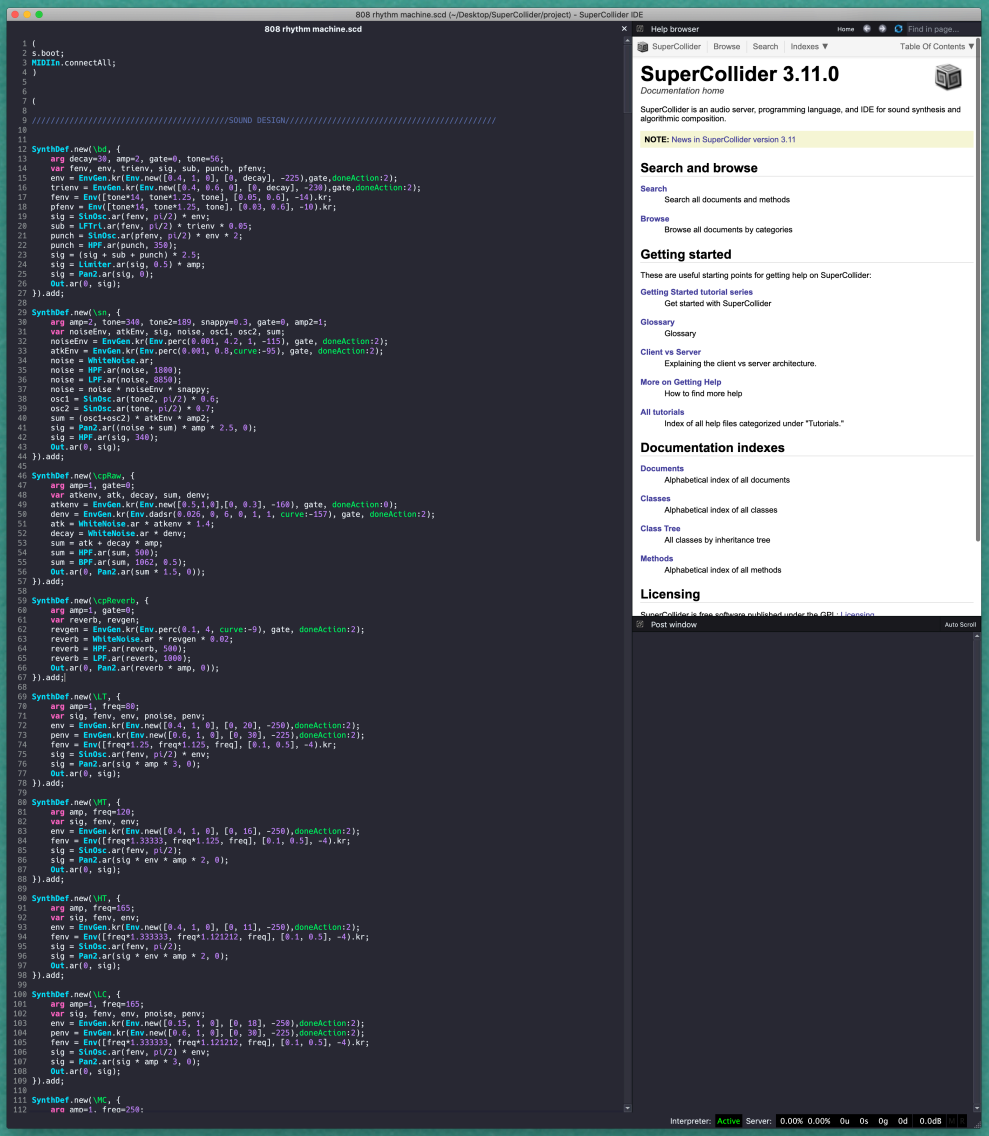
15 Secret - Please check it by yourself.



16 Turn the knob left to right, the background color is changed blue.

17 Instrument Name Label

18 Play Button - It plays immediately each instrument sound currently setting and a light flashes color red. Each button corresponds to in order of the instrument name label.



Text Editor, Post Widow, and Help Browser

Text Editor is for constructing the instrument. Type a code there and check the error on the post window. If you'd like to search about UGen, function, and so on, put the cursor within that then hit the key Cmd+D (macOS) Ctrl+D (Windows). You'll access easily the document on the Help Browser.

How to Use

Described above, SuperCollider is used sclang which is interpreted programming language. I guess that most people may be thinking now " I don't know about the code and what talking about, please don't make me scared." In this part, I explain how easy SC is and also basic knowledge to execute the 808 with some examples. Don't worry about it.

Installing

1. Download SuperCollider via https://supercollider.github.io.
2. Unzip the file then move the folder to Application Folder (macOS) or Program Files (Windows).

Basic Knowledge

How to execute the code

Set the cursor at head of text or within the code you will execute, then hit the key **Cmd+Enter(macOS) Ctrl+Enter(Windows).**

Trouble

If it is trouble not able to stop the sound hit the key **Cmd+Period(macOS) Ctrl+Period(Windows).**

These are minimum knowledge to use SuperCollider but some features and functions are described in this manual later. if you need more reference and would like to read this manual by referring to the documentation, you can see a more detailed tutorial and info in the link below.

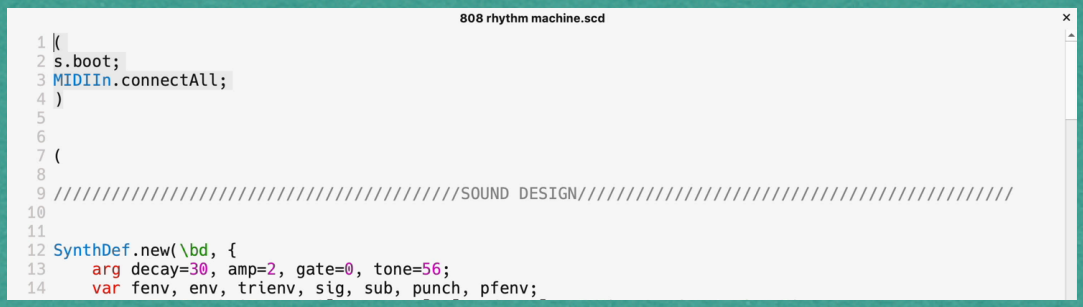
If you are beginner highly recommend watch this tutorials
Eli Fieldsteel's Tutorials Playlist : https://www.youtube.com/playlist?list=PLPYzvS8A_rTaNDweXe6PX4CXSGq4iEWYC

Official Documents -> http://doc.sccode.org/Tutorials/Getting-Started/00-Getting-Started-With-SC.html

Launching the Server and SC-808

1. Launch the Application SuperCollider(macOS) scide(Windows).
2. Select and Open the 808 files. File -> Open -> 808.scd
3. Booting the Server.

   Check the cursor at head of code booting a server.



   Then, hit the key Cmd+Enter(macOS) Ctrl+Enter(Windows).

4. Launch the 808

   Check the cursor at head of 808 code.

Then, hit the key Cmd+Enter(macOS) Ctrl+Enter(Windows).

```
(

////////////////////////////////////////SOUND DESIGN////////////////////////////////////////////////


SynthDef.new(\bd, {
    arg decay=30, amp=2, gate=0, tone=56;
    var fenv, env, trienv, sig, sub, punch, pfenv;
    env = EnvGen.kr(Env.new([0.4, 1, 0], [0, decay], -225),gate,doneAction:2);
    trienv = EnvGen.kr(Env.new([0.4, 0.6, 0], [0, decay], -230),gate,doneAction:2);
    fenv = Env([tone*14, tone*1.25, tone], [0.05, 0.6], -14).kr;
    pfenv = Env([tone*14, tone*1.25, tone], [0.03, 0.6], -10).kr;
    sig = SinOsc.ar(fenv, pi/2) * env;
    sub = LFTri.ar(fenv, pi/2) * trienv * 0.05;
    punch = SinOsc.ar(pfenv, pi/2) * env * 2;
    punch = HPF.ar(punch, 350);
    sig = (sig + sub + punch) * 2.5;
    sig = Limiter.ar(sig, 0.5) * amp;
    sig = Pan2.ar(sig, 0);
    Out.ar(0, sig);
}).add;

SynthDef.new(\sn, {
    arg amp=2, tone=340, tone2=189, snappy=0.3, gate=0, amp2=1;
    var noiseEnv, atkEnv, sig, noise, osc1, osc2, sum;
    noiseEnv = EnvGen.kr(Env.perc(0.001, 4.2, 1, -115), gate, doneAction:2);
    atkEnv = EnvGen.kr(Env.perc(0.001, 0.8,curve:-95), gate, doneAction:2);
    noise = WhiteNoise.ar;
    noise = HPF.ar(noise, 1800);
    noise = LPF.ar(noise, 8850);
    noise = noise * noiseEnv * snappy;
    osc1 = SinOsc.ar(tone2, pi/2) * 0.6;
    osc2 = SinOsc.ar(tone, pi/2) * 0.7;
    sum = (osc1+osc2) * atkEnv * amp2;
    sig = Pan2.ar((noise + sum) * amp * 2.5, 0);
    sig = HPF.ar(sig, 340);
    Out.ar(0, sig);
}).add;

SynthDef.new(\cpRaw, {
    arg amp=1, gate=0;
    var atkenv, atk, decay, sum, denv;
    atkenv = EnvGen.kr(Env.new([0.5,1,0],[0, 0.3], -160), gate, doneAction:0);
    denv = EnvGen.kr(Env.dadsr(0.026, 0, 6, 0, 1, 1, curve:-157), gate, doneAction:2);
    atk = WhiteNoise.ar * atkenv * 1.4;
    decay = WhiteNoise.ar * denv;
    sum = atk + decay * amp;
    sum = HPF.ar(sum, 500);
    sum = BPF.ar(sum, 1062, 0.5);
    Out.ar(0, Pan2.ar(sum * 1.5, 0));
}).add;

SynthDef.new(\cpReverb, {
    arg amp=1, gate=0;
    var reverb, revgen;
    revgen = EnvGen.kr(Env.perc(0.1, 4, curve:-9), gate, doneAction:2);
    reverb = WhiteNoise.ar * revgen * 0.02;
    reverb = HPF.ar(reverb, 500);
    reverb = LPF.ar(reverb, 1000);
    Out.ar(0, Pan2.ar(reverb * amp, 0));
}).add;

SynthDef.new(\LT, {
    arg amp=1, freq=80;
    var sig, fenv, env, pnoise, penv;
    env = EnvGen.kr(Env.new([0.4, 1, 0], [0, 20], -250),doneAction:2);
    penv = EnvGen.kr(Env.new([0.6, 1, 0], [0, 30], -225),doneAction:2);
    fenv = Env([freq*1.25, freq*1.125, freq], [0.1, 0.5], -4).kr;
    sig = SinOsc.ar(fenv, pi/2) * env;
    sig = Pan2.ar(sig * amp * 3, 0);
    Out.ar(0, sig);
}).add;

SynthDef.new(\MT, {
    arg amp, freq=120;
    var sig, fenv, env;
    env = EnvGen.kr(Env.new([0.4, 1, 0], [0, 16], -250),doneAction:2);
    fenv = Env([freq*1.33333, freq*1.125, freq], [0.1, 0.5], -4).kr;
    sig = SinOsc.ar(fenv, pi/2);
```

Finally, You'll see a GUI. Click the Bass Drum Button or others, and if you heard a sound it is boot successfully. Congrats!

Please check an Audio Setting on System Preference then rebooting the server, If you don't hear the sound.

Pattern

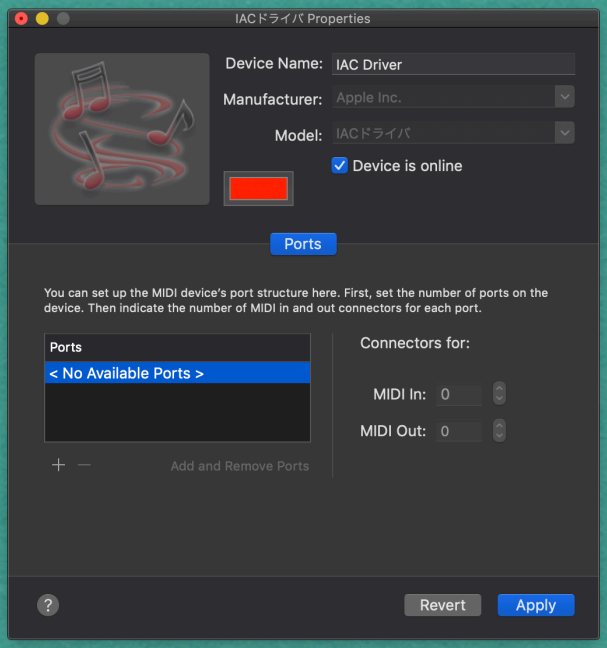
Making a pattern (or beat) is a significant feature for both original and this 808. This feature has not implemented on GUI for now, but some ways introduced below could build patterns more flexible and capable than hardware.

Controlled by DAW

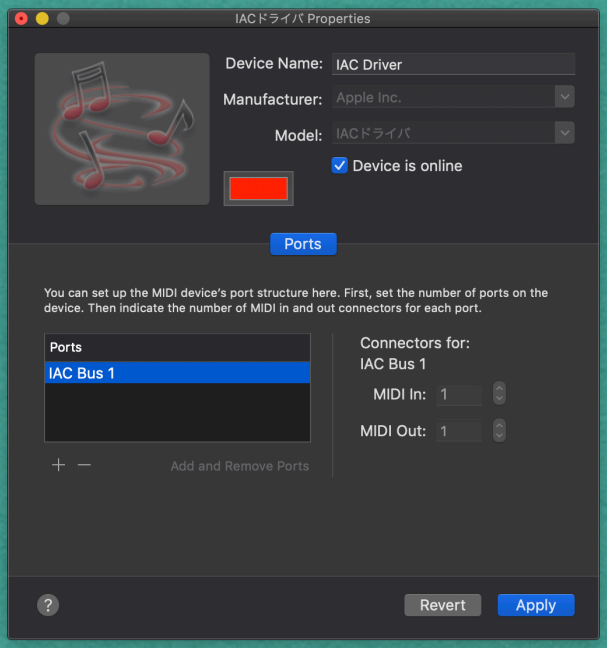
This way is most popular I guess. That uses internal routing of audio and midi, thus, necessary to install additionally the audio routing app like a Sound Flower and set up midi routing if you haven't these. This way is written from macOS perspective and be explained regard as have already been installed.

1. Open the Audio MIDI Setup.
2. Show MIDI Studio (Window -> Show MIDI Studio or Cmd+2)
3. Double click the IAC Devices and check the routing

    If a driver is no available (Picture 1), click the plus button to add IAC Driver then apply the preference. (Picture 2)

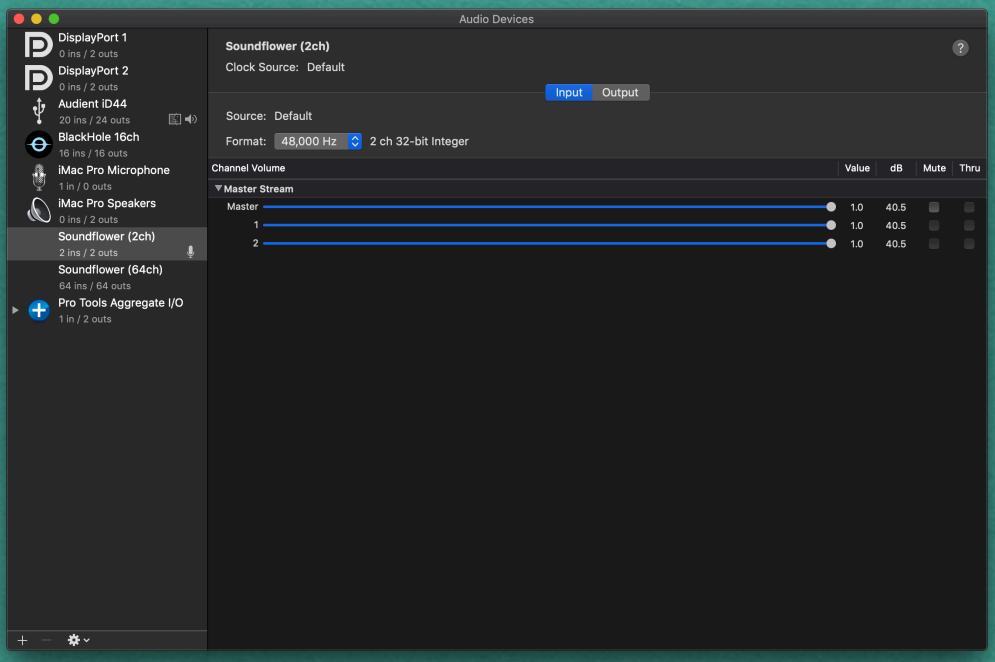


Picture 1

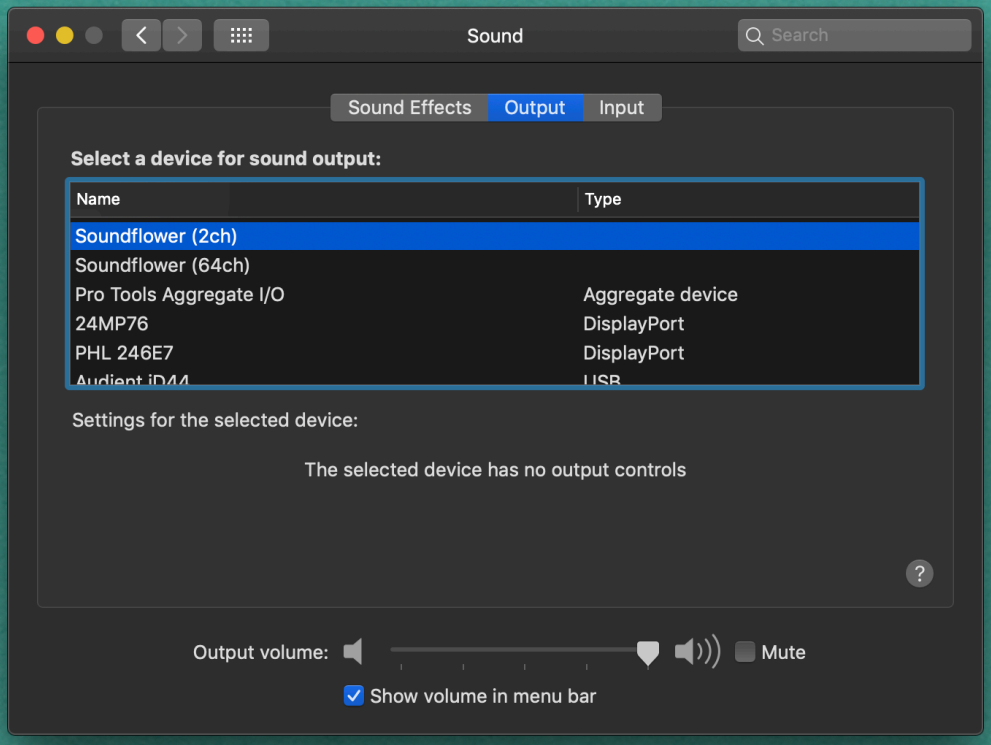


Picture 2

4. Show Audio Devices (Window -> Show Audio Devices or Cmd+1)
5. Choose internal audio device, set and select the properly sampling rate from “Format”. In this example, Chose a Soundflower (2ch) and set a sampling rate 48,000 Hz.



6. Setting the output device you chose on System Preference



Another Option

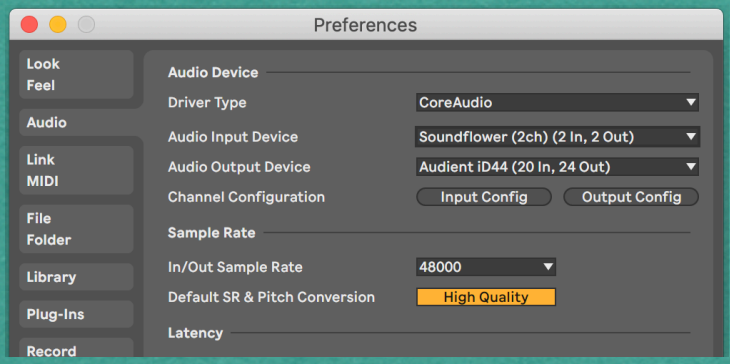
If you do not prefer changing system audio preference please execute the code below.

```
//option
s.options.outDevice_("Soundflower (2ch)"); //"audio device you use"
s.reboot;
```

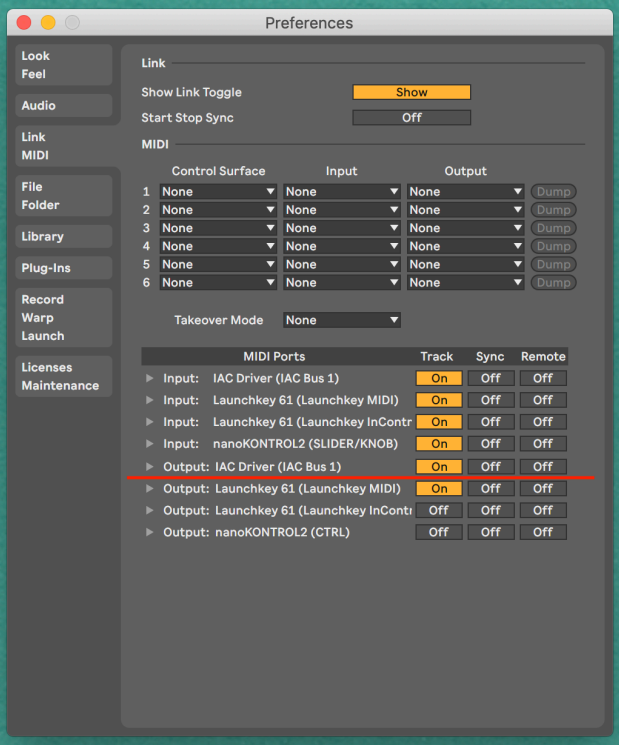
7. Launch a DAW and construct the routing

   Explaining the case of Ableton Live as an example, but that will be adaptable to any other DAW.

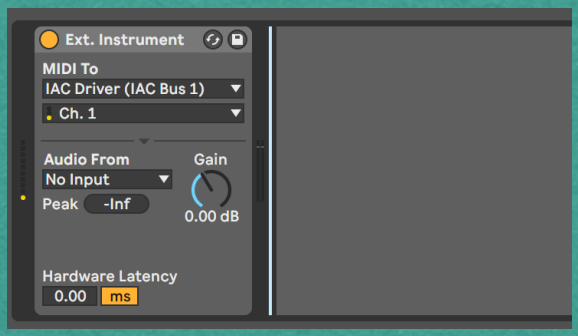
   Open the preference and set up an audio routing that input is an internal device and output is an audio interface you use.



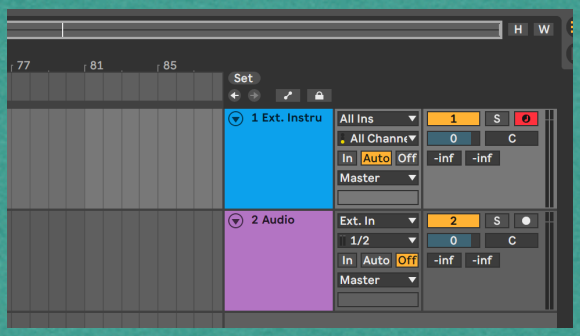
   Open the MIDI page of preference and turn an internal MIDI device active.



Insert a MIDI and an Audio Track, configure each track as following example.



MIDI Track: Set up to external instrument



Audio Track: Select a stereo input.

8. Launch and Boot a SuperCollider and 808.
   Finally, booting or rebooting a server then launch the 808.

It is ready for making a pattern, if these steps are done with no problem. The rest of a step is to make a beat on a MIDI Track as usual and listen to the sound returned from SuperCollider on Audio Track.

Each Instrument and Parameter is mapped to and controlled by MIDI notes and Control Change. Please see more MIDI Implement Chart.

On the SuperCollider

Of course, SuperCollider has a sequencer feature. It could build a complex pattern and which is used classes - Pbind, Pdef, Pseq, and so on, it is useful for Live Coding Performance particularly. Explaining those classes are omitted because there are many classes and official document and some Youtube tutorial are more informative and comprehensible than I would.

Youtube Tutorial I recommend
Eli Fieldsteel's Tutorial : https://youtu.be/nB_bVJ1c1Rg

Official Document of the Pattern : http://doc.sccode.org/Tutorials/Getting-Started/16-Sequencing-with-Patterns.html

Nevertheless, It includes the pattern example I create in the folder named **Pattern Example**. Please use it to learn by and experiment with changing each value.

Controlled by TidalCycles

TidalCycles is used live coding primary, it run by SuperDirt on SuperCollider and use another text editor such as ATOM. This is a stunning live coding tool!!

Please refer the official Installing instructions from https://tidalcycles.org/index.php/Installation

Usage Tutorials
Start tidalcycles and superdirt for the first time: https://tidalcycles.org/index.php/Start_tidalcycles_and_superdirt_for_the_first_time

Tutorial: https://tidalcycles.org/index.php/Tutorial

Additionally, you have to add 808 to SuerDirt and improve or change the code: Out.ar to OffsetOut.ar and Pan.ar to DirtPan.ar. Please see more here.

Code

The Code is exactly the core of an 808. Without codes no constructing it. Codes are divided into three sections - Sound Design, MIDI, and GUI. Described several times above, all of sections and elements of it are built by sclang that is interpreted programming language.

You may be feeling the difficultly like I felt once, however, if you have a basic knowledge of synthesis and MIDI, the impression feeling now will be changed soon. In this part, It will be explained more deeply as comprehensible as possible.

Let's dive into it.

Sound Design

In a Sound Design Section, emulate the process of generating an original sound. Keeping in mind and clearing the thing are this is SuperCollider depending on UGen like a module in short, thus it is impossible to perfect and strict emulating and modeling such as a Filter and an EQ Curve easily. However, it is devised for example combining with LPF, HPF and EQ instead of unique BPF.

Understanding the algorithm in this section is learning precious secrets of how original one works and synthesis. These ones will be opened up your sound design skill and the admiration and respect for Tadao Kikumoto's engineering team in 1980 are going to be deeply more.

This time, clarify the Bass Drum and Cymbal&HiHat algorithm as an example.
(before or while explaining them, highly recommend watching and referring the tutorial 3 & 4 from playlist link in above page if you beginner.)

Bass Drum

The principle generating is mainly by a sine wave that pitch is transitioned high to low. According to the service manual, fundamental frequency should set 56Hz approximately, in addition, Original one has capability changed a tuning by named tone. This feature is implemented by using the argument "tone".
Kick sound is generated by the process that determining a sine and another waves' frequency by pitch envelope transitioned, afterward, these sounds are summed and amplitude is controlled by envelope curve. Please see the flow char (Figure 1) and the whole of bass drum code.

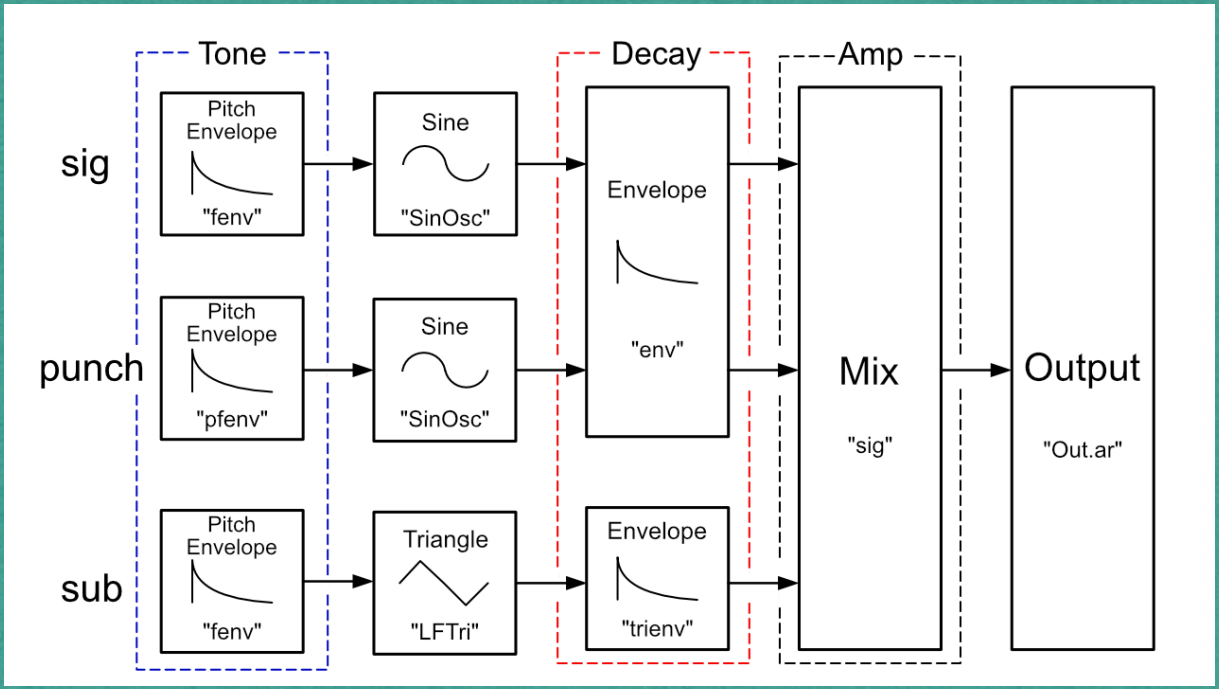


Figure1. Flow Chart of Bass Drum

```
SynthDef.new(\bd, {
	arg decay=30, amp=2, gate=0, tone=56;
	var fenv, env, trienv, sig, sub, punch, pfenv;
	env = EnvGen.kr(Env.new([0.11, 1, 0], [0, decay], -225),doneAction:
2);
	trienv = EnvGen.kr(Env.new([0.11, 0.6, 0], [0, decay],
-230),doneAction:0);
	fenv = Env([tone*7, tone*1.35, tone], [0.05, 0.6], -14).kr;
	pfenv = Env([tone*7, tone*1.35, tone], [0.03, 0.6], -10).kr;
	sig = SinOsc.ar(fenv, pi/2) * env;
	sub = LFTri.ar(fenv, pi/2) * trienv * 0.05;
	punch = SinOsc.ar(pfenv, pi/2) * env * 2;
	punch = HPF.ar(punch, 350);
	sig = (sig + sub + punch) * 2.5;
	sig = Limiter.ar(sig, 0.5) * amp;
	sig = Pan2.ar(sig, 0);
	Out.ar(0, sig);
}).add;
```

Bass Drum Code

To clarification, separated the code. Firstly, explain the instrument name, argument, and variable. Which are called declare but can also are said that preparation for the construction.

```
SynthDef.new(\bd, {
	arg decay=30, amp=2, gate=0, tone=56;
	var fenv, env, trienv, sig, sub, punch, pfenv;
```

The definition of instrument name is used \ or " ". In this case, as you can see, it is named \bd. Also please remember the instrument name would be important when you set up the MIDI, GUI action, and making a pattern on SC.

In the term of the argument, declare the argument, in particular, you'd like to control, by MIDI, GUI, and something. In the bass drum, as described above, Tone and Decay are characterized its sound. Therefore it is necessary to declare the variable or with value; This case uses the "decay" and "tone", concomitantly, which is assigned the value to each variable. Moreover, it is added argument "amp" and "gate" which is to volume and envelope.

The declaration of the variable to use in the code is the almost same way of argument. Variables using on it: fenv, pfenv, env, trienv, sig, sub, and punch, are declared and set.

Secondary, describe the envelope. Which is able to be divided into two, the one is for pitch transition, another is amplitude, specifically speaking, "env" and "trienv" are for amp and decay and "fenv" and "pfenv" are for pitch and tone. Please see a code and each specification of envelope curve. (Figure 2)

```
      env = EnvGen.kr(Env.new([0.11, 1, 0], [0, decay], -225),doneAction:
2);
      trienv = EnvGen.kr(Env.new([0.11, 0.6, 0], [0, decay],
-230),doneAction:0);
      fenv = Env([tone*7, tone*1.35, tone], [0.05, 0.6], -14).kr;
      pfenv = Env([tone*7, tone*1.35, tone], [0.03, 0.6], -10).kr;
```

Analyze with more attention, all envelopes include variable declared on the argument, which means depending on the variable's value. In other words, are able to characterize a sound.

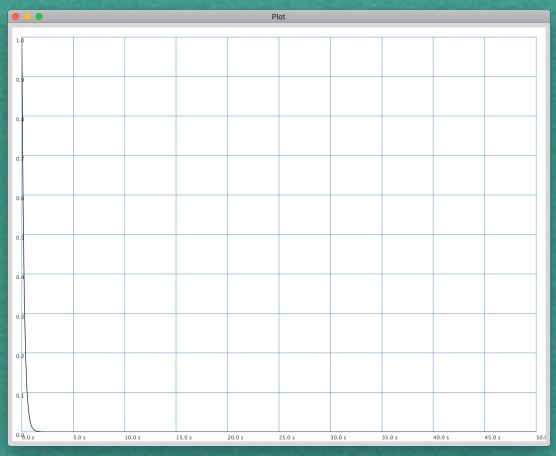


Figure 2a. env

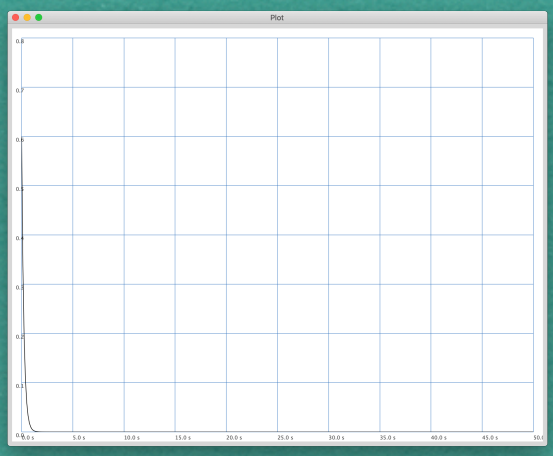


Figure 2b. trienv

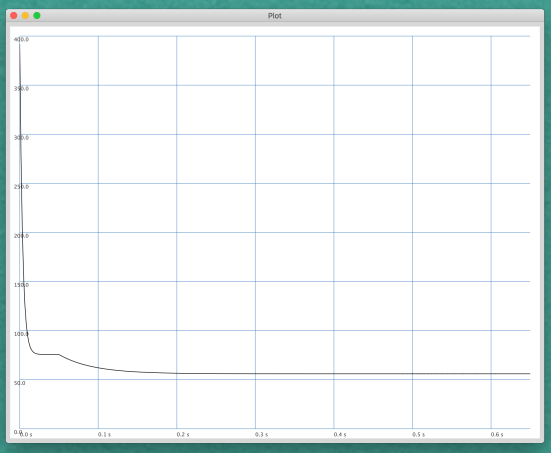


Figure 2c. fenv

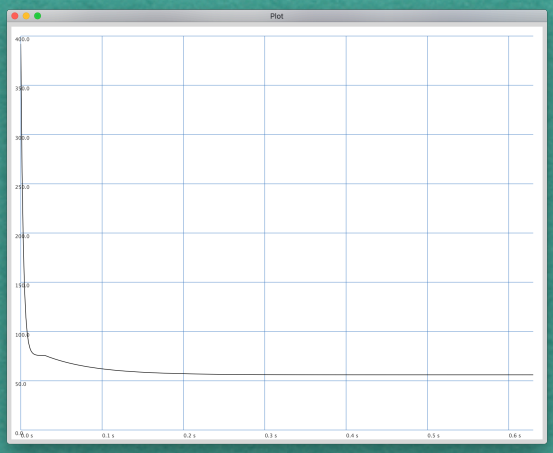


Figure 2d. pfenv

Finally, explain the rest. That is part of constructing an instrument with elements prepared before. The thought of this part is similar to the modular synth, I think. "sig" is the main audio signal, and it uses a sine wave oscillator. That is assigned "fenv" pitch envelope, to the frequency of "SinOsc.ar" and is set the phase to pi/2 to get a more punchy attack (Please refer the reason why is from here). Those actions are similar to input the cv signal to the oscillator module. And also "sig" audio transition is controlled by env, which actions similar to the gate.

```
        sig = SinOsc.ar(fenv, pi/2) * env;
        sub = LFTri.ar(fenv, pi/2) * trienv * 0.05;
        punch = SinOsc.ar(pfenv, pi/2) * env * 2;
        punch = HPF.ar(punch, 350);
        sig = (sig + sub + punch) * 2.5;
        sig = Limiter.ar(sig, 0.5) * amp;
        sig = Pan2.ar(sig, 0);
        Out.ar(0, sig);
}).add;
```

"sub" and "punch" is used the same algorithm of "sig". the signal of "sub" is literally for sub bass and "punch" signal is to add a punchy attack thus "pfenv" is set an attack time a little bit faster than "fenv".
After all, These three signals are summed in "sig", then set the panning to center. If it was not set the panning you hear the sound from left because these signal are mono.

Tom and Conga are applied these methods.

Cymbal & Close HiHat

The structure of close hihat and cymbal is based on six oscillators, which have nominal frequency (or ranges) of 205.3, 369.6, 304.4, 522.7, 359.4–1149.9, and 254.3–627.2 Hz (Werner, K. J., Abel, J., & Smith, J. (2014). The TR-808 Cymbal: a Physically-Informed, Circuit-Bendable, Digital Model). The 5 and 6 oscillators also have the role of cowbell and according to the service manual, these two should set 800 and 540 Hz approximately.

The generating process of Cymbal is six oscillators sound are summed then divided into two signals, one of two is passed through BPF for mid-range and the other is passed through BPF for high-range and this signal is divided into two. Thus at this point, there are three signals, which are also passed through three different types of HPF for each signal, in addition, one of two HPF for high-range are divided into two as well and it has four signals at this point. These are assigned four different types of envelopes including which is to control "decay". These four signals are summed ultimately and output. Mentioning the controlling of "Tone", it is adjusted by volume level of the signal pass through BPF for mid-range and HPF.

Close hihat is constructed by the element and method of the cymbal. Six oscillators generate the sound and are summed. The dissimilar point of between hihat and cymbal is signal is divided complex or not. Hihat is undivided after passed through summing and BPF. Thus it has only one signal, and this is processed in order of BPF, HPF, and Envelope.

Please see the flow chart (Figure 3) and the whole of cymbal code.

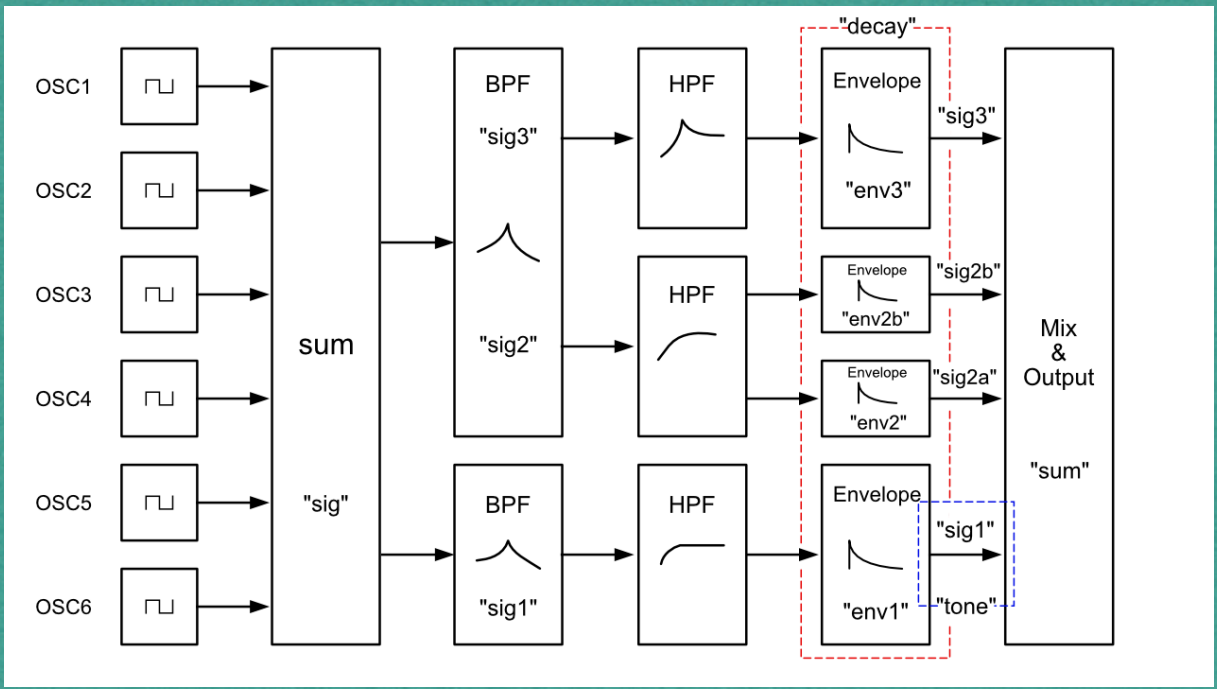


Figure 3. Flow Chart of Cymbal

```
SynthDef.new(\cymbal, {
	arg decay=2, amp=20, pan=0, gate=0, tone=0.002;
	var sig, sig1,sig2, sig2a, sig2b, sig3, env1, env2, env2b, env3,
osc1, osc2, osc3, osc4, osc5, osc6, sum;
	env1 = EnvGen.kr(Env.perc(0.3, decay, curve:-3), doneAction:2);
	env2 = EnvGen.kr(Env.new([0, 0.6, 0], [0.1, decay*0.7], -5),
doneAction:0);
	env2b = EnvGen.kr(Env.new([0, 0.3, 0], [0.1, decay*20], -120),
doneAction:0);
	env3 = EnvGen.kr(Env.new([0, 1, 0], [0, decay*5], curve:-150),
doneAction:0);
	osc1 = LFPulse.ar(203.52) * 0.6;
	osc2 = LFPulse.ar(366.31) * 0.6;
	osc3 = LFPulse.ar(301.77) * 0.6;
	osc4 = LFPulse.ar(518.19) * 0.6;
	osc5 = LFPulse.ar(811.16) * 0.6;
	osc6 = LFPulse.ar(538.75) * 0.6;
	sig = osc1 + osc2 + osc3 + osc4 + osc5 +osc6;
	sig1 = BLowShelf.ar(sig, 2000, 1, 5);
	sig1 = BPF.ar(sig1, 3000);
	sig1 = BPeakEQ.ar(sig1, 2400, 0.5, 5);
	sig1 = BHiPass.ar(sig1, 1550, 0.7);
	sig1 = LPF.ar(sig1, 3000);
	sig1 = BLowShelf.ar(sig1, 1000, 1, 0);
	sig1 = sig1 * env1 * tone;
	sig2 = BLowShelf.ar(sig, 990, 2, -5);
	sig2 = BPF.ar(sig2, 7400);
	sig2 = BPeakEQ.ar(sig2, 7200, 0.5, 5);
	sig2 = BHiPass4.ar(sig2, 6800, 0.7);
	sig2 = BHiShelf.ar(sig2, 10000, 1, -4);
	sig2a = sig2 * env2 * 0.3;
	sig2b = sig2 * env2b * 0.6;
	sig3 = BLowShelf.ar(sig, 990, 2, -15);
	sig3 = BPF.ar(sig3, 6500);
	sig3 = BPeakEQ.ar(sig3, 7400, 0.35, 10);
	sig3 = BHiPass4.ar(sig3, 10500, 0.8, 2);
	sig3 = sig3 * env3;
	sum = sig1 + sig2a + sig2b + sig3;
	sum = LPF.ar(sum, 4000);
	sum = Pan2.ar(sum, 0);
	sum = sum * amp;
	Out.ar(0, sum);
}).add;
```

Mentioned above, Cymbal sound is determined by tone, decay, and also volume level. Therefore it is necessary to declare the argument of "tone", "decay", "amp", and so on. The variable is declared as well. Please see more arguments and variables used on the cymbal, from the separated code below.

```
SynthDef.new(\cymbal, {
	arg decay=2, amp=20, pan=0, gate=0, tone=0.002;
	var sig, sig1,sig2, sig2a, sig2b, sig3, env1, env2, env2b, env3,
osc1, osc2, osc3, osc4, osc5, osc6, sum;
```

Explain about envelopes. Cymbal is applied four envelopes, "env1" is for long decay of core of tone, "env2" is for long decay of high-range, "env2b" is for fast attack and short decay of high-range and "env3" is for punchy attack sound of high-range. Please see the code and specifications of the envelope curve (Figure 4).

```
      env1 = EnvGen.kr(Env.perc(0.3, decay, curve:-3), doneAction:2);
      env2 = EnvGen.kr(Env.new([0, 0.6, 0], [0.1, decay*0.7], -5),
doneAction:0);
      env2b = EnvGen.kr(Env.new([0, 0.3, 0], [0.1, decay*20], -120),
doneAction:0);
      env3 = EnvGen.kr(Env.new([0, 1, 0], [0, decay*5], curve:-150),
doneAction:0);
```

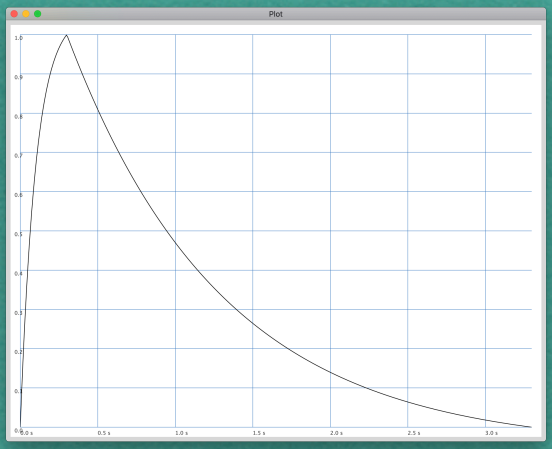


Figure 4a. env1

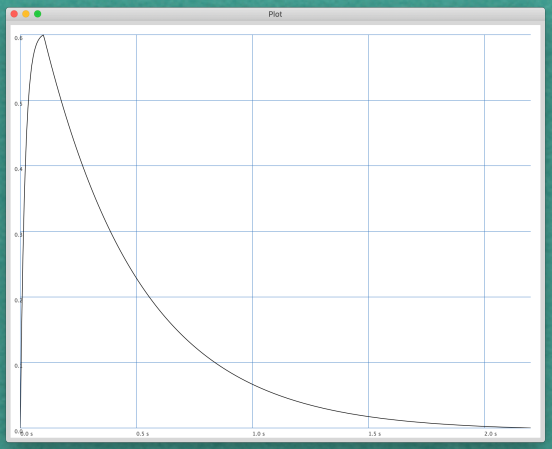


Figure 4b. env2

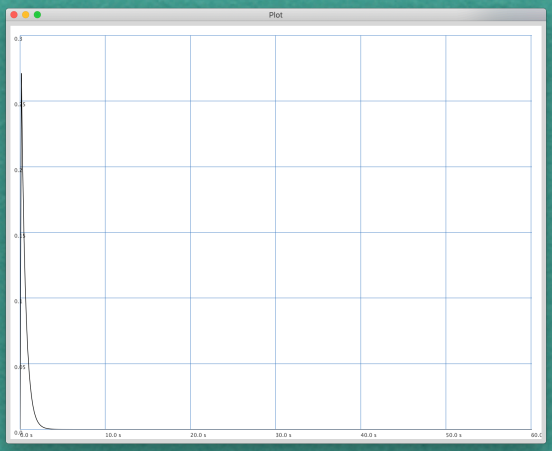


Figure 4c. env2b

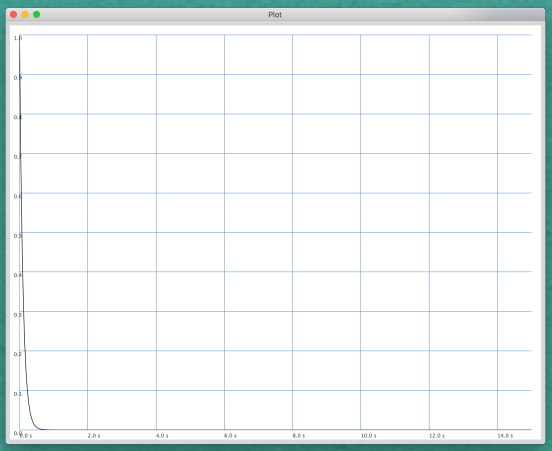


Figure 4d. env3

Six oscillators of square waveform is set and summed to "sig". In this explanation, focus to the elucidation of frequency relations of each one.
Different frequencies are applied to each oscillator. This 808's frequency value is different compare to mentioned above, however, essential mechanism is same.

```
osc1 = LFPulse.ar(203.52) * 0.6;
osc2 = LFPulse.ar(366.31) * 0.6;
osc3 = LFPulse.ar(301.77) * 0.6;
osc4 = LFPulse.ar(518.19) * 0.6;
osc5 = LFPulse.ar(811.16) * 0.6;
osc6 = LFPulse.ar(538.75) * 0.6;
sig = osc1 + osc2 + osc3 + osc4 + osc5 +osc6;
```

These frequency values look like a randomly. Organizing these values and analyze the relation of these with following formula.
In this explanation, using the values introduced above.

Organize

```
osc1 = LFPulse.ar(205.2);
osc3 = LFPulse.ar(304.4);
osc2 = LFPulse.ar(369.6);
osc4 = LFPulse.ar(522.7);
osc6 = LFPulse.ar(540);
osc5 = LFPulse.ar(800);
```

Six oscillators are sort in order of the value is small, divide them into three groups of (A)osc1&osc3, (B)osc2&osc4, and (C)osc6&osc5.

Formula
a and b are substituted frequency value (Hz), n of cent the differential value between a and b.

$$n = 1200 \cdot \log_2\left(\frac{b}{a}\right)$$

Substitute the values of (A), (B), and (C) for the equation. Please use functions Calculator if you need.

(A)

$$n = 1200 \cdot \log_2\left(\frac{304.4}{205.3}\right)$$
$$n = 681.88...$$

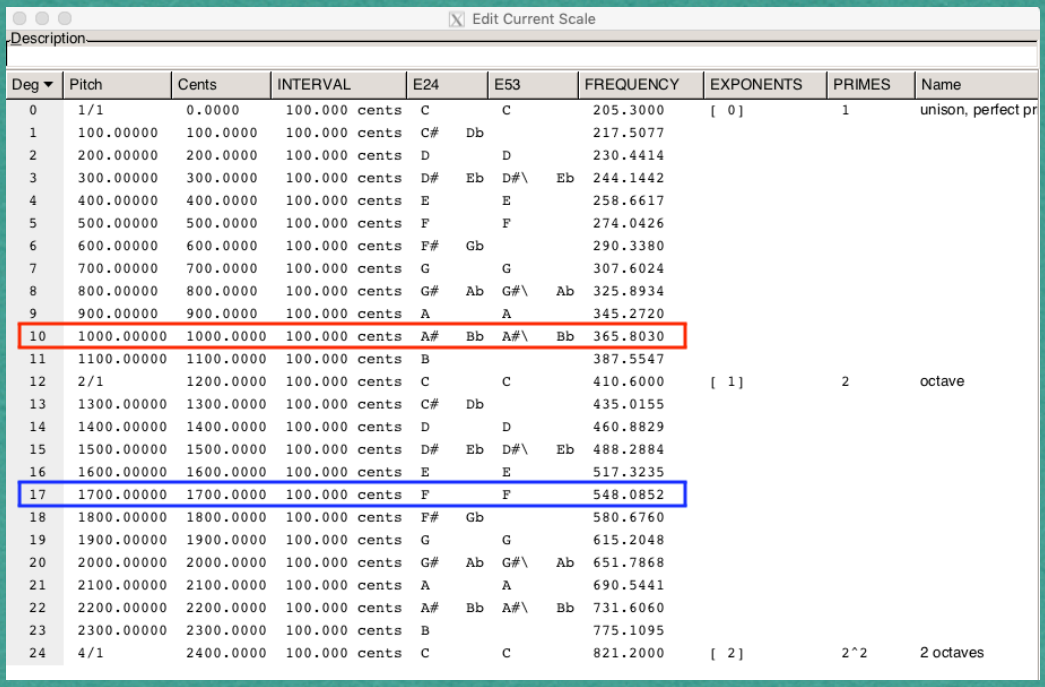
(B)

$$n = 1200 \cdot \log_2\left(\frac{522.7}{369.6}\right)$$
$$n = 600.02...$$

(C)

$$n = 1200 \cdot \log_2\left(\frac{800}{540}\right)$$
$$n = 680.44...$$

The result suggests that cent of (A) and (B) have a range of 680.44 to 681.88 cents and which are close to the perfect fifth (700 cents), the sound close to hamonic. (C) is almost the Tritone or diminished fifth (600 cents). Additionally, it is necessary to make sure if (B) and (C) are on (A) tuning table. What if these are on (A), the total sound is "harmonical" so far away from the metallic sound constructed by inharmonic.
First of all, assigned 205.3 Hz of (A) to C temporarily and make a 12 equal temperament tuning table based on that, with Scala. (Figure 5)


Edit Current Scale

Description

Deg	Pitch	Cents	INTERVAL	E24	E53	FREQUENCY	EXPONENTS	PRIMES	Name
0	1/1	0.0000	100.000 cents	C	C	205.3000	[0]	1	unison, perfect pr
1	100.00000	100.0000	100.000 cents	C# Db		217.5077			
2	200.00000	200.0000	100.000 cents	D	D	230.4414			
3	300.00000	300.0000	100.000 cents	D# Eb	D#\ Eb	244.1442			
4	400.00000	400.0000	100.000 cents	E	E	258.6617			
5	500.00000	500.0000	100.000 cents	F	F	274.0426			
6	600.00000	600.0000	100.000 cents	F# Gb		290.3380			
7	700.00000	700.0000	100.000 cents	G	G	307.6024			
8	800.00000	800.0000	100.000 cents	G# Ab	G#\ Ab	325.8934			
9	900.00000	900.0000	100.000 cents	A	A	345.2720			
10	1000.00000	1000.0000	100.000 cents	A# Bb	A#\ Bb	365.8030			
11	1100.00000	1100.0000	100.000 cents	B		387.5547			
12	2/1	1200.0000	100.000 cents	C	C	410.6000	[1]	2	octave
13	1300.00000	1300.0000	100.000 cents	C# Db		435.0155			
14	1400.00000	1400.0000	100.000 cents	D	D	460.8829			
15	1500.00000	1500.0000	100.000 cents	D# Eb	D#\ Eb	488.2884			
16	1600.00000	1600.0000	100.000 cents	E	E	517.3235			
17	1700.00000	1700.0000	100.000 cents	F	F	548.0852			
18	1800.00000	1800.0000	100.000 cents	F# Gb		580.6760			
19	1900.00000	1900.0000	100.000 cents	G	G	615.2048			
20	2000.00000	2000.0000	100.000 cents	G# Ab	G#\ Ab	651.7868			
21	2100.00000	2100.0000	100.000 cents	A	A	690.5441			
22	2200.00000	2200.0000	100.000 cents	A# Bb	A#\ Bb	731.6060			
23	2300.00000	2300.0000	100.000 cents	B		775.1095			
24	4/1	2400.0000	100.000 cents	C	C	821.2000	[2]	2^2	2 octaves

Figure 5. 12 equal temperament based on 205.3 Hz of (A)

(B) and (C) frequencies don't exist anywhere on (A) tuning table. This result indicates (B) and (C) are on another tuning table. However, there are values that are close to (B) and (C); (B) is marked by a red frame, and (C) is marked by a blue frame (Figure5).
Calculate the distance from (A) to (B) and (C) with values that are marked and (B) and (C), and the above formula.

The distance from (A) to (B)

$$n = 1200 \cdot \log_2 \left(\frac{369.6}{365.8} \right)$$

$$n = 17.89...$$

The distance from (A) to (C)

$$n = 1200 \cdot \log_2 \left(\frac{548}{540} \right)$$

$$n = 25.45...$$

These results indicate that each group has the original scale which is different from each other, the little difference of that is significant to the coexistence of a dissonant of metallic sound, and musically sounds. (Figure 6)
Please see more how to retune these on the chapter of the modification.

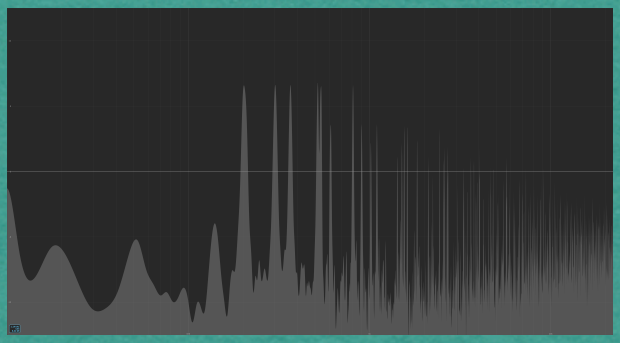


Figure 6a.
The six oscillators signals ("sig") that are before passed through filters.

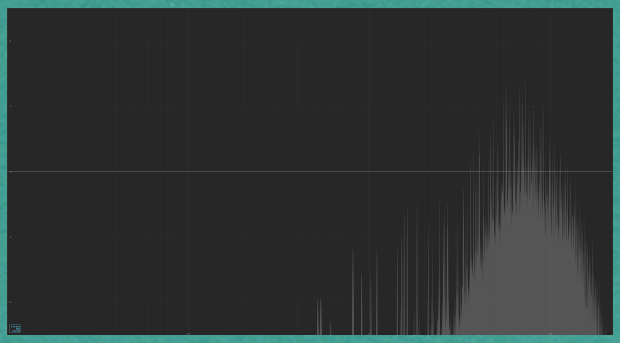


Figure 6b.
The six oscillators signals ("sum") that are after passed through filters.

After six oscillator sounds are summed, a signal is divided into four eventually. First, explain the signal of the core of a tone. It passes through the BPF set around 3000 Hz and PeakEQ boosted at 2400 Hz improving this BPF. After that it through HPF cutting under 1500Hz, then cut the frequency upper than 3000Hz and boot up the around 1000Hz by Lowshelf. As you know this signal focus to mid-range. Finally, this signal is assigned "env1" and "tone" that is controlling the "sig1" volume level.

```
sig1 = BLowShelf.ar(sig, 2000, 1, 5);
sig1 = BPF.ar(sig1, 3000);
sig1 = BPeakEQ.ar(sig1, 2400, 0.5, 5);
sig1 = BHiPass.ar(sig1, 1550, 0.7);
sig1 = LPF.ar(sig1, 3000);
sig1 = BLowShelf.ar(sig1, 1000, 1, 0);
sig1 = sig1 * env1 * tone;
```

"sig2" is for high-range and has two roles, one is for the attack, the other is the long decay. It has the same progress to "sig1" but BPF and HPF are set the frequency higher than "sig1". The BPF is set around 7400Hz and HPF is set around 7000Hz. Eventually, it divided into "sig2a" and "sig2b", "sig2a" is for a long decay and assigned the "env2". "sig2b" is for an attack and assigned the "env2b".

```
sig2 = BLowShelf.ar(sig, 990, 2, -5);
sig2 = BPF.ar(sig2, 7400);
sig2 = BPeakEQ.ar(sig2, 7200, 0.5, 5);
sig2 = BHiPass4.ar(sig2, 6800, 0.7);
sig2 = BHiShelf.ar(sig2, 10000, 1, -4);
sig2a = sig2 * env2 * 0.3;
sig2b = sig2 * env2b * 0.6;
```

The role of "sig3" is generating the sound of fast and punch attacks. It passes through the BPF set around 6500Hz and HPF that is set 10500Hz, the frequency is higher than "sig2". It assigned the "env3" that gave the fastest attack of all.

```
      sig3 = BLowShelf.ar(sig, 990, 2, -15);
      sig3 = BPF.ar(sig3, 6500);
      sig3 = BPeakEQ.ar(sig3, 7400, 0.35, 10);
      sig3 = BHiPass4.ar(sig3, 10500, 0.8, 2);
      sig3 = sig3 * env3;
```

These four signals are summed into one again, after that, it passes through LPF to improve a summed sound of high-range. Finally, setup the panning to center and assigned argument "amp" to control the volume with velocity and GUI knob.

```
      sum = sig1 + sig2a + sig2b + sig3;
      sum = LPF.ar(sum, 4000);
      sum = Pan2.ar(sum, 0);
      sum = sum * amp;
      Out.ar(0, sum);
}).add;
```

Close HiHat

The generating process of close hihat is more simple than cymbal because its signal is undivided than a cymbal. Its element is similar to cymbal as well, for example, it uses the same of cymbal's six oscillators. First, check the whole of close hihat code, argument, and variable.

```
SynthDef.new(\hat, {
      arg decay=0.42, amp=1, pan=0;
      var sig, sighi,siglow, sum, env, osc1, osc2, osc3, osc4, osc5, osc6;
      env = EnvGen.kr(Env.perc(0.005, decay, 1, -30),doneAction:2);
      osc1 = LFPulse.ar(203.52);
      osc2 = LFPulse.ar(366.31);
      osc3 = LFPulse.ar(301.77);
      osc4 = LFPulse.ar(518.19);
      osc5 = LFPulse.ar(811.16);
      osc6 = LFPulse.ar(538.75);
      sighi = (osc1 + osc2 + osc3 + osc4 + osc5 + osc6);
    siglow = (osc1 + osc2 + osc3 + osc4 + osc5 + osc6);
    sighi = BPF.ar(sighi, 8900, 1);
    sighi = HPF.ar(sighi, 9000);
    siglow = BBandPass.ar(siglow, 8900, 0.8);
    siglow = BHiPass.ar(siglow, 9000, 0.3);
    sig = BPeakEQ.ar((siglow+sighi), 9700, 0.8, 0.7);
    sig = sig * env * amp;
    sig = Pan2.ar(sig, pan);
    Out.ar(0, sig);
}).add;
```

As hi hat's volume is controlled by velocity and GUI, It is necessary to declare the argument "amp". Additionally, declare variables "sighi" and "siglow" because it is divided into two signals.

```
SynthDef.new(\hat, {
	arg rel=0.42, amp=1, pan=0;
	var sig, sigh, sigl, sum, env, osc1, osc2, osc3, osc4, osc5, osc6;
```

Hithat is used one envelope (Figure 7) and also uses six oscillators.

```
	env = EnvGen.kr(Env.perc(0.005, rel, 1, -30),doneAction:2);
	osc1 = LFPulse.ar(203.52);
	osc2 = LFPulse.ar(366.31);
	osc3 = LFPulse.ar(301.77);
	osc4 = LFPulse.ar(518.19);
	osc5 = LFPulse.ar(811.16);
	osc6 = LFPulse.ar(538.75);
```

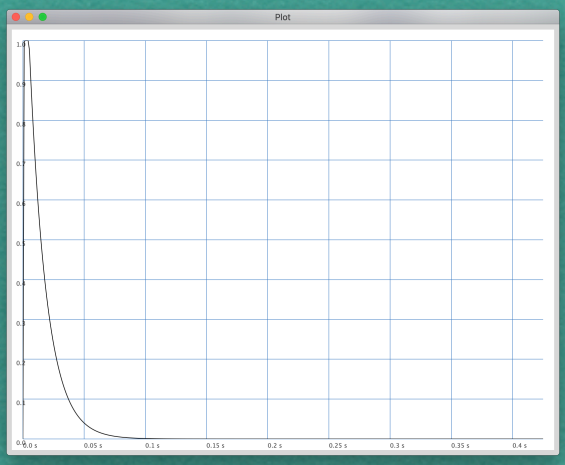


Figure 7. env

The signal of summed of six is divided into two, one is for more high range "sighi", the other is "siglow" focusing the range a bit under than "sighi". Two signals' BPF and HPF frequency value is same but the curve spec is difference.

```
	sighi = (osc1 + osc2 + osc3 + osc4 + osc5 + osc6);
	siglow = (osc1 + osc2 + osc3 + osc4 + osc5 + osc6);
	sighi = BPF.ar(sighi, 8900, 1);
	sighi = HPF.ar(sighi, 9000);
	siglow = BBandPass.ar(siglow, 8900, 0.8);
	siglow = BHiPass.ar(siglow, 9000, 0.3);
```

Finally, "sigh" and "siglow" are summed in PeakEQ that boost 9700Hz to get more metallic sound, afterward, it is assigned "env" and "amp" to control the volume with velocity and GUI knob. Set the panning to center, then a sound output.

```
	sig = BPeakEQ.ar((siglow+sighi), 9700, 0.8, 0.7);
	sig = sig * env * amp;
	sig = Pan2.ar(sig, pan);
	Out.ar(0, sig);
}).add;
```

MIDI

In MIDI Section, Setting up mapping each instrument to each note. In addition, corresponding MIDI CC to the elements of characterizing sound. - decay, tone, or something.
As all instruments have same algorithm, this time, explain the Bass Drum setup as the example. First of all, make an array of the global variable to controlled by MIDI. In Bass Drum case, it is used the "~bd".

```
~bd = Array.fill(128,{nil});
```

Describe about MIDI setup and please see the whole of BD midi setup code below.
(before or while explaining them, highly recommend watching and referring the tutorial 9 from playlist link in above page if you beginner.)

```
MIDIdef.noteOn(\bdON, {
	arg vel, nn;
	[vel, nn].postln;
	~bdamp = vel.linexp(0, 127, 0.3, 2);
	~bd[nn] = Synth.new(\bd,
		[
			\amp, ~bdamp,
			\decay, ~decaybd,
			\tone, ~toneBD,
			\gate, 1
		]
	);
},36);

MIDIdef.noteOff(\bdOFF, {
	arg vel, nn;
	~bd[nn].set(\gate, 0);
	~bd[nn] = nil;
},36);

MIDIdef.cc(\bdDECAY, {
	arg val, cnum;
	[val, cnum].postln;
	~decaybd = val.linlin(0, 127, 6, 110);
	~bd.do({
		arg bdcc;
		bdcc.set(\decay, ~decaybd);
	});
},1);

MIDIdef.cc(\bdTONE, {
	arg val, cnum;
	[val, cnum].postln;
	~toneBD = val.linexp(0, 127, 50, 70);
	~bd.do({
		arg bdcc;
		bdcc.set(\tone, ~toneBD);
	});
},11);
```

Explain note-on and note-off action. The note-on part has also a role to connect global values and synth built above and its arguments, furthermore, the velocity action controlling volume level is made using global variable; "~bdamp". "~bdamp" has a range of 0.3 to 2.0dB between 0 to 127 bit.
Afterward, correspond global variable "~bd" to synth; \bd, in addition, correspond arguments - "amp", "decay", and "tone", to global variables "~bdamp", "~decaybd", and "~toneBD" which include be explain later.
The role of the note-off part is to clear the midi information including the velocity, decay, and so on that filled where at note-on part.
Additionally, these two part have a significant role; key mapping. Please see the number on the tail of each part, in this example, it is emphasized by the color red.

```
MIDIdef.noteOn(\bdON, {
	arg vel, nn;
	[vel, nn].postln;
	~bdamp = vel.linexp(0, 127, 0.3, 2);
	~bd[nn] = Synth.new(\bd,
		[
			\amp, ~bdamp,
			\decay, ~decaybd,
			\tone, ~toneBD,
			\gate, 1
		]
	);
},36);

MIDIdef.noteOff(\bdOFF, {
	arg vel, nn;
	~bd[nn].set(\gate, 0);
	~bd[nn] = nil;
},36);
```

This number means Bass Drum is assigned at MIDI note number 36 that is C2. You can assign instruments to the note number you'd like to.

Next is the Control Change (MIDI CC). That thought is similar to the note-on or off part. Control the sound with the global variable not explained above, and MIDI CC.
"\bdDECAY" is for decay literally, declare the global variable "~decaybd", which has a range of 6 to 110 seconds between 0 to 127 bit. Use the function "do" and correspond the global variable to argument \decay. which means apply the global variable's value to the instrument while midi info is on the array. In this case, that is described "~bd.do".
As the same as note-on and off part, the number on the tail of each part emphasized by the color red is MIDI CC Number. Thus BD decay is controlled by cc1 and that can assign another cc number you'd like to as well.

```
MIDIdef.cc(\bdDECAY, {
	arg val, cnum;
	[val, cnum].postln;
	~decaybd = val.linlin(0, 127, 6, 110);
	~bd.do({
		arg bdcc;
		bdcc.set(\decay, ~decaybd);
	});
},1);

MIDIdef.cc(\bdTONE, {
	arg val, cnum;
	[val, cnum].postln;
	~toneBD = val.linexp(0, 127, 50, 70);
	~bd.do({
		arg bdcc;
		bdcc.set(\tone, ~toneBD);
	});
},11);
```

"\bdTONE" is for tone literally, declare the global variable "~toneBD", which has a range of 50 to 70 Hz seconds between 0 to 127 bit. BD tone is controlled by cc11 and that can assign another cc number you'd like to as well. It is omitted to a detailed explanation because that algorithm is the same as \bdDECAY.

Please refer MIDI Implement Chart to full MIDI information.

GUI

GUI section is separated into two departments. One is to construct the GUI window and align knobs and buttons, the other is to apply the action of knobs and buttons to the argument to control a sound.
The department of constructing GUI is not explained in this part because it is not important for all of you to understand it. And also what if it is described in detail, you would be boring. However, it is mentioned in the modification part, if you'd like to customize a GUI, please see and refer to the modification of a GUI.

This time, focusing the GUI Action.
It has a significant role in controlling and setting the initial sound. The GUI action assigns the initial values of each knob to the arguments of each instrument when the sc808 is launched. Therefore if there is no action setting initial value it has no output signal until be assigned the value by GUI or MIDI CC.
That is described the Bass Drum as example in this part.

It is necessary to make an array as the same way as MIDI, but please use a different name to MIDI. This example uses the name "~bdGUI".

```
~bdGUI = Array.fill(128,{nil});
```

Please see the whole of the Bass Drum GUI Action code below. As you can see, it is similar to the thought of the MIDI section, thus it is omitted the explanation that was described at the MIDI section. However, there are some crucial elements that explanation is indispensable, this section is focused that.

```
~knobs[0].children[0].action_({
	arg obj;
	~bdamp = obj.value.linexp(0, 1, 1, 2.1);
	~bdGUI.do({
		arg amp;
		amp.set(\amp, ~bdamp);
	});
})
.valueAction_(0.5);

~knobs[1].children[0].action_({
	arg obj;
	~toneBD = obj.value.linexp(0, 1, 50, 70);
	~bdGUI.do({
		arg tone;
		tone.set(\tone, ~toneBD);
	});
})
.valueAction_(0);

~knobs[2].children[0].action_({
	arg obj;
	~decaybd = obj.value.linlin(0, 1, 6, 110);
	~bdGUI.do({
		arg decay;
		decay.set(\decay, ~decaybd);
	});
})
.valueAction_(0.5);

~buttons[0].children[0].mouseDownAction_({
	~bdGUI = Synth.new(\bd,
		[
			\amp, ~bdamp,
			\decay, ~decaybd,
			\tone, ~toneBD,
			\gate, 1
		]
	);
	~buttons2[0].children[0].states = [["•", Color.red,
Color.red(0.85)]];
})
.action_({
	~bdGUI = nil;
	~buttons2[0].children[0].states = [["•", Color.black,
Color.red(0.97)]];
});
```

"~knobs" number is rows and "children" is columns, ~knobs[0].children[1] means a knob is at the cross point of 1st row and 2nd column.
The knobs controlling the sound is also use global variable assigned the value to the argument. One of the most important points on the department of GUI Action is it uses the same global variable as MIDI CC. Otherwise, it doesn't work well sometimes.
In addition, there is a thing not described yet; that is function "valueAction" emphasized value color red. Which is to set initial value when launching the 808. That is, mentioned above, what if this function is not used, the audio signal is not outputted until be assigned the value. The different perspectives, that also said you can change that value and save it, which is able to make the default setting or presets.

```
~knobs[0].children[0].action_({
	arg obj;
	~bdamp = obj.value.linexp(0, 1, 1, 2.1);
	~bdGUI.do({
		arg amp;
		amp.set(\amp, ~bdamp);
	});
})
.valueAction_(0.5);

~knobs[1].children[0].action_({
	arg obj;
	~toneBD = obj.value.linexp(0, 1, 50, 70);
	~bdGUI.do({
		arg tone;
		tone.set(\tone, ~toneBD);
	});
})
.valueAction_(0);

~knobs[2].children[0].action_({
	arg obj;
	~decaybd = obj.value.linlin(0, 1, 6, 110);
	~bdGUI.do({
		arg decay;
		decay.set(\decay, ~decaybd);
	});
})
.valueAction_(0.5);
```

The button action is also similar to MID note on and off action.
"mouseDownAction" is MIDI note-on in other words, as same that, "action" function is also MIDI-off in different expression. Button's actions also work for changing the button color for the accent and flashing light as "・".

```
~buttons[0].children[0].mouseDownAction_({
	~bdGUI = Synth.new(\bd,
		[
			\amp, ~bdamp,
			\decay, ~decaybd,
			\tone, ~toneBD,
			\gate, 1
		]
	);
	~buttons2[0].children[0].states = [["•", Color.red,
Color.red(0.85)]];
})
.action_({
	~bdGUI = nil;
	~buttons2[0].children[0].states = [["•", Color.black,
Color.red(0.97)]];
});
```

That is all contents of the code and explained the basic knowledge of each section. If you have what you don't understand well please change the value to know how codes work like an I did two months ago.

Additionally, if you'd like to customize and modify, please refer to the next part.

Modification

Circuit bending is hard for most people and has the risk of the heritage becoming just a steel scrap. SC-808 could do that with no risk, and what if you failed to that you can back to the previous status. That is, you can try infinity and experimentally, and save it if you successes making the customized 808 for only you.
This Part, suggests a few modify example; The retuning of Close Hihat, Open-Hihat, Cymbal, and Cowbell and changing the GUI background color.

The Retuning of Close Hihat

Described above, Six oscillators have a musical frequency relation. Explaining the way change the frequency value keeping that balance. Of course, changing values randomly and intuitively is really interesting, but this explanation, focusing on how to ascent and descent the whole value proportionally.
To retune it, use the cent value and following formula. "b" is the result, a is currently frequency (Hz), and n is the cent. Use the functions calculator

$$b = a \times 2^{\frac{n}{1200}}$$

That is described using the nominal frequency value as an example, and this time, these are made 50 cents ascent (Quarter Tone); n = 50.

The case of **osc1**
Substitute the frequency value and the cent value; a=205.3, n=50.

$$b = 205.3 \times 2^{\frac{50}{1200}}$$

$$b = 211.31...$$

osc2 a=369.6

$$b = 380.43...$$

osc3 a=304.4

$$b = 313.31...$$

osc4 a=522.7

$$b = 538.01...$$

osc5 a=540

$$b = 555.82...$$

osc6 a=800

$$b = 823.44...$$

This method is adaptable to cowbell, in addition, changing the individual cent values to such as perfect fifth, calculate the frequency. The next example makes a cowbell sound the Pythagorean Perfect Fifth (701.955 cents).

Substitute the root frequency and cent value; a=540, n=701.955

$$b = 540 \times 2^{\frac{701.955}{1200}}$$

$$b = 809.99...$$

Even with only change the filter value, you'll get another characterized sound that like a be retuned.
Finally, after calculating that, don't forget the change the values of the code.

Changing the GUI Background Color

GUI Background Color is configured by RGB, controlled by the knob action.
The function "knob.action" has the range between 0 to 1, 0 is the color that knob is set at left, and 1 is the color that knob is set at right. RGB is the abbreviation of Red, Green, and Blue, each element is used global variables "~red", "~green", and "~blue" to control each value assigning to function ".background". Default background colors are Black that knob at left (initial) and Blue that knob at right. Please see the code of GUI Background Color, below.

```
Knob(~colorknobView[0], 135@135)
.color_(
	[
		Color.new(1, 0.45),
		Color.gray(0.85),
		Color.grey,
		Color.black
	]
)
.action_({
	arg obj;
	~red = obj.value.linlin(0, 1, 0.22, 0.27);
	~green = obj.value.linlin(0, 1, 0.22, 0.4);
	~blue = obj.value.linlin(0, 1, 0.22, 0.46);
	~gui.background_(Color.new(~red, ~green,~blue));
})
.valueAction_(0);
```

In this explanation, change the color blue that knob is set at right, to the green, use flowing values.

Red = 0.3 Green = 0.55 Blue = 0.25

Adopting these values above to the code, please see the code adopted and GUI (Figure 8).

```
.action_({
      arg obj;
      ~red = obj.value.linlin(0, 1, 0.22, 0.3);
      ~green = obj.value.linlin(0, 1, 0.22, 0.55);
      ~blue = obj.value.linlin(0, 1, 0.22, 0.25);
      ~gui.background_(Color.new(~red, ~green,~blue));
})
.valueAction_(0);
```

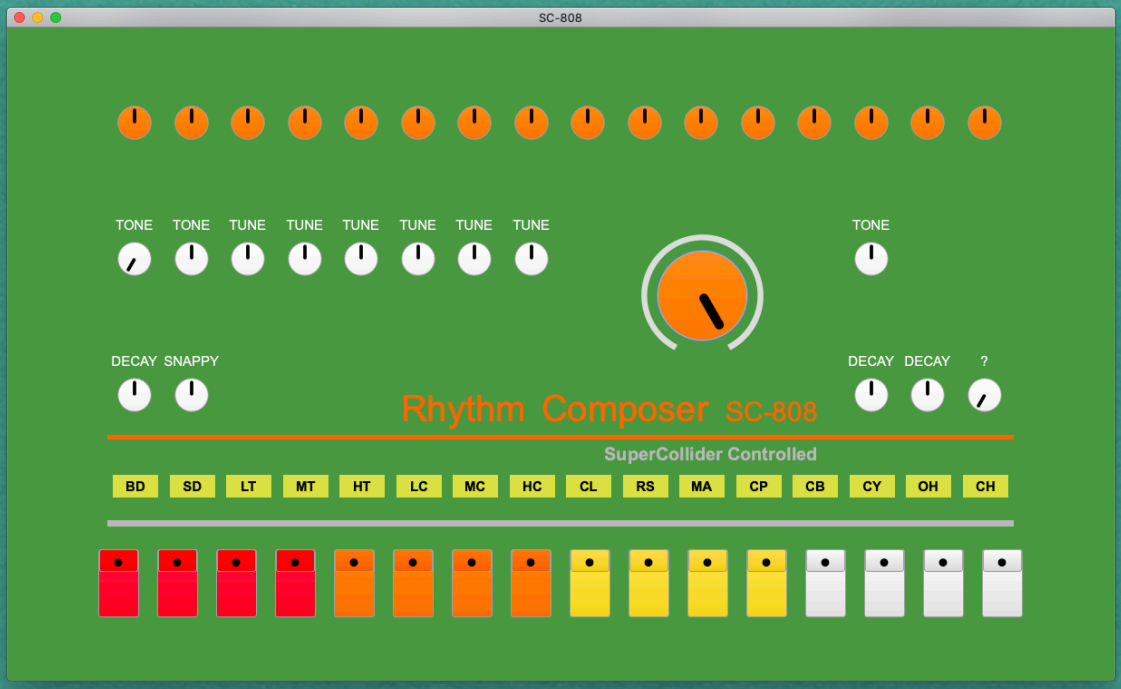


Figure 8. Green GUI

These modification examples are just one of the puzzle pieces. Please try finding and make your sound. And please show me your best one and hope you enjoy it!

Appendix

Tutorial Links

SuperCollider Official
http://doc.sccode.org/Browse.html#Tutorials

Eli Fieldsteel
https://www.youtube.com/playlist?list=PLPYzvS8A_rTaNDweXe6PX4CXSGq4iEWYC

Sound Engraver
https://www.youtube.com/playlist?list=PLKLdMcPt-nbLoutv1togicT35GPUWr3eo

TidalCycles
https://tidalcycles.org/index.php/Start_tidalcycles_and_superdirt_for_the_first_time

https://tidalcycles.org/index.php/Tutorial

Functions Calculator

$$n = 1200 \cdot \log_2\left(\frac{b}{a}\right)$$

https://www.symbolab.com/solver/functions-calculator/n%3D1200\cdot\log_{2}\left(\frac{b}{a}\right)

$$b = a \times 2^{\frac{n}{1200}}$$

https://www.symbolab.com/solver/functions-calculator/b%3Da\cdot2^{\frac{n}{1200}}

MIDI Implement Chart

Instrument and Knob	Note Number	Control Change
Level	Velocity	
Bass Drum	36	–
BD Decay	–	1
BD Tone	–	11
Snare	38	–
Snare Tone	–	21
Snare Snappy	–	22
Low Tom	41	–
Low Tom Tune	–	102
Mid Tom	43	–
Mid Tom Tune	–	103
Hi Tom	45	–
Hi Tom Tune	–	104
Low Conga	47	–
Low Conga Tune	–	105
Mid Conga	48	–
Mid Conga Tune	–	106
Hi Conga	50	–
Hi Conga Tune	–	107
Craves	52	–
Rim Shot	40	–
Maracas	51	–
Crap - Raw	39	–
Crap - Reverb	37	–
Cowbell	49	–
Cymbal	46	–
Cymbal Tone	–	109
Cymbal Decay	–	110
Open HiHat	44	–
Open HiHat Decay	–	108
Close Hihat	42	–