

連載

SUPER COLLIDER チュートリアル (4) SUPER COLLIDER TUTORIALS (4)

美山 千香士

Chikashi Miyama

ケルン音楽舞踏大学

Hochschule für Musik und Tanz Köln

概要

本連載では、リアルタイム音響合成環境の SuperCollider(SC) の使い方を、同ソフトを作品創作や研究のために利用しようと考えている音楽家、メディア・アーティストを対象にチュートリアル形式で紹介する。SuperCollider(SC) is a realtime programming environment for audio synthesis. This article introduces SC to musicians and media artists who are planning to utilize the software for their artistic creations and researches.

1. 今回の目標：SC でエフェクタを作成する

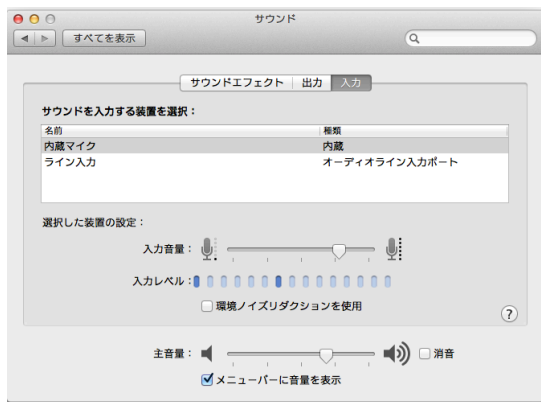


図 1. Mac のサウンドの設定

前回までは、SC を用いて音を生成する方法を学習してきたが、今回はマイクからの入力音をリアルタイムに加工する方法に焦点をあて、リング・モジュレーション、ディストーション、ピッチ・シフター、ディレイ、リバーブなどのエフェクタの作成をオムニバスの紹介していく。ハウリングを避けるために、本稿のプログラムを実行する際には、ヘッドフォンなどを着用すること。また、OS 側の設定で必ずマイクをオーディオ入力機器

に選択すること。Mac の場合は、図 1 のように、「環境設定」パネルの「サウンド」で内蔵マイクを OS のサウンド入力装置として設定し、SC サーバーの起動時にリスト 1 のように、内蔵マイク入力 (Built-in Microphone) がリストの最初に列挙されているのを確認する。

リスト 1. 入出力デバイスのリスト

1	Number of Devices: 3
2	0 : "Built-in Microph"
3	1 : "Built-in Input"
4	2 : "Built-in Output"

2. マイクからの入力音を得るには

マイクからの入力を SC で得るには、リスト 2 のように **SoundIn** という UGen を用いる。このプログラムでは単純にマイクからの音声をオーディオ出力装置に送っている。プログラムを実行すると、内蔵マイクに拾われた音がそのままヘッドフォンから聞こえてくる。

リスト 2. マイク入力

```

1 | {
2 |   a = SoundIn.ar(0);
3 |   [a, a];
4 | }.play;
```

リストでは、SoundIn の引数として「0」という数値が与えられているが、これは最初のオーディオ入力チャンネル、すなわちコンピュータのステレオ・マイクの L チャンネルを取得するという意味である。もし、「0」の代わりに [0, 1] のように Array を与えると左右両方のチャンネルを取得することができる。また、3 行目の [a, a] という Array はヘッドフォンの左右両チャンネルにマイクの L チャンネルからの音を送るためのものである。[a, a] の代わりに、もし「a」のみを書いた場合は、ヘッドフォンの左チャンネルのみにしか音が送られない。

SoundIn

コンピュータのマイクやサウンドカードからの音声入力を得る

`.ar(bus, mul, add)`

bus... 入力チャンネル番号。Array を与えることで、複数のチャンネルを同時に取得可能

3. リング・モジュレーション

まず初めに、単純なエフェクトの一例として、リング・モジュレーションをプログラムする。リング・モジュレーションは入力音の周波数成分の周囲にサイドバンドと呼ばれる周波数成分を作り出して入力音の音色を変化させるもので、リスト3のように、SoundInからの入力音に正弦波を掛けあわせてプログラムする事ができる。

リスト3. リング・モジュレーション

```
1 {
2   a = SoundIn.ar(0) * SinOsc.ar(MouseX.kr
3     (20,880));
4   [a, a];
5 }.play;
```

リストでは、MouseXを用いてマウスカーソルの画面上の位置で正弦波の周波数をコントロールできるようにしている。プログラムを実行し、マイクに向かって喋りながら、左右にマウスカーソルを動かすと、リング・モジュレーションにより加工された声に変化していくのが聞き取れる。

4. ディストーション

オーディオ信号を増幅し歪ませる事で、豊かな倍音を含んだ音を得る、ロック・ギターでお馴染みのディストーションは、SC3ではリスト4のようにdistortメソッドを用いてプログラムする事ができる。

リスト4. ディストーション

```
1 {
2   a = SoundIn.ar(0, 100).distort * 0.1;
3   [a, a];
4 }.play;
```

.distortを用いて確実に入力音を歪ませるには、ある程度の振幅のある入力信号が必要となる。このため、リストではSoundInのmul引数を100に設定して入力音を100倍に増幅し、.distortにより歪ませた後、「* 0.1」で聴取しやすいよう、レベルの調整をしている。.distortの他にも.softclipを使うとdistortと違った歪み方をさせることが可能である。図2は10倍に増幅した正弦波をそれぞれ.softclipとdistortで歪ませた時の波形の違いを図示したものである。

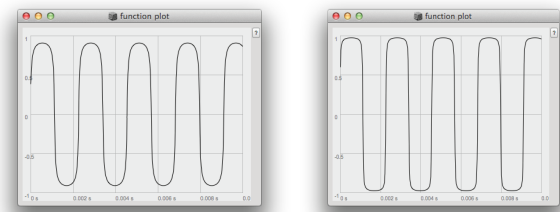


図2. .distort(左)と.softclip(右)により歪ませた正弦波

5. ピッチ・シフター

入力音のピッチ（音高）を自在に変化させるピッチ・シフターはPitchShiftというUGenを用いれば簡単にプログラム可能である。PitchShiftはごく短いバッファに入力音を繰り返し録音し、それをpitchRatioパラメータで指定された速度で再生することにより入力音の音程を変化させている。リスト5では、このパラメータを0.5に設定しているため、入力音が半分の速度での再生される、このため入力音を1オクターブ低くシフトした音が得られる。

リスト5. ピッチ・シフター

```
1 {
2   a = PitchShift.ar(SoundIn.ar(0),
3     pitchRatio:0.5);
4   [a, a];
5 }.play;
```

例えば、入力音より短三度高い音と長三度低い音を付加してピッチ・シフターを利用した三和音を作る事も可能である。このように音階上にピッチシフトを行う場合は、リスト6のように以前に学習した「.midicps」を利用する。

リスト6. ピッチ・シフターによる和音

```
1 {
2   a = PitchShift.ar(SoundIn.ar(0),
3     pitchRatio: 3.midicps / 0.midicps);
4   a = PitchShift.ar(SoundIn.ar(0),
5     pitchRatio: -4.midicps / 0.midicps)
6     + a;
7   [a, a];
8 }.play;
```

.midicpsは、MIDIノートナンバーを周波数に変換するメソッドであり。「0.midicps」を実行すると、MIDIノートナンバー「0」の周波数が得られる。また、MIDIノートナンバー「0」より短三度高い、つまり半音階で3高いMIDIノートナンバー「3」と、長三度低い、つまり半音階で4低い、ノートナンバー「-4」の周波数をmidicpsでそれぞれもとめ、それをノートナンバー「0」の周波数で除算することで、それぞれノートナン

パー「4」と「0」、「-3」と「0」間の周波数比、つまりどれだけ速く（あるいは遅く）入力音を再生すれば、長三度高い、または短三度低い音程が得られるか、を数値として得る事ができる。これを *pitchRatio* の値として PitchShift に与える事で、入力音より短三度高い音と長三度低い音を作り、ピッチ・シフターを用いた長三和音を実現する事ができる。

PitchShift

ピッチシフトを行う UGen。音程は *pitchRatio* で指定する。dispersion のパラメータを用いることで、音程のランダム化なども可能。

.ar(in, windowSize, pitchRatio, pitchDispersion, timeDispersion, mul, add,)

in... 入力信号

windowSize... ウィンドウ・サイズ

pitchRatio... ピッチ・レシオ。再生速度

pitchDispersion... ピッチ分散度

timeDispersion... 時間分散度

6. デレイ

本項では入力音を一定時間遅延させるデレイとその応用を数種紹介する。

6.1. シンプルなデレイ

遅延効果を SC で用いるには DelayN という UGen を使用する。この UGen の第 2、第 3 引数はそれぞれ、最大デレイ・タイム最大 (*maxdelaytime*) とデレイ・タイム (*delaytime*) で、最大デレイ・タイムの値を元に SC はデレイ用のバッファを用意し、それを利用してデレイ・タイムで指定された時間だけ入力音を遅延させて出力する。最大デレイ・タイムを超えたデレイ・タイムを設定した場合、デレイ・タイムは最大デレイ・タイムにクリップされる (リスト 7)。

リスト 7. デレイ

```
1 {
2   a = DelayN.ar(SoundIn.ar(0), 0.5, 0.5)
3   [a, a];
4 }.play;
```

DelayN

入力音を *delaytime* で指定した時間だけ遅延させる。

.ar(in, maxdelaytime, delaytime, mul, add)

in... 入力信号。

maxdelaytime... 最大デレイ・タイム。デレイ用のバッファの確保に使用する。

delaytime... デレイ・タイム。最大デレイ・タイムより大きい値を指定した場合、最大デレイ・タイムにクリップされる。

6.2. フィードバック・デレイ

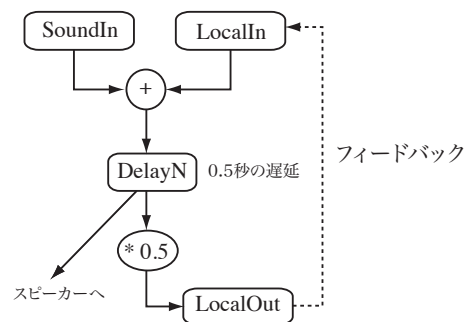


図 3. フィードバック・デレイの成り立ち

遅延させた信号を振幅を弱め、もう一度 Delay.ar に入力することによって、入力音を減衰させながら何度も繰り返す、やまびこのようなエフェクト、フィードバック・デレイが実現できる (図 3)。このようなフィードバック・デレイを実現するには、リスト 8 のように LocalIn と LocalOut という UGen を用いる。リストでは、4 行目で DelayN による 0.5 秒の遅延処理と「* 0.5」による振幅の減衰が施された信号を LocalOut に送り、それが 2 行目の LocalIn によって取り出され、入力音と加算されている。

リスト 8. フィードバック・デレイ

```
1 {
2   a = SoundIn.ar(0) + LocalIn.ar(1);
3   d = DelayN.ar(a, 0.5, 0.5);
4   LocalOut.ar(d*0.5);
5   [d, d]
6 }.play
```

LocalIn

LocalOut により内部バスに送られた信号を取り出す

.ar(numChannels, default)

numChannels... 出力するチャンネル数

default... LocalOut からの信号が到達する前に出力されるデフォルト値

LocalOut

引数で与えられた信号を内部バスに送る。

.ar(channelArray)

channelArray... 入力信号。複数のチャンネルを内部バスに送る場合は Array を使う

```
3 | d = DelayN.ar(i, 0.25, 0.25) * 0.5;
4 | a = LocalIn.ar(2);
5 | l = a[0] + i;
6 | r = a[1] + d;
7 | l = DelayN.ar(l, 0.5, 0.5) * 0.5;
8 | r = DelayN.ar(r, 0.5, 0.5) * 0.5;
9 | LocalOut.ar([l,r]);
10 | [l,r+d]
11 | }.play
```

6.3. ピンポン・ディレイ

フィードバック・ディレイにアレンジを加え、各ディレイ音が左右のスピーカーから交互に聞こえる、ピンポン・ディレイを作る事もできる。

入力音

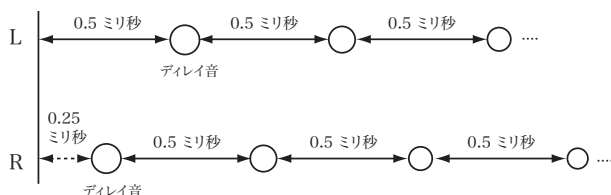


図 4. ピンポンディレイ

左右のスピーカーに交互にディレイを送るには、左右のスピーカー用にそれぞれ独立したフィードバック・ディレイを用意し、片方のスピーカーのフィードバック・ディレイに音を入力するタイミングをもう片方に入力するタイミングより、ディレイ・タイムの半分 (この例では 0.25 秒) ずらすことにより実現できる (図 4)。

リスト 9 は 0.25 秒ごとに左右のスピーカーから交互にディレイ音が聞こえるピンポン・ディレイのプログラム例である。3 行目では、入力音に 0.25 秒の遅延と「* 0.5」により振幅を減衰を施し、それを変数 d に代入している。また、4 行目で LocalIn を用いて 9 行目の LocalOut からの 2 チャンネルの信号を Array として取得し、変数 a に格納。5、6 行目でその L チャンネル (a[0]) を入力音 i と、R チャンネル (a[1]) を、0.25 秒の遅延を伴った d とミックスしている。それを、7、8 行目で、それぞれ個別の DelayN に入力することによって、両チャンネルを 0.5 秒遅延させ、続く「* .5」で減衰させ、9 行目で LocalOut に送り、フィードバックを生じさせている。

10 行目は [l,r+d] となっているため、左チャンネルからは変数 l の音、つまり 0.5、1.0、1.5、2.0.. 秒後のディレイが聞こえ、右のチャンネルからは、変数 d と r の音のミックスした音、つまり 0.25、0.75、1.25、1.75... 秒後のディレイが聞こえてくる。

リスト 9. ピンポン・ディレイ

```
1 | {
2 | i = SoundIn.ar(0);
```

6.4. マルチタップ・ディレイ

リスト 10 のように DelayN と Array と組み合わせて、リズムカルなディレイ (=マルチタップ・ディレイ) を作る事も可能である。

リスト 10. マルチタップ・ディレイ

```
1 | {
2 | t = 0.1;
3 | a = [1, 2, 4, 6, 7] * t;
4 | d = DelayN.ar(SoundIn.ar(0), a, a);
5 | m = Mix.ar(d);
6 | [m, m];
7 | }.play
```

このリストで変数 a はマルチタップ・ディレイのリズムパターンが定義された Array が格納されている。Array は [1, 2, 4, 6, 7] というリズムパターンに 0.1 が掛けられているので、その内容は [0.1, 0.2, 0.4, 0.6, 0.7] となる、この Array を 4、5 行目の DelayN の最大ディレイ・タイムとディレイ・タイムにそれぞれ適応している。このように引数に Array が与えられた場合、SC は自動的に DelayN を複製する。

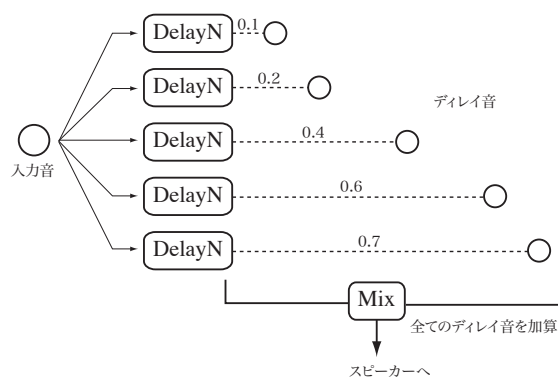


図 5. マルチタップ・ディレイ

ここでは 5 つの要素からなる Array が与えられたため、DelayN が 5 つ作られ、それぞれに異なった最大ディレイ・タイム及びディレイ・タイムが指定される。そして、入力音は Array で指定されたように 0.1 秒、0.2 秒、0.4 秒、0.6 秒、0.7 秒遅延して出力される、これらは DelayN からオーディオ信号の Array として出力され、5 行目で Mix という UGen を使って、全ての要素をミックスしている (図 5)。

また、このリストの変数 *t* の値を変えると、マルチタップ・ディレイのリズム・パターンを変えずにテンポだけを変える事ができる。

Mix

array 内の音声信号を全て加算する

.ar(array)

array... 音声信号の Array、全てのチャンネルが加算される

ディレイは非常に多様な可能性を持っており、本項で紹介したような純粋な遅延効果の他にも、フランジャーやコーラスのようなエフェクトをこれに応用して作事ができる。これらについてはまた回を改めて紹介する。

7. リバース

残響効果を入力音に加えるには、以下のように FreeVerb という UGen を利用するのが最も簡単である。

リスト 11. リバース

```
1 {
2   a = FreeVerb.ar(SoundIn.ar(0), 1.0);
3   [a, a];
4 }.play;
```

FreeVerb

入力音にリバース（残響効果）を付加する UGen

.ar(in, mix, room, damp, mul, add)

in... 入力信号

mix... 入力音と加工音のバランス、1 で加工音のみ、0 で入力音のみ

room... ルームサイズ、1.0 が最大、0.0 が最小

damp... 高周波数のダンブ。0 から 1 の範囲で指定

8. フィルタ

SC には様々なフィルタが予め用意されている。リスト 12 では、入力音にハイパス・フィルタを施し、3000Hz 以下の低周波成分を減衰させている。

リスト 12. ハイパス・フィルタ

```
1 {
2   a = HPF.ar(SoundIn.ar(0), 3000);
3   [a, a];
4 }.play;
```

SC には HPF(High Pass Filter) の他にも LPF(Low Pass Filter)、BPF(Band Pass Filter)、Moog、TwoPole など様々なフィルタが用意されている。「Filter」をヘルプで調べると、全てのフィルタのリストを見ることができる。

HPF

2 次バターワース・ハイパスフィルタ

.ar(in, freq, mul, add)

in... 入力信号

freq... カットオフ周波数

9. ワウワウ

リスト 13 では、バンド・パスフィルタ中央周波数のパラメータを LFO でコントロールすることにより、ファンク・ギターなどでお馴染みの「ワウワウ」のエフェクトを作成している。

リスト 13. ワウワウ

```
1 {
2   l = SinOsc.ar(5, 0, 1000, 1500);
3   a = BPF.ar(SoundIn.ar(0), l, 0.5);
4   [a, a];
5 }.play;
```

BPF の三番目の引数は *rq*(reciprocal of *Q*) 値であり、この値が少なければ少ないほどフィルタを通過する帯域幅が狭くなり、強くワウワウがかかる。

BPF

2 次バターワース・バンドパスフィルタ

.ar(in, freq, rq, mul, add)

in... 入力信号

freq... 中央周波数

rq... reciprocal of *Q* 値

10. エフェクタを SYNTHDEF としての定義する

SC3 はこれまでに制作してきたようなエフェクタを複数組み合わせ、SynthDef として定義することも可能である。リスト 14 ではピッチ・シフター、ディレイ、リバースを連結して、より複雑なエフェクトをプログラムし、それを「myEffect」という SynthDef として定義している。

リスト 14. エフェクトの SynthDef

```
1 SynthDef("myEffect", {
2   i = SoundIn.ar(0);
3   i = PitchShift.ar(i, pitchRatio:0.5) + i;
4   i = DelayN.ar(i, 0.5, 0.5) + i;
5   i = FreeVerb.ar(i, 1.0) + i;
6   Out.ar(0, [i, i]);
7 }).load;
```

リストでは 3 から 5 行目にかけて、「+i」が行末に書かれているが、これはエフェクトにより加工された音

と、エフェクトへの入力音を足し、両方の音が聞こえるようにするためである。この SynthDef も前回同様に Synth を用いて使用する (リスト 15)。

リスト 15. エフェクト SynthDef の使用

```
1 | Synth ("myEffect ")
```

11. まとめ

今回は SC による様々なエフェクトのプログラミングを学習した。エフェクタは、さらに SC のバス (Bus) と言われる機能を利用して、エフェクタの Synth を複数繋ぐ、音を生成する Synth とエフェクタを定義した Synth を組み合わせて、自作シンセ音に各種エフェクトを適用するという等、様々な応用が可能である。このバス機能については、また回を改めて紹介する。

12. 参考文献

- [1] *SuperCollider*, <http://supercollider.sourceforge.net>(アクセス日 2014 年 6 月 10 日)
- [2] Wilson, S., Cottle, D. and Collins, N. (eds), *The SuperCollider Book* The MIT Press, 2011

13. 著者プロフィール

13.1. 美山 千香士 (Chikashi Miyama)

作曲家、電子楽器作家、映像作家、パフォーマー。国立音楽大学音楽デザイン学科より学士・修士を、スイス・バーゼル音楽アカデミーよりナッハ・ディプロムを、アメリカ・ニューヨーク州立バッファロー大学から博士号を取得。作曲・コンピュータ音楽を葉孝之、エリック・オニヤ、コート・リッピ氏らに師事。Prix Destellos 特別賞、ASCAP/SEAMUS 委嘱コンクール 2 位、ニューヨーク州立大学学府総長賞、国際コンピュータ音楽協会賞を受賞。2004 年より作品と論文が国際コンピュータ音楽会議に 13 回入選、現在までに世界 19 カ国で作品発表を行っている。2011 年、DAAD(ドイツ学術交流会) から研究奨学金を授与され、ドイツ・カールスルーエの ZKM で客員芸術家として創作活動に従事。近著に「Pure Data-チュートリアル&リファレンス」(Works Corporation 社)がある。現在ドイツ・ケルン音楽大学講師。スイス・チューリッヒ芸術大学コンピュータ音楽・音響研究所 (ICST) 研究員。バーゼル造形大学のプロジェクト「Experimental Data Aesthetics」プログラマ。2013 年に松村誠一郎氏と日本語の Pure Data ポータル Pure Data Japan(<http://puredatajapan.info>) を創設。公式ウェブサイト: <http://chikashi.net>