連載

SUPER COLLIDER チュートリアル (3)
SUPER COLLIDER TUTORIALS (3)

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：MIDI キーボードで SYNTH を演奏する

前回は、Env や EnvGen を使ったエンベロープの作成、SynthDef による Synth の定義、Routine を用いたメロディーの演奏法等を学習した。今回は、Synth を MIDI キーボードを用いて演奏できるプログラムを作成し、それを通して SC における MIDI メッセージの取り扱いを学習する。

2. MIDI 機器の接続

2.1. 接続の確認

SC を立ち上げ、手持ちの MIDI キーボードを繋ぎ、リスト 1 のコマンドを実行する。

リスト 1. MIDIClient の初期化

```
1 MIDIClient.init;
```

MIDIClient は OS の MIDI レイヤーにアクセスするクラスであり、MIDI を SC で扱う際には必ずまず初めにこのクラスに.init メッセージを送る必要がある。

MIDIClient.init を実行すると、MIDI の接続が初期化され、OS から取得した現在の MIDI 機器の接続状況がリスト 2 のように表示される。「MIDI Sources」以下にはコンピュータへの MIDI メッセージの送信元、「MIDI Destinations」以下にはコンピュータからの MIDI メッセージの送信先が MIDIEndPoint として列挙されている。

筆者は AKAI の MIDI キーボード LPK25 を接続しているので、このキーボードの名前が表示されている。また、Mac を使っているので、IAC Driver[1] もリストされている[2]。

リスト 2. 現在接続されている MIDI 機器

```
1 MIDI Sources:
2   MIDIEndPoint("IAC Driver", "Bus 1")
3   MIDIEndPoint("LPK25", "LPK25")
4 MIDI Destinations:
5   MIDIEndPoint("IAC Driver", "Bus 1")
6   MIDIEndPoint("LPK25", "LPK25")
```

2.2. 入力をテストする

コンピュータに繋がっている全ての MIDI 機器からのメッセージを SC で受け取るためには、MIDIIn というクラスに「connectAll」メッセージを送る。その後、MIDIFunc というクラスを用いて様々な MIDI メッセージを SC が受け取った際に、どのようにそれらのメッセージを処理するかを指定する。リスト 3 では MIDIFunc に noteOn メッセージを送り、MIDI キーボードが打鍵された時に SC に送られてくる MIDI ノート・オン・メッセージに対してどのような処理を行うかを定義している。

リスト 3. MIDI 入力の確認

```
1 MIDIClient.init;
2 MIDIIn.connectAll;
3 m = MIDIFunc.noteOn({
4   arg vel,note,chan;
```

[1] Inter Application Communication Driver。Mac におけるアプリケーション間で MIDI メッセージを送受信するためのドライバ。これを利用して DAW と SC を連携させることも可能。

[2] この接続では 1 つのデバイスに対して、1 つのポートであるが、1 つのでデバイスで複数のポートが提供される場合もある。

```
5|   [note,vel,chan].postln
6| });
```

noteOn メッセージの引数として指定された関数内では 3 つの引数 (arg)「vel」、「note」、「chan」が宣言されており、MIDI キーボードを打鍵した際に vel には打鍵の強さ (ベロシティ)、note には音高を表す MIDI ノートナンバー、そして chan には MIDI チャンネルが代入される。

このプログラムでは、この 3 つの要素を 1 つの Array に格納し、それに対して.postln メッセージを送っているので、MIDI キーボードを打鍵すると、[100, 60, 0] のように受信した MIDI ノート・オン・メッセージのベロシティ (100)・ノートナンバー (60)・MIDI チャンネル (0) がポスト・ウインドウ表示される。また、ここで定義した MIDIFunc は変数 m に格納されている[3]。

リスト 4. 関数の解放

```
1| m.free;
```

プログラムのテストを行った後、リスト 4 のように必ず MIDIFunc を格納した変数、m に.free を送り、定義した MIDIFunc の解放を行う必要がある。するとこれ以降、MIDI キーボードを打鍵してもポストウインドウに受け取った MIDI メッセージが表示されなくなる。

.free の実行を怠り、さらにもう一度 MIDIFunc.noteOn メッセージを実行すると、MIDI ノート・オン・メッセージに反応する関数を多重に定義してしまうこととなる。もし、このような意図しない多重定義を行ってしまった場合は、SC の「Language」メニューから「Reboot Interpreter」を選び、一度 SC インタプリタをリセットする必要がある。

3. SYNTH を演奏する

3.1. 固定長エンベロープの Synth を演奏する

リスト 5. 固定長エンベロープの Synth の例

```
1| SynthDef.new("jssa3",{
2|   arg freq = 440,amp = 0.1;
3|   a=Saw.ar(freq,amp);
4|   e=Env.new([0,1,1,0],[0.05,0.1,0.1]);
5|   Out.ar(0,a*EnvGen.ar(e,doneAction:2));
6| }).load;
```

リスト 5 は前回作成した”jssa3”という SynthDef である。MIDI キーボードでこの Synth を演奏するにはリスト 6 のように MIDIFunc.noteOn で登録する関数の中に Synth を組み込む。

リスト 6. Synth と MIDI メッセージを関連付ける

```
1| MIDIClient.init;
2| MIDIIn.connectAll;
3| m = MIDIFunc.noteOn({
4|   arg vel,note,chan;
5|   Synth("jssa3",["freq",note.midicps,"amp",
     vel/127]);
6| });
```

これにより、MIDI キーボードの打鍵により、MIDIFunc で登録した関数が実行され、SynthDef で定義した Synth の音が鳴るようになる。このプログラムもテストが終わったら、リスト 4 を実行し、MIDIFunc を解放する必要がある。

3.2. MIDI キーボードが手元にない場合

もし、MIDI キーボードが手元にない場合は、リスト 7 の要領で MIDIIn クラスに.doNoteOnAction を送り、擬似的に生成した MIDI ノート・オン・メッセージを SC に送り、MIDIFunc で登録した関数の反応をテストする事ができる。

リスト 7. 擬似的に MIDI メッセージを送る

```
1| MIDIIn.doNoteOnAction(0,0,60,64);
```

この時の引数は左から、入力ソース ID、MIDI チャンネル、MIDI ノートナンバー、ベロシティとなる。故に、リスト 7 を実行すると、MIDI チャンネル 0、MIDI ノートナンバー 60(C4)、ベロシティ 64 の擬似 MIDI ノート・オン・メッセージを SC に送ることができる[4]。

4. 音の長さを自在に変える

前回のチュートリアルでは固定長のエンベロープを用いており、太鼓やチェンバロのように、音の長さを自在に変えるのが不可能であったが、この項では**可変長エンベロープ**を用いて、ヴァイオリンやオルガンのように、自在に音の長さを変えられるエンベロープを設計する。可変長のエンベロープの設計を行うには Env クラスに対して.new ではなく、.adsr や.asr などのメッセージを送る。

4.1. 可変長エンベロープの使用法

リスト 8. 可変長エンベロープの定義

```
1| SynthDef("jssa4", {
2|   arg freq = 440,amp = 0.5,gate = 1;
3|   e = Env.adsr(0.7,0.5,0.7,1.0);
4|   a = Saw.ar(freq,amp)*EnvGen.ar(e,gate,
     doneAction:2);
5|   Out.ar(0,a);
6| }).load;
```

[3] MIDI チャンネル・ナンバーは規格上は 1 から 16 とされている [2] が、SC では MIDI チャンネルは 0 から始まる

[4] MIDIIn.connectAll を使っているので、入力ソース ID にはどのような値を入れても、反応が得られる

リスト 8 では、Env に.adsr メッセージを送っている。この時メッセージの 4 つの引数はそれぞれ A=アタック・タイム (0.7 秒)、D=ディケイ・タイム (0.5 秒)、S=サステイン・レベル (アタックの 0.7 の振幅)、R=リリース・タイム (1.0 秒) となる。リストでは EnvGen に Env.adsr で定義したエンベロープを第 1 引数として渡し、さらに変数 gate を第 2 変数として渡している。この gate の値が 0 より大きくなった時に EnvGen はエンベロープのアタック部からサステイン部までを出力し、0 になった時にリリース部に進める。このように、gate に 0 を入れるタイミングによって音の長さを自在にコントロールすることができるエンベロープが可変長エンベロープである (図 1)。

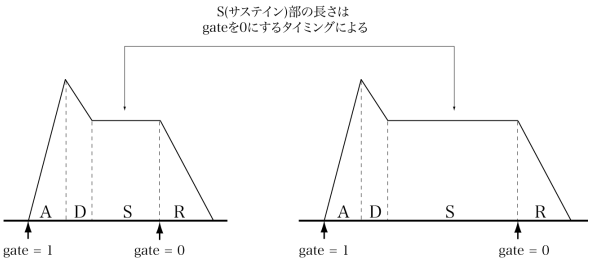

図 1. 可変長エンベロープの例

この SynthDef の定義では gate の初期値が 2 行目で「1」に定められているため、この SynthDef を使って発音する際は、引数として gate に 1 を指定しなくとも、自動的に発音が始まる。また「doneAction:2」が指定されているため、エンベロープ終了と同時に自動的にこの Synth に使用されたリソースは解放される。

Env

.adsr(*attackTime*, *decayTime*, *sustainLevel*, *releaseTime*, *peakLevel*, *curve*, *bias*)

attackTime· · · *gate* を 1 にしてから *peakLevel* 到達までの時間。

decayTime· · · *peakLevel* から *sustainLevel* への減衰時間。

sustainLevel · · · サステイン部のレベル。1.0 ならば *peakLevel* と同じレベルとなる。

releaseTime · · · *gate* に 0 が送られてから、完全に消音するまでの減衰に要する時間。

peakLevel · · · エンベロープのピークのレベル。

(その他の引数についてはヘルプファイルを参照のこと)

SynthDef での定義が終わったら、リスト 9 を実行し、可変長エンベロープをテストしてみる。

リスト 9. 可変長エンベロープの Synth の実行

```
1 x = Synth("jssa4", ["freq", 60.midicps]);
```

プログラムを実行すると、C4(ピアノの真ん中のド) が鳴り始めるはずである。これを止めるには、リスト 10 のように、.set(”gate”,0) メッセージを Synth が格納されている変数 x に対して送る。すると、エンベロープがリリース部に移行し、1 秒後に音が完全に消え、doneAction の効果により Synth に使われていたリソースが自動的に解放される。

リスト 10. 可変長エンベロープのリリース

```
1 x.set("gate", 0);
```

4.2. MIDI キーボードと組み合わせる

このように可変長エンベロープを用いる場合、基本的に発音した全ての音に.set(”gate”, 0) を送り、音を止める必要がある。これを MIDI キーボードと組み合わせる場合はリスト 11 のように、MIDIFunc.noteOff を用いる。MIDIFunc.noteOff は、MIDIFunc.noteOn が MIDI キーボードが打鍵された時の振る舞いを指定するものであるのに対し、鍵盤から指が離れた時、離鍵した時の振るまいを定義する。

リスト 11. 可変長エンベロープと MIDI キーボード

```
1  MIDIClient.init;
2  MIDIIn.connectAll;
3  m = MIDIFunc.noteOn({
4    arg vel,note,chan;
5    x = Synth("jssa4",["freq",note.midicps,"
       amp",vel/127]);
6  });
7  n = MIDIFunc.noteOff({
8    arg vel,note,chan;
9    x.set("gate",0);
10 });
```

リスト 11 では可変長エンベロープの Synth が使われているので、プログラムを実行して、任意の 1 つの鍵盤を打鍵すると、MIDIFunc.noteOn で指定した関数が実行され、音がなり始める。そして、その鍵盤の離鍵のタイミングに合わせて、MIDIFunc.noteOff で指定した関数が実行され、x.set(”gate”,0) を介してエンベロープがリリースされ、音の生成が停止される。

プログラムでは MIDIFunc.noteOn と MIDIFunc.noteOff はそれぞれ、変数 m と f に格納されている。プログラムのテストが終わったら、m と n 両方の変数に.free を送り、MIDIFunc を解放する (リスト 12)。

リスト 12. MIDIFunc の解放

```
1 m.free;
2 n.free;
```

しかし、このプログラムでは、1 つの鍵盤を打鍵し、その鍵盤を離鍵する前に、他の鍵盤を打鍵し、重音や和音の演奏を試みた場合、両方の鍵盤を離鍵しても音が鳴り続けてしまう。図 2 は、このプログラムの実行時に F4 のキーを押した後に、A4 を押し重音を演奏した際

に、SC 内でどのような処理が行われるかを示したものである。F4 の Synth が格納された変数 x は A4 の打鍵と同時に A4 を発音している Synth に上書きされる。このため、その後、F4 のキーを離鍵した時に F4 の発音が停止されず、A4 の音が停止されてしまい、F4 が鳴りっぱなしとなる。

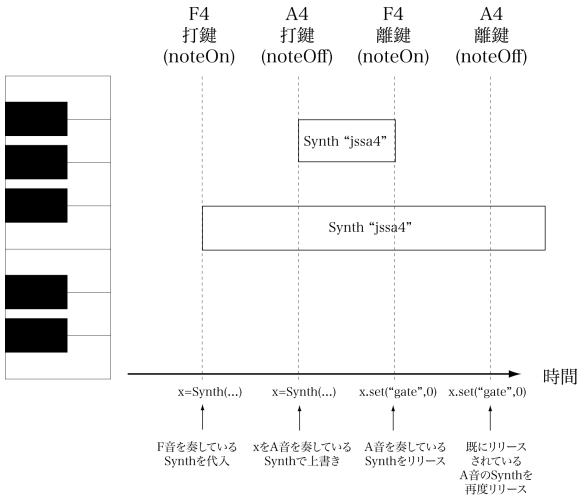

図 2. 重音・和音の問題

5. ポリフォニック・シンセサイザの実現

前項のような問題を回避し、重音・和音の演奏ができる**ポリフォニック・シンセサイザ**を実現するには、リスト 11 の x 変数のように、一時的に Synth を代入しておくための変数を鍵盤の数だけ用意すればよい。このように多くの変数が必要となる場合は、前回勉強した Array を使うと効率的である。リスト 13 では、Array に.newClear メッセージを送り、中身が「空」(=nil) の 127 個の要素からなら配列「a」を定義している。これであらゆる鍵盤が打鍵された時に、その鍵盤の MIDI ノートナンバーに匹敵する配列のインデックスに Synth を一時的に格納する事ができ、発音とリリースの処理を個別に各音に対して行う事ができるようになる (図 3)。

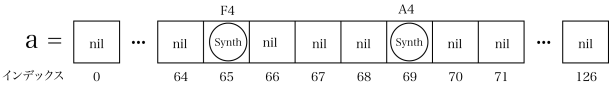

図 3. 配列と Synth

リスト 13. ポリフォニック・シンセサイザ

```
1 MIDIClient.init;
2 MIDIIn.connectAll;
3 a = Array.newClear(127);
4 m = MIDIFunc.noteOn({
5   arg vel,note,chan;
6   a[note] = Synth("jssa4",["freq",note.
      midicps, "amp", vel/127]);
7 });
8 n = MIDIFunc.noteOff({
9   arg vel,note,chan;
10  a[note].free;
11 });
```

このプログラムもテストが終わった後にリスト 12 を実行し、MIDIFunc を解放する必要がある。

6. まとめ

今回は SC と MIDI の連携、ポリフォニック・シンセサイザのプログラムを学習した。SC ではガーベッジコレクションや、エンベロープ関連のクラス、MIDIFunc による MIDI メッセージの取り回しの良さなどが相俟って比較的簡単にリソース消費の少ない実用的なシンセサイザをプログラムする事ができる。次回はこのシンセサイザに更に様々な機能を追加し、より表現力の高いものにしてゆく。

7. 参考文献

[1] *SuperCollider*, `http://supercollider.sourceforge.net`(アクセス日 2014 年 1 月 8 日)

[2] *MIDI Manufactures Association*, `http://www.midi.org`(アクセス日 2014 年 1 月 8 日)

8. 著者プロフィール

美山 千香士 (Chikashi Miyama)

作曲家、電子楽器創作家、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。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 12 回入選、現在までに世界 18 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、チューリッヒ芸術大学非常勤講師。ケルン・メディア芸術大学フェロー。チューリッヒ ICST の SpatDIF プロジェクト及び MGM プロジェクトにプログラマとして参加している。