Vol.2015-MUS-106 No.4
Vol.2015-EC-35 No.4
2015/3/2

タッチインタラクションを想定したリアルタイム・モーダルチェンジ・インタラクションシステムの構築

黒田元気[†1]　伊藤彰教[†2]　渡邉賢悟[†3]　伊藤謙一郎[†1]　近藤邦雄[†1]

スマートフォンなどのタッチデバイスが普及したことで、アート、エンタテイメント双方の分野において、新たなインタラクションに応じて音楽の様相を動的に変化させることは珍しくなくなった。しかし、技術的にはサウンドファイルのエフェクト処理や動的なミキシング、固定された調のループなどが多く、モーダルチェンジなど音列生成を柔軟に操作するなどの音楽的な操作が容易に行えるような環境および作品は普及していない。このため、インタラクションと音楽の関係についても楽理面からの考察が行いやすい状況とは言い難い。このため本研究では、iOS 上で Supercollider を動作させ、iOS ネイティブのインタラクション API から動的にスケールを変更・加工するシステムを実装し、エンタテイメント分野の音楽で頻出するモードスケール・コードおよびその変更と様々なタッチ系の入力操作を容易に組み合わせる環境を構築した。これにより楽理面との関係に関する検討が容易になった。

1. はじめに

エンタテイメント・コンピューティングの中でも主要な分野のひとつであるゲーム開発においては，「adaptive music」という状況適応型音楽の可能性が試されてきた[1]．手法としてはリアルタイムミキシングを用いて音楽の要素を動的に組み合わせるファイルベース処理のものから，音楽を音符単位で動的に変更する手法など様々なものがあり，これらのプロトタイピングとして Pure Data を用いる技術的提案もなされている[2]．特に音符・スケール・リズムなど音楽の構成要素単位で可変させる技術を後押ししたのは Microsoft 社の Direct Music API と関連するミドルウエア群だが，本論文執筆時点では非推奨[3]となっており，adaptive music に関する技術的・音楽的可能性を手軽に試みる環境が限られてしまってきている．一方，スマートフォンの普及に伴い，エンタテイメント・プラットフォームとしての存在感が高まっている．ゲームコンソールに比べ比較的オープンな開発環境であり[4,5]，音楽情報処理分野でのソフトウエア群の porting も進められている[6]ため，アルゴリズミック・コンポジション研究などのこれまでの音楽情報処理研究の知見をエンタテイメント分野へ応用する新たな技術的可能性が高まっている．

こうした技術的環境を鑑み本研究では，リアルタイムに音楽の要素を音符単位で可変可能な SuperCollider に着目し，これをサウンドエンジンとしてスマートフォンのネイティブ UI API から制御する手法について，実装を通じて検討を進めた．主要な課題としては，SuperCollider の iOS 上での挙動調整，ネイティブ API から iOS 機内部の SuperCollider Server への SuperCollider コードフラグメントの送信方法，これらを技術的基盤として音楽的なモーダルチェンジがリアルタイムに実行可能かの３点に絞り研究を進めた．

2. iOS 上での SuperCollider の利用

2.1 これまでの動作事例

iOS 上で動作する SuperCollider の開発は，これまでにも SourceForge 上にて行われてきた．筆者らはこれらを利用し，ビルド可能な環境を確認した．

表 1　既存の iOS 版 SuperCollider のビルド環境

OS	OS X 10.9.4
統合開発環境	Xcode 4.6.2
コンパイラ	LLVM GCC 4.2
音響生成	SuperCollider 3.2
ビルド先端末	iPod touch 第 4 世代
端末 OS	iOS 6.1.5

公開されているパッケージを用いて　iOS 上にビルドした SuperCollider は，Post Window，ドキュメントセレクタ画面，エディタ画面，サーバ情報画面の 4 つの画面から構成される（図 1）．

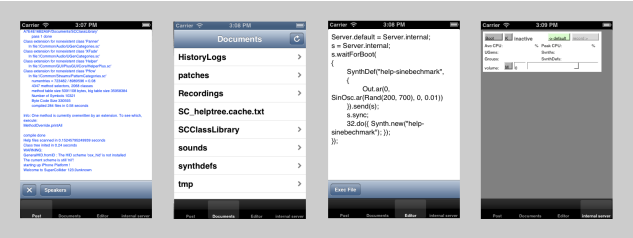

図 1　既存の iOS 版 SuperCollider の GUI 一覧

エディタ画面を表示させ，任意の箇所にタッチするとキーボードが表示され，SuperCollider プログラムを入力して実行することができる（図 2）．

†1 東京工科大学　メディア学部
Dep. of Media Science, Tokyo Univercity of Technology
†2 東京工科大学片柳研究所クリエイティブ・ラボ
Creative Lab., KARL, Tokyo Univercity of Technology
†3 渡辺電気株式会社
Watanabe-DENKI Inc.

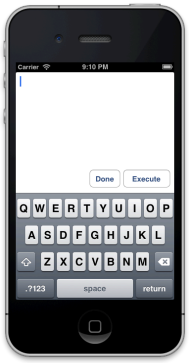

図 2　既存の iOS 版 SuperCollider のエディタとキーボード

ビルドの方法に関する情報は少なく，ビルドを試みるユーザが確実にその方法を把握できるとは言いがたい状況にある．そのうえ，これまで iOS 用 SuperCollider として公開されていたパッケージは，iOS6.1 上に，LLVM GCC 4.2 でコンパイルを行うことが想定されており，最新の環境に対応するものではない．

2.2 iOS7 以降に対応した SuperCollider の開発

本研究者の 1 名である渡邉は，公開されている iOS 用 SuperCollider のパッケージを，iOS7 以降でも動作するように編集し，GitHub にて「wdkk/supercollider_iOS」として公開した[7]．このパッケージは下記の環境（表 1）にて，iOS デバイス上に SuperCollider のビルドが可能である．

表 2　wdkk/supercollider_iOS のビルド環境

OS	OS X 10.9.5
CPU	1.4GHz Intel Core i5
統合開発環境	Xcode 6.1
コンパイラ	Apple LLVM 6.0
音響生成	SuperCollider 3.2
ビルド先端末	iPod touch 第 5 世代
端末 OS	iOS 8.1.3

2.3 パッケージの構造

SuperCollider をビルドするためのメインプロジェクトは，プロジェクト内の platform/iphone/iPhone_Language.xcodeproj であり，音響生成部の SuperCollider サーバを構成するサブプロジェクトである iPhone_Synth.xcodeproj をもち，さらに iPhone_Synth.xcodeproj は，オーディオファイルを扱うためのライブラリを構成するサブプロジェクトである libsndfile.xcode をもつ．この方法でビルドしたアプリケーションは，従来の iOS6 以前に対応したパッケージと同等の画面および機能を備えている．

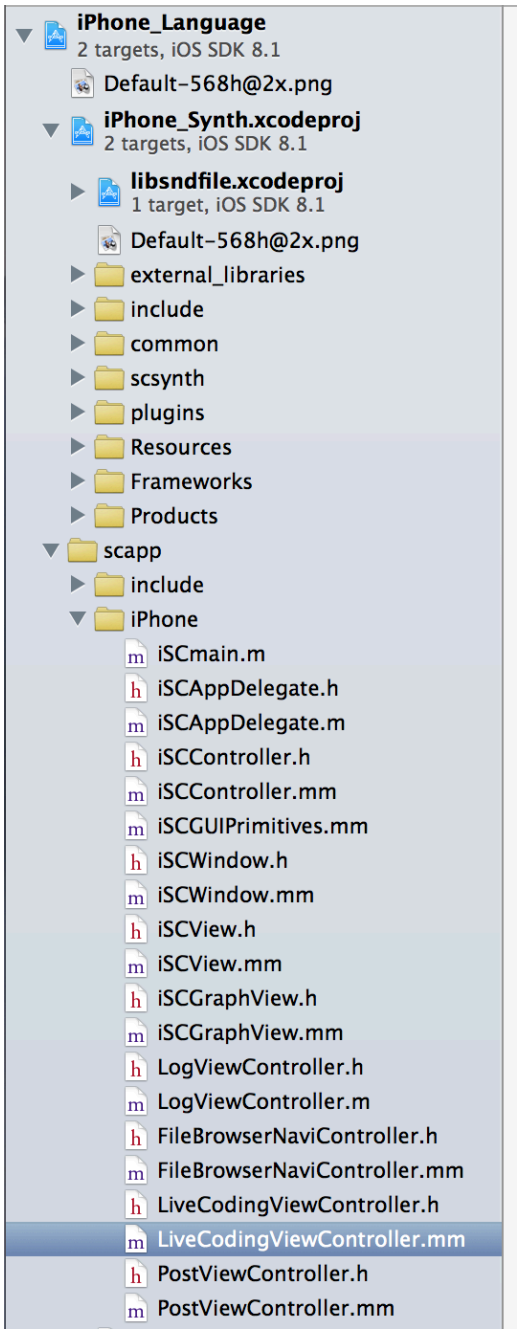

図 3　wdkk/ wdkk/supercollider_iOS のプロジェクトツリー

3. SC Server とネイティブ UI API の連携

3.1 システム概要

本研究の目的は，デバイスの操作によって音楽を変化させ，インタラクションと音楽の関係を楽理的に考察することにある．つまり，本研究において SuperCollider を利用するには，エディタを介さず，インタラクションに応じて即時に SuperCollider のコード断片を SuperCollider サーバに送出できる必要がある．そこで，コード送出機能をもったユーザインターフェイスを新たにアプリケーション上に制作した．

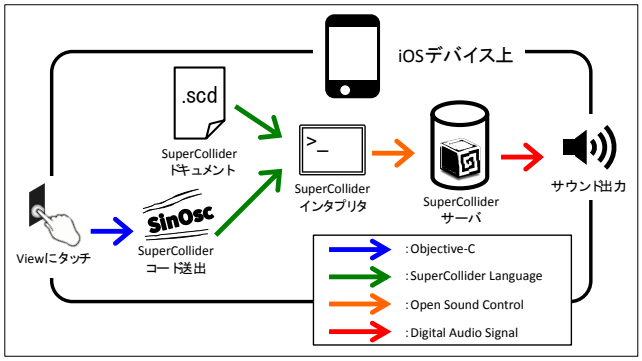

図 4　システムデザイン

3.2 View オブジェクトからのコード送信

メインプロジェクトの scapp/iPhone ディレクトリ下には，LiveCodingViewController.h および LiveCodingViewController.mm が格納されており，この両ファイルが SuperCollider においてコードを入力するエディタを構成している．iOS ネイティブのオブジェクト描画およびインタラクション API から，で SuperCollider の音響生成機構が利用できることを実証するため，エディタ上に iOS における View オブジェクトを設置し，これに対応するインタラクションオブジェクトとして「TouchView」クラスを追加した．TouchView.h および TouchView.mm に記述された TouchView クラスは，タッチを検出し，他メソッドに記述した処理を行わせるものである．このクラスを LiveCodingViewController に結びつけ，設置したビューのタッチにより SuperCollider コードをサーバに送信できるようにした．

(1) View 変数の追加

まず LiveCodingViewController.h に以下の記述を追加した．

```
@property TouchView *tv;
```

(2) タッチを検出可能な View の追加

次に，LiveCodingViewController.mm の viewDidLoad メソッド内に以下の記述を追加した．

```
self.tv = [[TouchView alloc]
    initWithFrame:CGRectMake(20, 200, 60, 60)];
self.tv.backgroundColor = [UIColor redColor];
self.tv.delegate_touches = self;
self.tv.sel_touches_began = @selector(touch:withEvent:);
[self.view addSubview:self.tv_red];
```

(3) タッチ検出時のメソッドの追加

続いて，以下のとおり，新たなメソッドを追加した．

```
- (void)touchRed:(NSSet *)touches
    withEvent:(UIEvent *)event
{
    iSCController *scc = (iSCController*)(self.target);
    [scc interpret:@"a = {SinOsc.ar()}.play"];
}
```

(4) ヘッダのインポート

さらに，参照するヘッダファイルを追加するため，以下の追記を行った．

```
#import "iSCController.h"
```

これらの追記を行ったことで，ビューをタッチした時のイベントをすべて LiveCodingViewController.mm 上に記述できるようになった．上記の追記では，タッチを検出すると正弦波を再生する赤色のビューが設置されるが，(1), (2), (3)の記述を複数回行うことで，独自のビューを任意の数配置した UI を構築することができる．

3.3 SC コード送出機能

LiveCodingViewController.mm に新たに追加したメソッド（前節(3)）では，SuperCollider コードを文字列情報として保持し，タッチを検出した際に SC サーバに送信する．「a = {SinOsc.ar()}.play」という文字列は，振幅 1，周波数 440Hz の正弦波を再生させる SuperCollider コードそのものである．

すなわち，ビューをタップすることで，SuperCollider サーバに任意のプログラムを送信できるようになった．

4. モーダルチェンジシステムの実装

4.1 モーダルチェンジ

サウンドファイルの加工や，MIDI でのトランスポーズなどで比較的困難な方法であり，かつ本システムの評価として容易な確認手法として，インタラクションによるモーダルチェンジを採用した．モーダルチェンジは，ダイアトニックスケール上での 2 か所の半音の位置が変化することで起きる音楽的現象である．C 音を基音としたモーダルチェンジの譜例を以下に示す．（図 6）

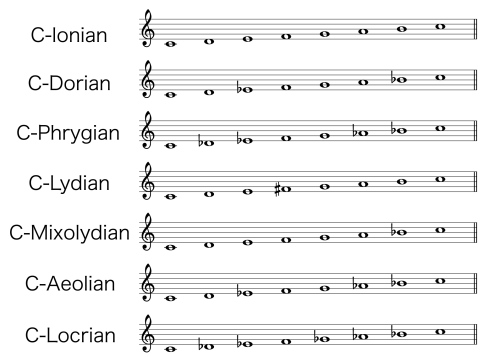

図 5　C を基音としたモーダルチェンジ

4.2 SuperCollider による音列再生

SuperCollider で音列を生成する方法はいくつかの方法が考えられる．しかし，Pbind，Ptpar などのパターンを構築する SuperCollider クラスは，再生する音列の情報をひとつの変数として一括してサーバに送信するので，情報の一部を改変することや，音列を再生しながら入れ替えることが難しい．したがって，リアルタイムなフレーズの変化には利用できない．

4.3 TempoClock による実装

リアルタイムなモーダルチェンジを行うため，モーダルチェンジの実装は，TempoClock クラスを使用して行った．モーダルチェンジを行う SuperCollider プログラムのうち，再生する音列を決定している部分を以下に示す．

```
// 再生するノートナンバーの定義
~offset = 60;
~seq1 = [0,2,3,5, 7,9,10,12]
~seq2 = [3,5,7,9, 10,12,14,15];

// カウンタの初期化
~ct = 0;
// テンポの設定
~bpm = 100;

// テンポを秒数に変換
~clock = TempoClock(~bpm/60);
//タイミング管理の開始
~clock.sched(0.0, {
   var delta, key1, key2;
   s.bind{
      if(~ct == 0){
         ~group = Group.new;
         ~syn = Synth.head(~group, "melody", [¥gate, 1]);
      };
      // 各配列内のノートナンバーの音を順に取得
      key1 = ~syn[0].wrapAt(~ct) + ~offset;
      key2 = ~syn[1].wrapAt(~ct) + ~offset;

      // 取得したノートナンバーにもとづき音を再生
      ~syn.set(¥pitch1, key1);
      ~syn.set(¥pitch2, key2);
      ~syn.set(¥gate, 1);
   };
   ~clock.sched(delta * 0.6, { s.bind{ ~syn.set(¥gate, 0) } }) ;
   // カウンタの値を増加
   ~ct = ~ct+1;
   delta;
});
```

このプログラムは，最初に再生する音の MIDI ノートナンバーと，テンポの情報を変数としてサーバに格納する．変数の先頭にチルダを付すことで，サーバに変数を格納している．音列再生のためのスケジューラである TempoClock を定義し，生成されるタイミングに応じて音を再生している．プログラム中の wrapAt メソッドを利用することで，格納した MIDI ノートナンバー情報を，順番通り繰り返して再生する．この方法によって，再生するフレーズをリアルタイムに変更し，なおかつ長さの異なるフレーズを同時再生することができた．しかし，音を再生するテンポ情報が全パートで共通であるため，発音のタイミングが異なる複数のパターンを再生することができない．

4.4 再生スケジュールの分割による改良案

前述の問題を解決するため，各パートにそれぞれ再生スケジュールをもたせたものも実装した．以下に音列再生部のプログラムを示す．

```
(
// 各パートの音量情報
~kkvol = [4];
~snvol = [4];
~hatvol = [4,2,3,2, 4,2,3,2, 4,2,3,2, 4,2,3,2];

// テンポの定義
t = TempoClock.new(120/60);

kk = t.schedAbs(t.beats.ceil,
   { |beat|
      a = Synth(¥kick,
         [
            vol: (~kkvol/4).squared.wrapAt(beat)
      ]);
      // 次は 1 拍後に再生
      1 });
sn = t.schedAbs(t.beats.ceil + 1,
   { |beat|
      b = Synth(¥snare,
         [
            vol: (~snvol/4).squared.wrapAt(beat)
      ]);
      // 次は 2 拍後に再生
      2 });
hat = t.schedAbs(t.beats.ceil,
   { |beat|
      c = Synth(¥hat,
```

Vol.2015-MUS-106 No.4
Vol.2015-EC-35 No.4
2015/3/2

```
        [
            vol: (~hatvol/4).squared.wrapAt(beat)
        ]);
        // 次は 0.25 拍後に再生
        0.25 });
)
```

この方法では，各パートに異なるデルタタイムを設定でき，パートごとに異なるタイミングでの発音を行うことができた．しかし，現状この手法では MIDI ノートナンバーをもつパートを再生した場合に Node を開放できない不具合があり，改善が必要である．

4.5 実装システム

上記のような技術的検討を経て，サウンドエンジン部に「wdkk/supercollider_iOS」を，描画オブジェクトには iOS のネイティブ API による描画を利用する共に，インタラクションに応じ supercollider のコードフラグメントを送信可能にする UI 制御クラス「TouchView」を組み合わせ，実装を行った．これにより，２声のフォブルドンのモーダルチェンジや，エレクトリックドラムのリズムパターンの動的変化などをリアルタイムで制御することが可能になった．

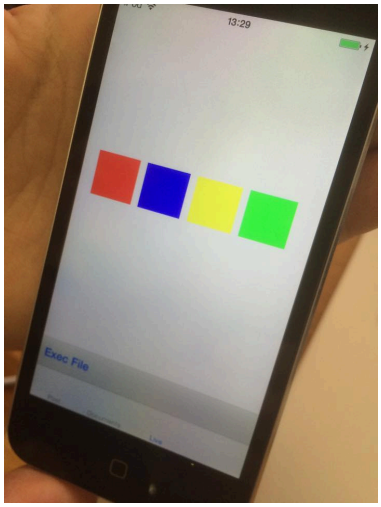

図 6　実装システム

5. おわりに

本研究では，音楽情報処理分野，特にアルゴリズミック・コンポジション研究の知的資産を，現代的なエンタテイメント・コンピューティング・プラットフォームであるスマートフォン上で活用するための一手段として，SuperCollider の iOS 上での動作調整，ネイティブ API から iOS 機内部の SuperCollider Server への SuperCollider コードフラグメントの送信方法，これらを技術的基盤として音楽的なモーダルチェンジがリアルタイムに実行可能かの３点に絞り研究を進め，技術的には概ね良好な結果を得た．これにより，機能和声音楽や現代音楽だけではなく，Jazz や Blues のアルゴリズム研究[8] を元にしたよりエンタテイメント性の高い adaptive music への応用が期待できる．また，現時点ではタッチインタラクションを中心とした評価だが，クラス化したことにより，多様なインタラクションに対応した評価環境が整備できた．現在はスマートフォン特有のインタラクションと音楽と関連についても研究が盛んであり[9]，こうした研究についても今後取り組みたい．

参考文献

1) Adaptive Music The Secret Lies within Music Itself, http://www.gdcvault.com/play/1012601/Adaptive-Music-The-Secret-Lies
2) The Dynamic Audio of Vessel, http://gdcvault.com/play/1015369/The-Dynamic-Audio-of
3) DirectX ソフトウエア開発キット, https://msdn.microsoft.com/ja-jp/library/ee416796%28v=vs.85%29.aspx
4) iOS Developer Program, https://developer.apple.com/jp/programs/ios/
5) Android Developer, http://developer.android.com/index.html
6) Csound for iOS, http://www.csounds.com/shop/csound-for-ios/
7) wdkk/supercollider_iOS, , https://github.com/wdkk/supercollider_iOS
8) Nierhaus, G. : Algorithmic Composition Paradigms of Automated Music Generation, Springer (2009).
9) Swift, B. : Chasing a Feeling: Experince in Computer Supported Jamming, Holland.S. et, al.(ed.): Music and Human-Computer Interaction, pp.85-100., Springer (2013).