研究報告

「クリエイティブ・ミュージック・コーディング」オーディオ・ビジュアル作品のための、オープンソースなソフトウエア・フレームワークの現状と展望

“CREATIVE MUSIC CODING”: OPEN-SOURCE SOFTWARE FRAMEWORKS FOR AUDIO VISUAL PERFORMANCES: CURRENT STATUS AND FUTURE PROSPECTS

田所 淳
Atsushi Tadokoro
慶應義塾大学
Keio University

概要

2000 年代に入って、コードを表現メディアとして使用するアーティスト達による活動が活発である。その多くは、Processing、openFrameworks、Cinder、vvvv といったオープンソースで公開されたソフトウエア・フレームワークを使用しており、そうしたフレームワークによる創作活動は「クリエイティブ・コーディング」と呼ばれ注目を浴びている。本稿では、こうしたクリエイティブ・コーディングの開発環境をコンピュータ音楽、特にオーディオビジュアル作品の創作に用いる手法について紹介し、さらに将来への展望を考える。

Since in the 2000s, it is worthy of notice that many artists use the programming language as their medium. They use open-source frameworks for artists such as Processing, openFrameworks, Cinder, vvvv. The coding process using these frameworks called “Creative Coding”. In this article, I describe various ways to use creative coding frameworks for computer music, especially for audio visual performances.

1. はじめに

1.1. クリエイティブ・コーディングとは

本稿では､クリエイティブ･コーディングとコンピュータ音楽との関わりについて取り上げていく。

コーディングとは、プログラミング言語を用いてソースコードを記述することである。人間がプログラミング言語を用いて指示を与えない限りは、コンピュータは何も生みだすことはできない。つまりコーディングをするということは、それ自体が何かを生みだすクリエイティブな行為である。では、クリエイティブ・コーディングとは何なのであろうか?

クリエイティブ・コーディングという用語は、まだ定着した用語とは言えず、その定義もあいまいである。まず初めに、この用語の本稿での定義について説明したい。

本稿では、クリエイティブ・コーディングという用語を、アーティストやデザイナー自身によって開発され活用されているコーディング環境と、その開発コミュニティーによる活動として用いる。創作のために既存のソフトウェアを使用するのではなく、コーディングを通してその道具自体を自らの手で創造する行為に重点を置く。

なぜアーティストやデザイナーがコーディングする必要があるのか。Processing の開発者の一人である Casay Reas は、次のように説明している [4, pp25]。

> Often, to realize a new or unique vision requires that artists and designers exceed the limitation of existing tools. Proprietary software products are general tools designed for the production of specific types of forms. [...] To go beyond these limitations, it is necessary to customize existing applications through programming or to write your own software.

つまり、クリエイティブ・コーディングとは、アーティスト自身がその表現の限界を打ち破るために自分自身でコーディングを行い、その結果生まれた創作のためのフレームワークである。既存のフォトレタッチ、ドローイング、3D モデリングといった特定の用途のため

のツールはここには含まれない。

1.2. オープンソース対プロプライエタリ

本稿ではオープンソースのフレームワークのみを対象にしたい。その理由は先にあげたクリエイティブ・コーディングの定義と関連している。

例えば、Max/MSP は多くのクリエイターによって支持され、作品が生みだされている。しかし、Max/MSP はプロプライエタリなアプリケーションであり、クリエイターが Max/MSP のアプリケーション自体を根本からカスタマイズすることはできない。対照的に、PureData はオープンソースで Miller Puckette を中心に開発されている。多くのクリエイターによってその機能が拡張され、Pd-extended としてリリースされている。

PureData と Max/MSP の対比からわかるように、オープンソースで公開されているフレームワークでなければ、その道具自体を自らの手で創造することができない。あくまでツールの枠組みの中での創造になってしまう。本稿ではそうした環境はクリエイティブ・コーディングからは除外したい。

1.3. コミュニティの存在

クリエイティブ・コーディングの開発体制は、DIWO(Do It With Others) というキャッチフレーズに象徴される。インターネットを介して世界中で繋がった開発者コミュニティーの存在が重要な意味を持っている。孤高のプログラマーが全てをコントロールするのではなく、世界中に分散した多くのコーダー達が、それぞれの得意分野に関係するパートを開発してコミュニティーに貢献していくという、自律分散的な開発体制が特徴である。

2. 主要な開発フレームワーク

現在使用されている、主要なクリエイティブ・コーディングのフレームワークをまとめてみたい。

Creative Applications [10] では、コードを主体として表現された様々なアートプロジェクトを紹介している。このサイトでは、作品を時系列に並べて紹介するだけではなく、カテゴリーごとに分類してタグ付けしている。開発に用いたフレームワークに関してもカテゴリーで分類されており、その件数を調べることで現時点での主要な開発フレームワークの人気をおおまかに捉えることが可能である。2014 年 1 月現在でのタグ付けされた開発言語とフレームワークを、掲載件数順に並べてみる。

1. openFrameworks (303)
2. Processing (296)
3. Flash (88)
4. Javascript (62)
5. MaxMSP (54)
6. Cinder (53)
7. vvvv (48)
8. Java (16)
9. Quarz Composer (10)
10. Three.js (6)
11. Unity (4)

もちろん、このランキングは Creative Applications のエディター個人に嗜好で収集されているものであり、客観的、普遍的なものではない。しかしながら、ネット上での評判をもとに膨大な件数から導き出されたものであり、いま創作現場で使用されている開発フレームワークのトレンドを知るための一つの指標として参考になるであろう。

この中で Flash、Max/MSP、Quarz Composer、Unity はプロプライエタリなものなので除外する。また、Javascript や Java は汎用プログラミング言語そのものであるので、これも対象から除外する。残った順位をまとめると、1. openFrameworks、2. Processing、3. Cinder、4. vvvv、5. Three.js の順である。

これらの開発フレームワークに関する情報を、表 1 にまとめた。

3. クリエイティブ・コーディングとコンピュータ音楽

3.1. 3 つの型

このセクションでは、こうしたクリエイティブ・コーディングなフレームワークをコンピュータ音楽作品の創作に使用する方法について考えていく。その構造から大きく 3 つのタイプに分類し、以下のように命名した。

1. 連携型 : Open Sound Control (OSC) でアプリケーション間を連携する方法
2. 拡張型 : フレームワーク内部で音響合成プログラムを作成する方法
3. 内包型 : フレームワーク内に、他の音響合成のエンジンを内包する方法

このセクションでは、それぞれのタイプごとに代表的な方法を紹介する。

3.2. 連携型

連携型は、OSC を使用して、独立する 2 つのアプリケーションを連携して使用する方法である。OSC を送受信することができるアプリケーションであれば相互に連携可能なので、音響合成のための専用のアプリケー

表 **1**. クリエイティブ・コーディング、主なフレームワーク

名称	リリース年	開発者	開発言語
vvvv [11]	1998	Joreg, Max Wolf, Sebastian Gregor, Sebastian Oschatz	C#
Desing by Numbers [12]	1999	John Maeda	Java
Processing [13]	2001	Ben Fry, Casey Reas	Java
openFrameworks [14]	2005	Zachary Lieberman, Theo Watson, Arturo Castro	C++
Cinder [15]	2010	The Barbarian Group	C++
Three.js [16]	2010	Mr.Doob	Javascript

ションと併用する様々な組み合わせが可能となる。

様々な組み合わせの中から、ここでは Processing と openFrameworks で OSC を送受信して、SuperCollider と連携する方法について説明する。

3.2.1. Processing で OSC を使用

Processing で OSC を使用するには、oscP5[17] ライブラリを追加する必要がある。oscP5 では送信先の IP と port 番号を設定して、任意のメッセージと値を組合せた OSC を自由に送信することができる。また、別のアプリケーションから送られてきた OSC メッセージを受信し、パースすることも可能である。

OSC メッセージを、PureData や SuperCollider など OSC を使用可能な言語と組合せることで、相互にメッセージを交換することが可能である。

一例として、Processing + oscP5 から、SuperCollider の楽器 (Synth) を演奏するサンプルを掲載する。まず SuperCollider 側で楽器 (Synth) を用意する。

```
SynthDef("test_inst",{
  arg freq=440, length=1.0, amp=0.5;
  var env, out;
  env = Env.perc(0.01, length);
  out = SinOsc.ar([freq,freq*1.001])
  * EnvGen.kr(env, doneAction:2) * amp;
  Out.ar(0, out);
}).store;
```

この楽器を Processing から演奏する。画面をマウスクリックすると音が鳴るようにプログラミングしてみる。Processing 側のコードは以下のようになる。

```
import oscP5.*;
import netP5.*;

OscP5 oscP5;
NetAddress remote;
int counter;

void setup() {
  size(400,400);
  oscP5 = new OscP5(this, 12000);
  remote = new NetAddress("127.0.0.1", 57110);
}
void draw() { }

void mouseReleased() {
  OscMessage msg = new OscMessage("/s_new");
  msg.add("test_inst");
  msg.add(1000 + counter);
  msg.add(1);
  msg.add(0);
  oscP5.send(msg, remote);
  counter++;
}
```

3.2.2. openFrameworks で OSC を使用

openFramworks で OSC を用いる際には、ofxOsc というアドオンを使用する。ofxOsc は openFramewokrs の配布パッケージに同梱されており、すぐに使用することが可能である。ofxOsc も oscP5 と同様、自由にメッセージと値のセットを組み立てられ、IP とポート番号で指定する任意の場所に送ることが可能である。また、OSC を受けとりそれをパースすることも可能である。

先程の例と SuperCollider の楽器 (test_inst) はそのままで、同じようにマウスを画面上でクリックすると OSC を送信して SuperCollider の楽器を演奏する ofxOsc を使用したサンプルを掲載する。

```
#include "testApp.h"

ofxOscSender sender;
int counter = 0;

void testApp::setup(){
  sender.setup("localhost", 57110);
}

void testApp::update(){ }

void testApp::draw(){ }

void testApp::mouseReleased(int x, int y, int button){
  ofxOscMessage msg;
  msg.setAddress("/s_new");
  msg.addIntArg(1000 + counter);
  msg.addIntArg(1);
  msg.addIntArg(0);
  sender.sendMessage(msg);
  counter++;
}
```

3.3. 拡張型

拡張型は、Processing や openFrameworks, Cinder などのフレームワークの機能自体を拡張して、音響合成やサウンドの入出力に対応させる方法である。それぞれのフレームワークには、音響合成のための拡張機能 (ライブラリ/アドオン) が存在する。それらの拡張機能により、フレームワーク単体で音響合成や、サウンドファイルの入出力などを行うことが可能となる。

様々な拡張機能が存在しているが、ここでは、Processing と openFrameworks の代表的な機能拡張 (ライブラリ/アドオン) をとりあげる。

3.3.1. *Processing + minim*

minim[18] は JavaSound API を利用した Processing の拡張ライブラリで、現在は Processing のパッケージに同梱されて配布されている。事実上 Processing の標準のサウンドライブラリとなっている。

minim は多岐にわたる機能をサポートしていて、オーディオファイルの読み書きと再生 (AudioPlayer, AudioRecorder)、リアルタイムのサウンド入出力 (AudioInput, AudioOutput)、FFT などが簡単に使用可能である。

さらには、リアルタイムの音響合成のためのユニットジェネレイターもいくつか実装されていて、簡単な音響合成も可能となっている。

minim での音響合成のサンプルとして、リアルタイムに FM 合成を行い出力する例を掲載する。

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
AudioOutput out;
Oscil fm;

void setup(){
  size(640, 480, P3D);
  minim = new Minim( this );
  out   = minim.getLineOut();
  Oscil wave = new Oscil(200, 0.8, Waves.SINE);
  fm   = new Oscil(130, 400, Waves.SINE);
  fm.patch(wave.frequency);
  wave.patch(out);
}

void draw(){
}
```

3.3.2. *openFrameworks + ofxTonic*

openFrameworks では、フレームワーク内部にサウンドファイルの再生 (ofSoundPlayer) や、リアルタイムなサウンド入出力 (ofSoundStream) の機能を備えている。スピードや音量を変化させてサウンドファイルを再生したり、オーディオ入力を C++ 操作して出力するといった処理が可能である。しかし、複雑な音響合成を行う場合、ユニットジェネレータやフィルタといった機能は用意されておらず、サンプル単位での計算を全て C++ で実装していく必要がある。

そのため、音響合成のためのパッケージをアドオンとして組込み利用する方法を用いる。openFrameworks では、Processing における minim のような定番のアドオンは存在していない。いくつか存在するアドオンの中から ofxTonic[19] を紹介したい。

ofxTonic は C++ で書かれた音響合成ライブラリ Tonic をアドオン化したものである。C++ で高速に演算される。その文法は、信号を「.」で接続してパッチングしていく直感的でわかりやすいものとなっている。

minim と同様、ofxTonic で FM 合成を行うコードをサンプルを掲載する。

```
#include "testApp.h"

void testApp::setup(){
  ofSoundStreamSetup(2, 0, this, 44100, 256, 4);

  SineWave car = SineWave();
  SineWave mod = SineWave();
  mod.freq(130);

  float basePitch = 400;
  float index = 200;
  Generator frequency = basePitch + (mod * index);
  car.freq(frequency);
  synth.setOutputGen(car);
}

void testApp::update(){
}

void testApp::draw(){
}
```

3.4. 内包型

内包型は、連携型に似ている。連携型と同様、OSC を介してアプリケーション間を連携する。しかし内包型が連携型と異なる点は、音響合成のための機能をフレームワークの内部に取り込んでしまい、単一のアプリケーションとして生成される部分である。

この方法は、音響合成言語がオープンソースであり、さらにその開発言語がフレームワークと同一である必要がある。フレームワーク側は音響合成のソースを内部に統合しその機能を OSC を介して呼びだせるようにしている。

この内包型の例として、openFrameworks と ofxPd の組み合わせ、そして、openFrameworks と ofxSuperCollider と ofxSuperColliderServer を統合する方法を取り上げる。

3.4.1. *openFrameworks + ofxPd*

Pure Data は C 言語を用いて開発されている。openFramworks は C++ ベースなので、C 言語で記述された関数を呼び出して使用することが可能である。ofxPd は、Pure Data の C 言語で書かれた音響合成機能を openFrameworks に内包し、Pure Data のパッチを読込み生成された音響を openFrameworks を経由して出力する。さらに、Pure Data のパッチに対して openFrameworks からパラメータを送出することも可能である。

Pure Data で作成したパッチを、openFrameworks に内包し音響生成するサンプルを掲載する。

Pure Data 側は外部から周波数を読み込んで、Sin 波を生成するシンプルなものを作成し、「osc.pd」というファイル名で保存する。 [図 1]

このパッチを ofxPd を経由して音響生成するには、openFrameworks 側で以下のようなプログラムを作成する。この例ではマウスの y 座標で周波数の値を決定し、内包した Pd のパッチに値を送信している。

```
#include "testApp.h"

void testApp::setup() {
```

```
  ofSetFrameRate(60);
  ofSoundStreamSetup(2, 1, this, 44100, 2048, 4);
  pd.init(2, 1, 44100);
  Patch patch = pd.openPatch("pd/osc.pd");
  pd.start();
}
void testApp::update() { }
void testApp::draw() { }

void testApp::exit() {
  pd.stop();
}

void testApp::mouseMoved(int x, int y) {
  float freq = ofMap(y, ofGetHeight(),
                     0, 100, 8000);
  pd.sendFloat("freq", freq);
}

void testApp::audioReceived
(float * in, int buf, int n) {
  pd.audioIn(in, buf, n);
}

void testApp::audioRequested
(float * out, int buf, int n) {
  pd.audioOut(out, buf, n);
}
```

3.4.2. openFrameworks + ofxSuperCollider + ofxSuperColliderServer

内包型のもう一つの例として、openFrameworks の中に SuperCollider を内包する方法がある。もともと ofxSuperCollider という SuperCollider の OSC に特化したアドオンが存在する。ただしこのアドオンだけでは単純に SuperCollider の音響合成サーバーである scsynth に OSC メッセージを送るだけである。ofxSuperCollider と共に ofxSuperColliderServer を使用することで、scsynth を openFrameworks に内包することが可能となる。使用法はとても簡単で、SuperCollider で定義した楽器をデータフォルダに格納して、openFrameworks 側で scsynth のサーバーを起動した後、内包した Synth に対して OSC を送出する。

```
#include "testApp.h"

#include "ofxSuperCollider.h"
#include "ofxSuperColliderServer.h"

ofxSCSynth *synth = NULL;

void testApp::setup(){
  ofxSuperColliderServer::init();
  synth = new ofxSCSynth("test_inst");
  synth->create();
}
```

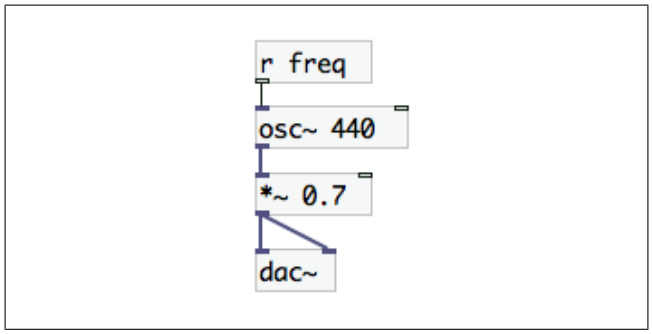

図 **1**. ofxPd 連携用 Pure Data パッチ

```
void testApp::update(){
}

void testApp::draw(){
}

void testApp::mouseMoved(int x, int y ){
  if(synth){
    synth->set("freq",
               ofMap(y, ofGetHeight(), 0, 100, 8000));
  }
}
```

4. クリエイティブ・コーディングの利点

以上で概観してきたように、様々な組み合わせ方法により、クリエイティブ・コーディングのフレームワークと音響合成言語を融合できることがわかった。では、こうした環境により何が可能となるのか考えてみたい。

4.1. オーディオビジュアル

openFrameworks、Processing、Cinder はビジュアル表現のために特化している。こうしたフレームワークと音響合成言語を組み合わせることで、高度なオーディオ・ビジュアル表現が可能となる。例えば、形態を生成する一つのクラスを音響合成言語の楽器と対応させ、色や形態、動きなどと音響が一対一で厳密に関連付けられた表現を容易に行うことができる。さらに、GLSL(OpenGL Shading Language)、OpenCL、framebuffer object などを活用して、マルチコア CPU や GPU の能力をフルに活用した高度グラフィクスプログラミングと音響を同時に生成するアプリケーションが開発可能である。

4.2. デバイスとの連携

デバイスとの連携という点においても、クリエイティブ・コーデイングのフレームワークを用いることは効果的である。例えば、Microsoft の Kinect や Leap Motion など、高性能で安価な入力デバイスが次々と開発、発売されている。こうしたデバイスには、それぞれ開発用 API が容易され公開されている。公開された API は、開発コミュニティーによって先を争ってフレームワークの拡張機能として統合される。その拡張機能を使用するフレームワークに読み込むだけで、様々な入力デバイスを作品に活用することが可能である。

5. 将来の展望

5.1. ネットワーク

HTML5 と総称される次世代の Web 技術の浸透により、Web ブラウザ内で高度な音響合成が可能となってきている。Web Audio API は Web ブラウザでの低レイテンシーのサウンド入出力に対応している。また Web

MIDI API は、MIDI デバイスと連携して MIDI 信号を処理することができる。こうした API に加え、WebSocket プロトコルを使用することで、これまでの http では難しかったクライアントとサーバー間のコネクションを維持したままでの双方向通信が可能となる。さらには、WebGL を使用することで GPU でアクセラレーションされた 3D グラフィクスが使用可能である。将来的には Web ブラウザ単体で、高度なオーディオ・ビジュアル表現の開発し、そのまま作品としてネットワーク上で発表することも容易にってくるであろう。

例えば、SuperCollider を Web ブラウザに移植しようという、CoffeeCollider[20] というプロジェクトなどは、Web ブラウザでの音楽の可能性を示している。

5.2. ライブ・コーディング

Pure Data や、SuperCollider + JITlib、ChucK など、コンピュータ音楽用の言語ではライブコーディングを行うことが可能なものも多い。また、impromptu[23] や、その後継の extempore[24] といったライブ・コーディングに特価した言語も開発されている。その一方で、Processing や、opneFrameworks、Cinder といったクリエイティブ・コーディングのフレームワークは、基本的には、プログラムを実行したままコーディングを進めることはできない。

今後は、単純化された文法と拡張性を備えつつ、ライブコーディング可能なフレームワークが増えてくると予想する。また、その環境は音響合成だけでなく、3D グラフィクス、動画、デバイスからの入力など様々な機能がパッケージングされていくであろう。そうした多岐にわたる機能を駆使しつつ、ライブコーディングで物語を紡ぐようにオーディオ・ビジュアルを創作していくというのが未来のクリエイティブ・コーディングになっていくのではないだろうか。

6. 参考文献

[1] John Maeda, *Design by Numbers*, The MIT Press, 2001.

[2] John Maeda, *Creative Code: Aesthetics + Computation*, Thames & Hudson, 2004

[3] Casey Reas, Ben Fry, *Processing: A Programming Handbook for Visual Designers and Artists*, The MIT Press, 2007.

[4] Casey Reas, Chandler McWilliams, LUST, *Form+Code in Design, Art, and Architecture*, Princeton Architectural Press, 2010

[5] Hartmut Bohnacker, Benedikt Gross, Julia Laub, Claudius Lazzeroni, *Generative Gestaltung*, Schmidt Hermann Verlag, 2009

[6] Manuel Lima, *Visual Complexity: Mapping Patterns of Information*, Princeton Architectural Press, 2011

[7] Collins, N., McLean, A., Rohrhuber, J. & Ward, A. *Live Coding Techniques for Laptop Performance*, Organised Sound 8(3): pp 321-30. Cambridge University Press 2003

[8] Lonce Wyse, *The Viability of the Web Browser as a Computer Music Platform*, Computer Music Jornal: Vol. 37, No. 4, pp.10-23, The MIT Press 2013.

[9] Ben Fry, *computational information design*, http://benfry.com/phd/, 2004

[10] CreativeApplications.Net, http://www.creativeapplications.net/

[11] vvvv, http://vvvv.org/

[12] Design by Numbers, http://dbn.media.mit.edu/

[13] Processing.org, http://processing.org/

[14] openFrameworks, http://openframeworks.cc/

[15] Cinder — The library for professional-quality creative coding in C++, http://libcinder.org/

[16] Three.js, http://threejs.org/

[17] oscP5, http://www.sojamo.de/libraries/oscP5/

[18] Minim, http://code.compartmental.net/tools/minim/

[19] ofxTonic, https://github.com/TonicAudio/ofxTonic

[20] CoffeeCollider, http://mohayonao.github.io/CoffeeCollider/

[21] Gibber, http://gibber.mat.ucsb.edu/

[22] Toplap, http://toplap.org/

[23] impromptu, http://impromptu.moso.com.au/

[24] extempore, https://github.com/digego/extempore

6.1. 著者プロフィール

田所 淳 (TADOKORO Atsushi)

1972 年千葉県生まれ。多摩美術大学美術学部情報デザイン学科非常勤講師、東京藝術大学藝術情報センター非常勤講師。慶應義塾大学政策・メディア研究科博士課程在籍。同研究科助教（有期・研究奨励 II)。著書に「Beyond Interaction［改訂第 2 版］- クリエイティブ・コーディングのための openFrameworks 実践ガイド（共著)」など。コードによる生成的なオーディオビジュアル表現を探求。

http://yoppa.org/

Atsushi Tadokoro, Born in Chiba, Japan in 1972, is part-time lecturer at Tama Art University and Tokyo University of the Arts. September 2013 to current, Ph.D. program in Media and Governance, Keio University.

http://yoppa.org/