6

WiiFlashを使いこなそう［Processing編］

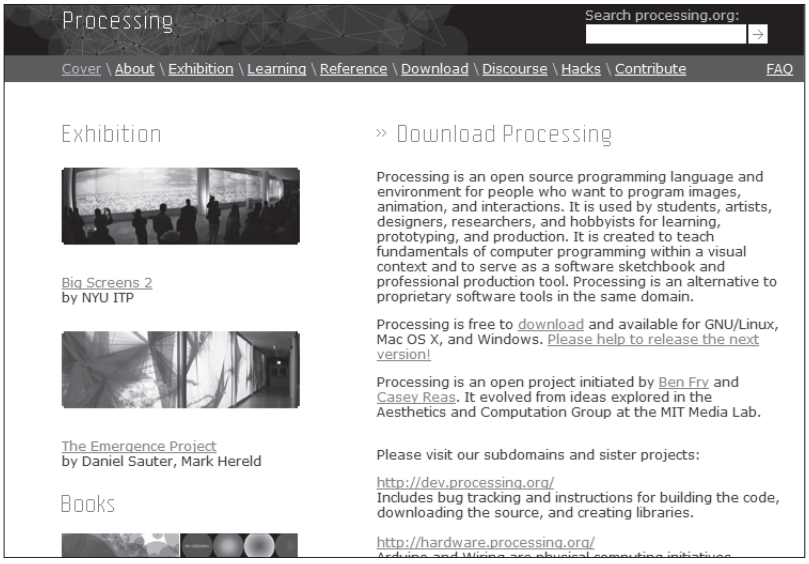
6章：木村秀敬

この章では、WiiFlashをFlash（ActionScript環境）ではなく、Processingを使って利用する方法を解説してしていきます。

6.1 ProcessingからWiiRemoteにつなぐ

Processingは、マサチューセッツ工科大学のBen Fry氏とCasey Reas氏によって開発されたフリーのプログラム開発環境です。映像を作り出す、インタラクティブな作品を作るといった用途に向いており、アーティストやデザイナーでも使いやすいように作られています。

図6-1　Processing.org（http://processing.org/）



ベースはJava言語なので、Linux、Mac OS X、Windowsと多様な環境で動作します。Javaで開発というと、JDKをインストールしたり、コマンドラインを使って操作したりといったことが必要では……と思われてしまうかもしれませんが、Processingはそこの敷居を下げるための工夫がなされています。そのため、Javaはもとより、今までプログラミング言語を使ったことがないと

いう方にも向いています。

また、プログラムについても「スケッチ」という呼び方を用い、サッと書いてすぐに実行できる点を強調しています。Java言語の入門にありがちなおまじない的なプログラムは不要で、いきなり主目的となる部分から書き始めることができます。このあたりは、JavaというよりもPerlやPythonなどの軽量言語のイメージです。

このように簡単に使えるProcessingですが、低機能かというとそういうことはありません。カメラからの動画入力や、ネットワークとの通信、OpenGLを使った3Dプログラミングなど、たくさんの機能がライブラリによってサポートされています。WiiRemoteも例外ではありません。ProcessingからWiiRemoteを扱うためのライブラリとして、Wrj4P5というライブラリが用意されています。

Wrj4P5
URL http://sourceforge.jp/projects/wrj4p5/wiki/FrontPage

しかし、このライブラリを使うために必要なBluetoothスタックなどはかなり限定されており、必ずしもすべての環境で動くとは限りません。特に、Windows環境では動かないことが多いようです。

そこでWiiFlashの登場です。前章で紹介されている通り、WiiFlashはもともとFlashからWiiRemoteを扱うためのツール、ライブラリですが、実は他のプログラミング言語からも使うことができます。一度仕組みさえ知ってしまえば、WiiFlashはWiiRemoteに対する汎用のインタフェースとして機能します。

Processingも例外ではなく、.NETライブラリを使うことでWiiFlash経由でWiiRemoteを扱うことができるようになります。筆者の感覚では、WiiFlashのほうがWrj4P5よりも多くの環境で動作しているようです。WiiFlashを使えば、より多くの環境でProcessingからWiiRemoteを扱うことができるのではないかと思います。

6.2 ProcessingからWiiFlashを使うには

ProcessingからWiiFlashを経由してWiiRemoteにつなぐために必要な環境は以下のとおりです。

- WiiFlashが動く環境
- Processing

ここでは、Processingのセットアップ方法と、WiiRemoteを扱うための方法について説明します。WiiFlashについては前章を参考にセットアップしてください。また、Processingそのものについての詳細な説明も割愛します。

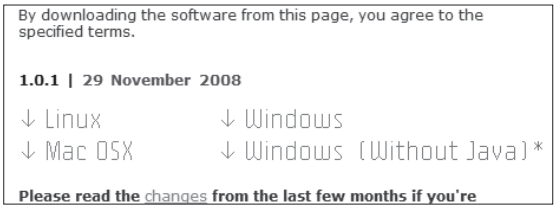
Processingのセットアップ

Processingのセットアップはとても簡単です。ProcessingのHPから圧縮ファイルをダウンロードしてきて解凍するだけです。特にインストーラを使ってインストールするなどの作業は必要ありません。ダウンロードは、以下のURLから行うことができます。

Download Processing
URL http://processing.org/download/index.html

ここから、使っているOSに合ったファイルをダウンロードしてください。Windowsについては注意書きにも書かれているとおり、特に理由がなければ無印のほう（「Without Java」）ではないほうをお勧めします。

図 6-2　Processing 公式 HP におけるダウンロード選択



なお、執筆時点での最新バージョンは 2008 年 11 月 29 日公開の 1.0.1 なので、以後の説明はこの 1.0.1 を対象とします。みなさんがこの文章を読まれているときの最新バージョンはすでに 1.0.1 ではなくなっているかもしれませんが、ご了承ください。

ダウンロードした圧縮ファイルを展開すると、図 6-3 のようなファイルが現れます。あとは processing.exe を起動するだけです。

図 6-3　processing.exe を実行

起動中のスプラッシュウィンドウが表示された後、図 6-4 のような画面になれば成功です。

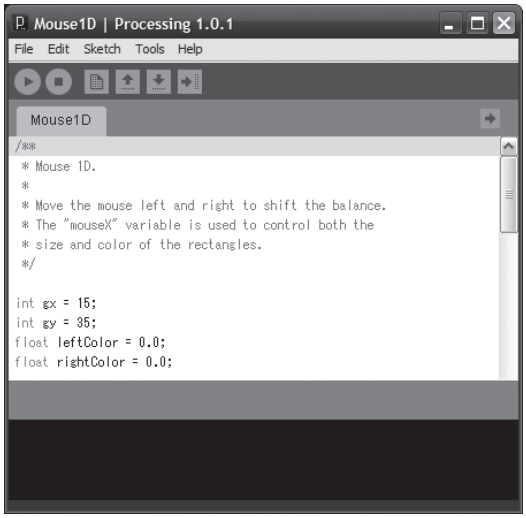
図 6-4　Processing 起動直後

6.3 Processingのサンプルを動かす

まずはWiiFlashを使わない、Processingそのもののサンプルを動かしてみましょう。試しにProcessingに付属している「Mouse1D」というサンプルを動かすことにします。

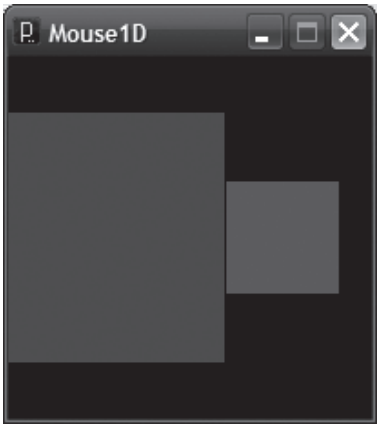
メニューから［File］の［Examples］を選択すると、とても多くのサンプルスケッチが用意されていることがわかります。この中から［Basics］→［Input］→［Mouse1D］を選択します。すると、50行ちょっとのスケッチ（ソースコード）が開かれます。

図6-5　Mouse1D



中身の理解は置いておいて、まずは実行してみましょう。左上のRunボタンを押すと、スケッチが実行されます。小さなウィンドウに2つの正方形が現れたでしょうか。マウスカーソルを左右に動かすと、それに反応して色やサイズが変化します。

図6-6　Mouse1D実行の様子



一通り、このサンプルがどのような動作をするか理解したら、ウィンドウを閉じてスケッチのほうに目を向けてみます。Javaやそれに近い言語を使ったことがある方なら、このスケッチがsetupメソッド、drawメソッド、updateメソッドの3つで構成されていることがわかるかと思います。

setupメソッドでは、ウィンドウサイズの設定などの初期化を行っています。drawメソッドでは、背景を塗りつぶし、マウスの位置に応じて2つの正方形を描く処理を行っています。updateメソッドでは、正方形の色や座標計算に使うパラメータを計算しています。

updateメソッド内の詳細な計算については特に理解しなくてもいいのですが、この引数にmouseXという変数を指定しているところに注目してください。mouseXとは、名前のとおり、現在のマウスカーソルのX座標です。カーソル座標をupdateメソッドに渡すことによって、leftColor、rightColor、gx、gyといった変数に適切な値が設定されます。そしてそれらの値を使って正方形が描かれます。

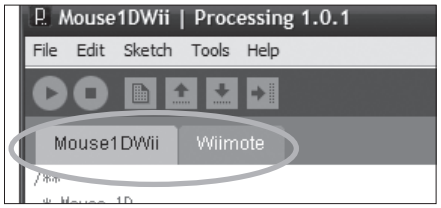
さて、このサンプルではマウスカーソルのX座標が使われていますが、これをWiiRemoteを使って操作できないものでしょうか？　次は、WiiFlashを使ってそれを実現する方法について説明します。

6.4 WiiFlashを使ったサンプルを動かす

サンプルスケッチのMouse1DWiiを開いてみてください。これを動かすには、まずWiiFlashを起動する必要があります。前章を参考に、BluetoothでWiiRemoteを認識させ、WiiFlashを起動してください。正しくWiiFlashが起動したら、ProcessingのRunボタンを押してください。表示される画面はまったくMouse1Dと変わりませんが、WiiRemoteをひねるように回転させると、マウスカーソルを動かしたときのように画面が変化します。

それではスケッチを詳しく見ていきましょう。スケッチには、以下のようにMouse1DWiiとWiimoteの２つのタブがあります。

図6-7　Mouse1DWiiとWiimoteの２つのタブ



Wiimoteのほうは後で詳しく説明するので、まずはMouse1DWiiのほうに着目してください。

Mouse1DWiiは、基となっているMouse1Dを少し書き換えたものです。以下のように４行の追加、変更があります。

コード6-1　Processing Mouse1DWii（追記部分）

```
// 前略
  float leftColor = 0.0;
  float rightColor = 0.0;
  Wiimote wiimote; ――――❶ WiiRemoteを使うための変数
  void setup(){
    size(200, 200);
    colorMode(RGB, 1.0);
    noStroke();
    wiimote = new Wiimote(this); ――――❷ 初期化
```

次ページにつづく

```
}

void draw() {
  wiimote.update(); ──────❸ WiiRemoteからの入力を取り込む
  background(0.0);
  update((int)((wiimote.x + 1) * width / 2)); ──────❹ 加速度センサーの値を使う
// 後略
```

まず最初に、WiiFlash経由でWiiRemoteを使うため、Wiimote型のwiimoteという変数を定義しています（❶）。setupメソッドでは、この変数を初期化しています（❷）。

drawメソッドではまず、WiiRemoteからの入力を取り込むために、Wiimoteクラスのupdateメソッドを呼び出します（❸）。

updateメソッドの詳細にはここでは踏み込みませんが、これによって現在のWiiRemoteの加速度などのパラメータが使えるようになります。たとえば、WiiRemoteを正面に向けたときにひねる方向（X方向）の加速度は、wiimote.xという変数に入っています。

最後は、描画する正方形のサイズに加速度を反映させる部分です（❹）。元のMouse1Dでは、updateメソッドにマウスカーソルのX座標（mouseX）を渡していました。Mouse1DWiiでは、これにならう形でWiiRemoteのX方向の加速度を変換し、updateメソッドに渡しています。

X方向の加速度を表すwiimote.xは、完全に左に傾けたときに－1、逆に完全に右に傾けたときに1という値になります。勢いがついていたりするとこの範囲を超えますが、基本的には－1から1の範囲と考えて問題ありません。スケッチに書いたとおり、1を足して画面幅（width）をかけ、2で割れば、－1から1の範囲を0から画面幅の範囲に変換できます。

Wiimoteの使い方

WiiRemoteをProcessingから使うためには、Wiimoteクラスを使います。Wiimoteクラスは、元のWiiFlashの実装を参考に筆者がオリジナルで作成したもので、動作については無保証とします。

このWiimoteを使うためのステップは大きく分けて3つです。

初期化

newでWiimoteのコンストラクタを呼び出し、Wiimote型のインスタンスを作ります。このとき、内部ではWiiFlashとの接続が行われるので、WiiFlashが起動していないとここでエラーが

発生し、以降の処理がうまくいきません。この処理は基本的に1度だけ呼べばいいので、setupメソッドの中で呼び出すのがよいでしょう。

コード6-2 Processing Wiimoteインスタンスの作成

```
Wiimote wiimote; // 変数の宣言
wiimote = new Wiimote(this); // setupメソッド内などで
```

データの取得

Wiimoteのupdateメソッドを呼び出し、WiiRemoteから、WiiFlash経由でデータを取得します。この処理は一般的にはdrawメソッドの中で一度だけ行います。

コード6-3 Processing WiiFlash経由でデータの取得

```
// drawメソッド内などで
wiimote.update();
```

一度updateメソッドを呼んでから次にupdateメソッドが呼び出されるまでは、WiiRemoteクラスから得られるWiiRemoteの状態は変わらないままです。

データの利用

Wiimoteのxやyなどのフィールドを参照することで、現在のWiiRemoteの状態を知ることができます。主に用いられるのは、各方向の加速度と、ボタンの状態でしょう。

コード6-4 Processing WiiRemoteの状態を取得

```
// (100, 100)の点からWiiRemoteの向きに応じて線を引く
line(100, 100, 100 + wiimote.x * 50, 100 + wiimote.y * 50);
```

Wiimoteクラスを使って得られるデータは次の表の通りです。

表6-1　Wiimoteクラス

フィールド名	型	意味
x, y, z	float	各軸加速度。-1～+1の値をとり、1Gのときに値が1となる
one, two, a, b, plus, minus, home, up, down, right, left	Button	それぞれのボタンの状態
batteryLevel	float	バッテリーの残量
extensionType	int	拡張コントローラーのタイプ

ボタン関連のフィールドは、Buttonクラスを使って表されています。Buttonクラスには、pressedとpushedの2つのboolean変数があります。

●pressed

ボタンが押されている間trueになります。押したままで何か動作をさせたいときに使います。

●pushed

ボタンが押された最初の1回だけtrueになり、Wiimoteのupdateメソッドが呼び出されるとfalseになります。ボタンを押したときに一度だけ動作させたいときに使います。

これら2つの変数タイプをうまく使い分けてください。

コード6-5　Processing　上下ボタンが押されている間パラメータを上下させる

```
// 上下ボタンが押されている間パラメータを上下させる
if (wiimote.up.pressed) {
   someparam++;
} else if (wiimote.down.pressed) {
   someparam--;
}
if (wiimote.a.pushed) {
   background(10); // Aボタンが押されたら画面を消去
}
```

6.5 サンプルスケッチを WiiRemote 対応にする

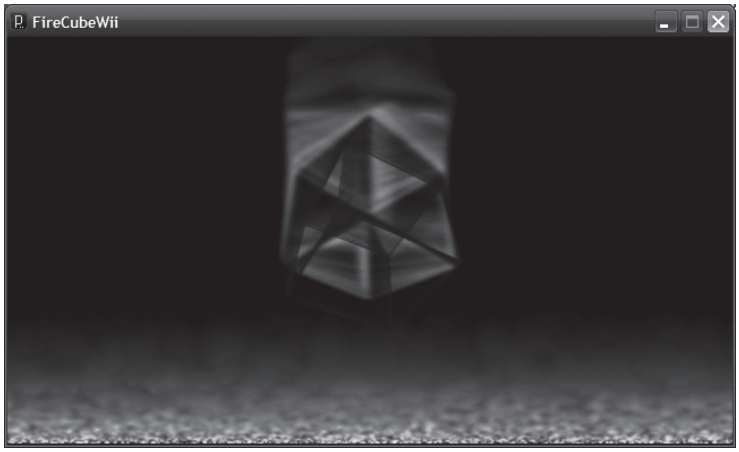
WiiRemote を応用したスケッチを作るにあたって、1 から Processing のスケッチを組んでそれを WiiRemote 対応にすると、Wiimote を使うという点からフォーカスがずれてしまいます。幸い、Processing には多数のサンプルスケッチが添付されているので、これらのうちいくつかを WiiRemote 対応にすることで解説していきます。

ここでは、FireCube、Directional、LightsGL、SineWaveSignal の 4 つのスケッチを WiiRemote に対応させた例を紹介します。WiiRemote に対応させたスケッチは、元のサンプルスケッチと区別するために、名前の最後に Wii を付けています。

FireCubeWii

FireCube は、炎が燃え上がるような複雑なエフェクトが 100 行程度で書かれたものです。このサンプルを、WiiRemote を振れば振るほど炎が出てくるようにしてみました。サンプルは FireCubeWii です。

図 6-8　FireCubeWii



drawの中では、加速度からpowerという値を計算しています。この値に応じて炎の出る量が変わります。

コード6-6 Processing 加速度からpowerという値を計算

```
float power = sqrt(wiimote.x * wiimote.x +
 wiimote.y * wiimote.y +
 wiimote.z * wiimote.z);
power = constrain(power - 1, 0.1, 1);
```

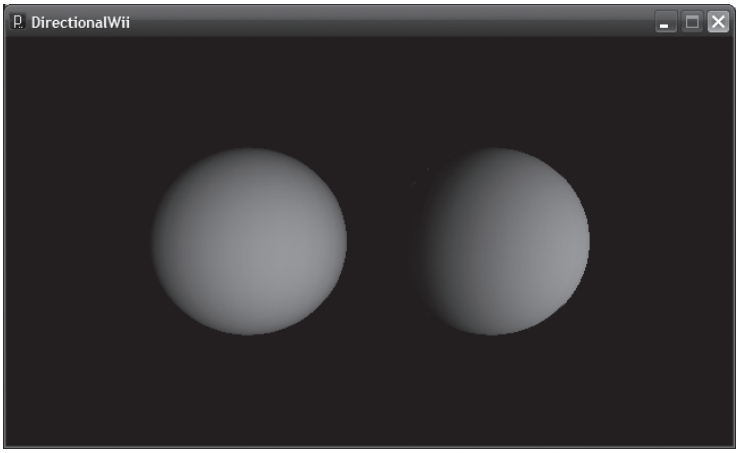
最初に計算しているのはWiiRemoteにかかっている加速度です。WiiRemoteが動かない状態では、この値がおよそ1になります。この状態では炎を出したくないので、次の行で1を引くとともに、constrainという関数を使って値を0.1から1の間に収めています。

ここで計算したpowerは、立方体と、下から上がってくる炎の初期値に設定します。詳しくはスケッチ中のpowerという変数を検索してみてください。

DirectionalWii

Directionalは、マウスカーソルの位置に応じてピンポン球のようなものがライトアップされるものです。これにWiiFlashから得られるX方向、Y方向の加速度を使うことで、あたかもWiiRemoteが懐中電灯になったかのような感覚が味わえます。

図6-9 DirectionalWii



このスケッチでは、懐中電灯のような感覚を出すために、WiiRemote の A ボタンが押されているときだけ球が表示されるようにしています。次のように、A ボタンが押されていれば draw メソッドから途中で抜け出します。

コード 6-7　Processing　A ボタンが押されていれば draw メソッドから抜け出す

```
if (!wiimote.a.pressed) {
 return;
}
```

光を当てる方向については、加速度の値をそのまま使っています。

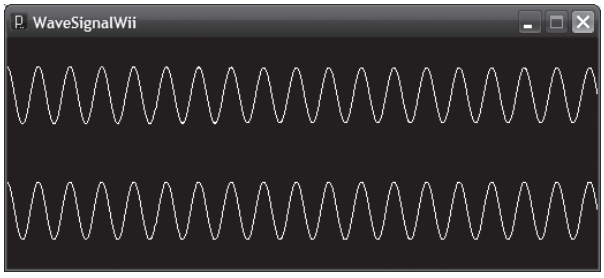
コード 6-8　Processing　光を当てる方向を出す

```
directionalLight(204, 204, 204, wiimote.x, wiimote.y, -1);
```

WaveSignalWii

最後は 1 つ変わったネタを取り上げます。WaveSignalWii は今までのスケッチとは異なり、音を使ったスケッチです。元となっているスケッチは［Libaries］→［Minim (Sound)］→［SineWaveSignal］です。このスケッチを起動すると、プーというような音が鳴り出します。これは画面にも表示されている通り、いわゆる正弦波です。マウスカーソルを上下左右に動かすことで、音の高さ（ピッチ）が上下したり、ステレオの左右のバランス（パン）が移動したりするのがわかるでしょうか。

図 6-10　WaveSignalWii



Processingでこのように音が鳴らせるのは、Minimというライブラリのおかげです。正弦波を表す変数を作り、ピッチやパンを指定するだけで、このように音を鳴らすことができます。

SineWaveSignalはその名の通り正弦波にしか対応していないのですが、これをWiiRemoteに対応させたWaveSignalWiiでは、のこぎり波や矩形波にも対応させました。左右ボタンを押すことでこれらを切り替えることができます。また、上下ボタンで音量を変えることもできます。

さらに、向きを変えることでピッチやパンも変化します。上に向けると音が高く、下に向けると音が低くなります。左右方向に向けると、そちらのほうから音が聞こえてくるようになります。

WiiFlashがProcessingからも使える理由

さて、ここまで特にこの話の詳細には踏み込んできませんでしたが、なぜProcessingからWiiFlashを使うことができるのでしょうか？　答えはWiiFlashに添付されているソースコードの中にあります。

メインとなるソースコードは、Core/api/source-classes/org/wiiflashディレクトリの中にあります。ここにあるのはWiiFlashそのもののソースコードではなく、FlashからWiiFlashにつなぐためのActionScript 3のソースコードですが、これだけでWiiFlashの挙動を推測することができます。Wiimote.asとWiiSocket.asを見れば、基本的な挙動をつかむことができます。

WiiSocket.asでは、connectメソッドでlocalhostの19028番に接続しています。このことから、WiiFlashが19028番のポートを使ってサーバを立てていることがわかります。そしてソケットからデータを受信したときの処理はonSocketDataメソッドに書かれています。ここを見ると、データが80バイト単位で受信されていることがわかります。最初の1バイトはコントローラーのIDとなっており、それ以降のデータはWiimoteクラスのupdateメソッドで読み込まれています。

WiiFlashから送られてくる基本的なデータをまとめると、次ページの表6-2のようになります。

表6-2　基本データレイアウト

名前	位置	型	意味
index	0	byte	コントローラーのID
batteryLevel	1	byte	バッテリー残量
buttonState	2	ushort	ボタンの状態
x	4	float	X方向の加速度
y	8	float	Y方向の加速度
z	12	float	Z方向の加速度
extensionType	16	byte	拡張タイプ

拡張タイプには、ヌンチャク、クラシックコントローラー、WiiBoardがあります。このタイプによって、17バイト目以降のデータの解釈方法が変わります。これらについてはNunchuk.as、ClassicController.as、BalanceBoard.asを見るとデータの内容がわかりますが、本書ではそれらについての説明は割愛します。

ProcessingからWiiFlashにつなぐためには、.NETライブラリのClientクラスを使います。Clientクラスを使うことで、ソケットを使ってWiiFlashと通信することができます。つないだ後は、readメソッドなどを使うことでWiiFlashからバイト列を読み込むことができます。

Wiimoteクラスの詳細な実装については本章では述べませんが、ソースコードはProcessing上で見られる状態となっているので、気になる方はそちらを参照してみるとよいでしょう。