

6. WiiFlash を使いこなそう (Processing 編)

デザイン後送

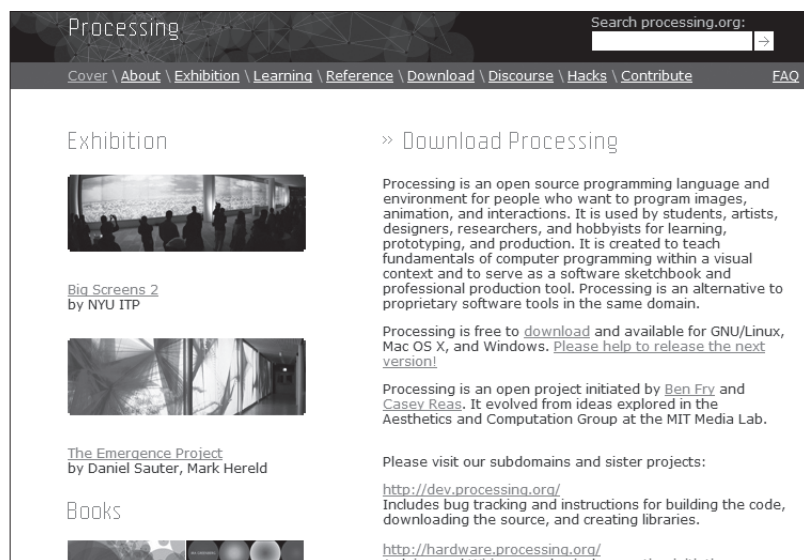
この章では、WiiFlashをFlash (ActionScript 環境) ではなく、Processing を使って利用する方法を解説してしていきます。

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 と Casey Reas 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……と思われるかもしれませんが、Processing はそ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の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についても割愛します。

Processing のセットアップ

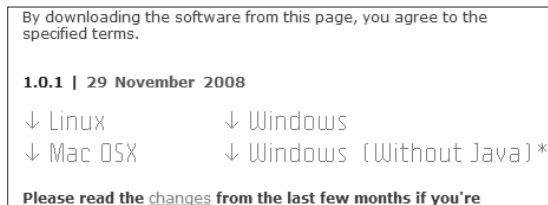
Processing のセットアップはとても簡単です。Processing の HP から圧縮ファイルをダウンロードしてきて解凍するだけです。特にインストーラを使ってインストールするなどの作業は必要ありません。ダウンロードは、以下の URL から行うことができます。

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。

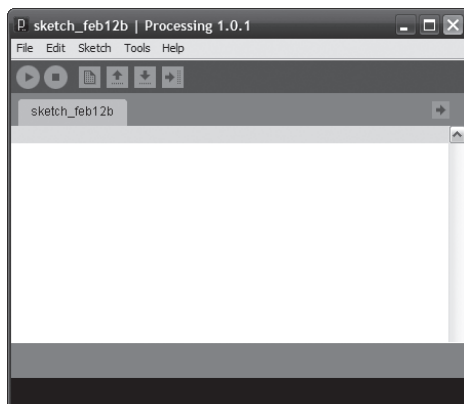
ダウンロードした圧縮ファイルを展開すると、図6-3のようなファイルが現れます。あとはprocessing.exeを起動するだけです。

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



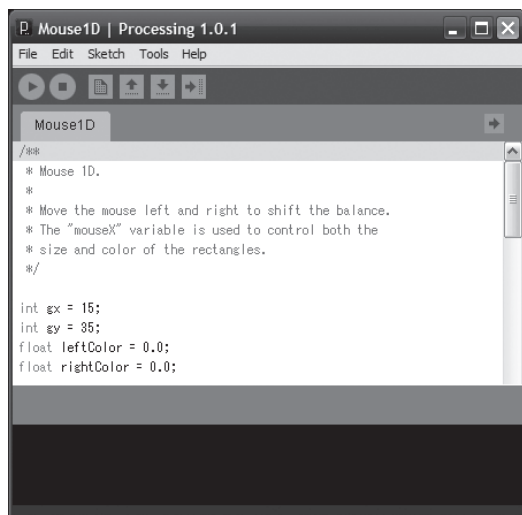
6.3

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。

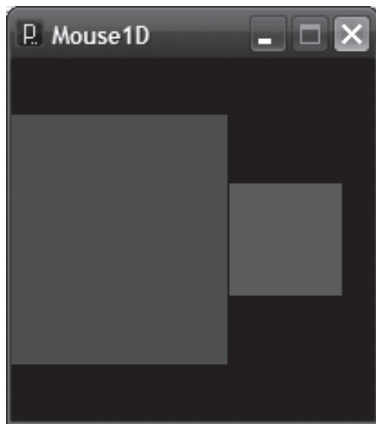
メニューから [File] の [Examples] を選択すると、とても多くのサンプルスケッチが用意されていることがわかります。この中から [Basics] → [Input] → [Mouse1D] を選択します。すると、50 行ちょっとのスケッチ (ソースコード) が開かれます。

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なのでしょうか？ 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。

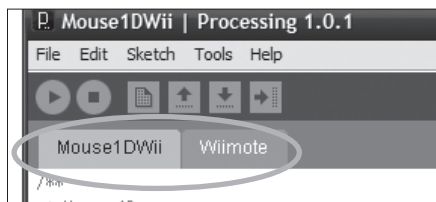
6.4

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度だけ呼ばばいいので、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

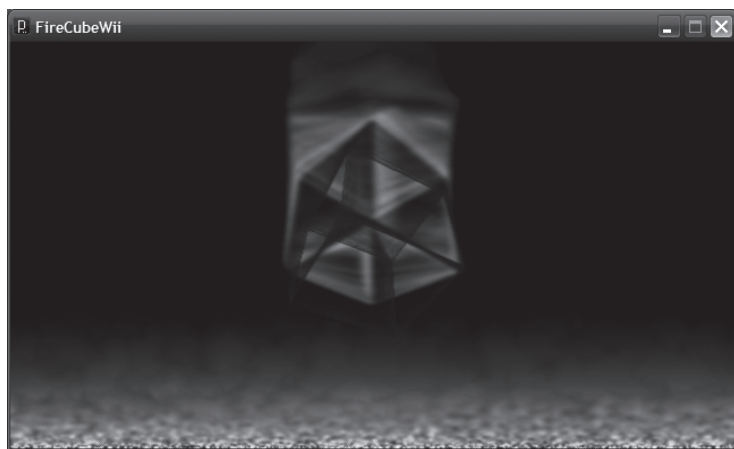
WiiRemote を応用したスケッチを作るにあたって、1 から Processing のスケッチを組んでそれを WiiRemote 対応にすると、Wiimote を使うという点からフォーカスがずれてしまいます。幸い、Processing には多数のサンプルスケッチが添付されているので、これらのうちいくつかを WiiRemote 対応にすることで解説していきます。

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);
```

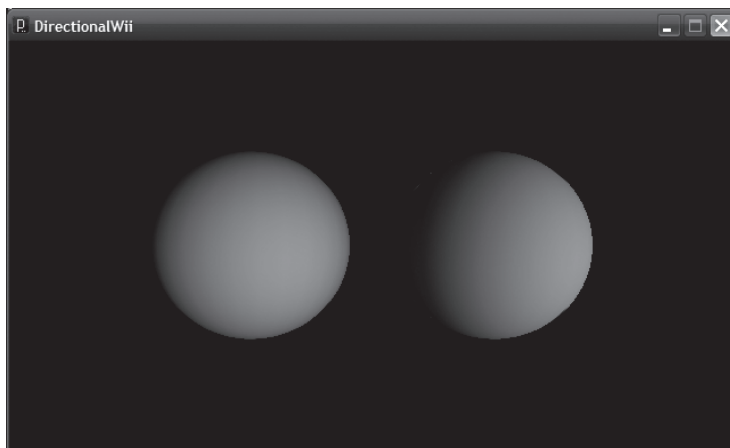
最初に計算しているのは WiiRemote にかかっている加速度です。WiiRemote が動かない状態では、この値がおよそ 1 になります。この状態では炎を出したくないので、次の行で 1 を引くとともに、constrain という関数を使って値を 0.1 から 1 の間に収めています。

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。

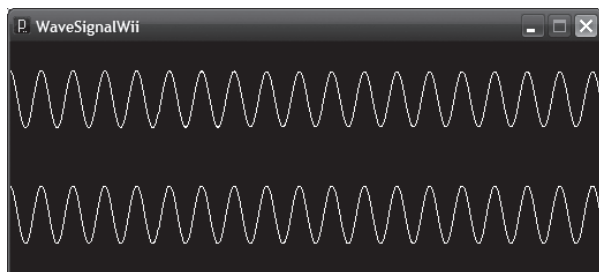
コード6-8 **Processing** 光を当てる方向を出す

```
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いませんでしたが、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 につながるための 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ください。