

「Authoring mobile Flash content for multiple screen sizes」の日本語訳

本ドキュメントは、Adobe サイトで公開されている記事「Authoring mobile Flash content for multiple screen sizes」をヒム・カンパニー 永井勝則が自主的に日本語に訳したものです。

<http://www.himco.jp/>

knagai@himco.jp

(2010/10)

本ドキュメントの原文は

http://www.adobe.com/devnet/flash/articles/authoring_for_multiple_screen_sizes.html

です。

複数の画面サイズに適応できるモバイル Flash コンテンツのプログラミング

Christian Cantrell

2010/1/26

求められる予備知識

本記事は ActionScript 3.0 に習熟していることが求められます

ユーザーレベル

すべてのレベル

Flash Platform がさらに多くのデバイスへの進出をつづけると、開発者には複数の画面サイズと解像度を念頭に置いたプログラミングテクニックが求められるようになります。本記事では、そういった Flash 開発者を支援すべく、画面の解像度や画素密度にかかわらずどのデバイスでも適切にレンダリングされるコンテンツをプログラミングするためのテクニックを解説します。

本記事で探ったテクニックは、プログラミングでベクターを作成し、アセットのサイズと位置をアルゴリズムを使って(とはいえ簡単です)ダイナミックに決める、言わば“低レベル”の方法です。アプリケーションではプログラミングから制御するこのレベルがつねに必要なようになりますが、将来的には“高レベル”のもっと簡単な方法も現れるでしょう。

Adobe は現在、以下で述べる事柄を自動的に適用し、異なる画面に適応するアプリケーションの記述を大幅に簡素化する、コードネーム Slider というモバイル Flex フレームワークの作成を急いでいます。とはいえ、Slider が陽の目を見るまでは、またはこのフレームワークモデルに適合しないアプリケーションを作成するときには、本記事で述べるティップスやトリックが、マルチスクリーン開発を進める助けとなるでしょう。

訳者注:

Sliderは翻訳時現在、Heroと名前を変え、パワーアップしたスマートフォンやタブレットに最適化されたフレームワークとして、開発がつづけられています (<http://opensource.adobe.com/wiki/display/flexsdk/Hero>)。

1. 用語

複数の画面サイズに適応する SWF ベースのアプリケーションのプログラミングテクニックに入る前に、関連する用語を見ておきましょう。みなさんはおそらくこれらの一般的な意味合いはよくご存じだと思いますが、実

際に使用するには、しっかりとした理解が必要です。

・画面サイズ:これは単に、画面(スクリーン)の対角線の長さのことで、通常はインチ単位です。画面サイズは解像度とPPIに関係しますが、画面の解像度や1インチ当たりのピクセル数は画面サイズから決めることはできません。

・解像度:画面やディスプレイの解像度は、その縦横の長さに含まれる1画面当たりのピクセルの数を言います。MotorolaのDroidは489 x 654の画面解像度です。わたしが使っている27インチの外部ディスプレイは1920 x 1200という解像度です。縦と横のピクセル数を掛けると、1画面で表示できるピクセルの総数が分かります。

・PPI、DPI、画素密度:PPI(pixels per inch)とDPI(dots per inch)、画素密度(pixel density)はどれも、物理的な間隔の単位当たりの(縦か横の)ピクセル数という同じ概念を指しています。これが複数の画面サイズのプログラミングで重要になる属性だとは想像しづらいかもしれませんが、以下に述べるように、PPIは解像度と同じくらい、場合によってはそれ以上に重要になります。

2. アプリケーションの設計

マルチスクリーンアプリケーションの目的は必ずしも、どのデバイスでも同じように見えるようにすることではなく、アプリケーションがインストールされるどのデバイスにも適応できるようにすることです。言い方を変えると、マルチスクリーンアプリケーションはホストデバイスの解像度やPPIにダイナミックに適応し、大きな画面では多くの情報を表示し小さな画面では要素を削ったり縮小して、ボタンが容易にタップできるよう物理的に十分な大きさを保つようにする必要があります。アプリケーションが異なる画面でも適切に動作するようにするには、適切なタイミングでアプリケーション自体の描画と再描画を行い、適切な制限をつけるといった方法でアプリケーションを設計する必要があります。

ステージのスケールモードとアラインメント

アプリケーションをレイアウトするにはその前に、ステージのスケールモードとアラインメントを設定しておくことが重要です。これはSprite(メインクラスのこと)のコンストラクタで、ステージのサイズ変更のイベント(resize)を登録する前に行っておくべきです。

```
this.stage.scaleMode = StageScaleMode.NO_SCALE;  
this.stage.align = StageAlign.TOP_LEFT;
```

ステージのスケールモードをNO_SCALEに設定するということは、コンテンツの拡大縮小やレイアウトは自動ではなく、みなさん自身の手ですべて処理するということです。この設定によって、アプリケーションの異なる画面サイズへのダイナミックな適応が可能になります。

ステージの align プロパティを TOP_LEFT に設定するというは、レイアウトは座標(0, 0)のステージ左上隅を基準に行う、ということです。

ステージの resize イベントの監視

マルチスクリーンアプリケーションのレンダリングをするための最適な場所は、ステージの resize イベントのハンドラです。ステージは、アプリケーションが初期化されステージのサイズ(アプリケーションが動作する領域)が設定されたときに resize イベントを送出します。純粋な ActionScript アプリケーションでは、次のように、メインの Sprite のコンストラクタでステージの resize イベントを監視するようにします。

```
this.stage.addEventListener(Event.RESIZE, doLayout);
```

ステージに resize イベントに関する登録をすると、doLayout()関数は、たとえば次のようにステージのサイズが変更されたときに呼び出されます。

- ・アプリケーションが初期化されたとき
- ・ウインドウのサイズが変わったとき(デスクトップで動作しているとき)
- ・デバイスの向きが変わったとき

ステージの resize イベントのハンドラでレイアウトを実行することで、アプリケーションはステージにサイズ変更があったとき、その原因に関係なく必ずそれ自体をレイアウトします。

Note: ステージのサイズを求めるには、stage.stageWidth と stage.stageHeight プロパティを使用します。

SWF ファイルの縦横サイズの設定には必ずしも SWF のメタデータを使う必要はありません。実はそのようにすると、アプリケーションの初期化時に resize イベントハンドラの呼び出しが行われなくなる可能性があります。アプリケーションの幅と高さの設定は、次のようにアプリケーション記述ファイルの initialWindow タグ内で行うのが最善の方法です。

```
<initialWindow>  
<width>320</width>  
<height>480</height>  
<!--そのほかの属性の設定 -->  
</initialWindow>
```

3. アセットのサイズの決定

異なる画面サイズと解像度を持つデバイスで実行するように設計するアプリケーションでは、アセットのサイズを通常ダイナミックに決める必要があります。言い方を変えると、あるデバイスではちゃんと表示されうまく動作するボタンも、高解像度のデバイスでは小さすぎて読みづらく、タップが難しいかもしれない、ということです。したがって、開発者はピクセルとインチ両方の観点から考える方法を知っておくことが重要になります。

ピクセル単位で考える

アプリケーションに単色の背景を追加するには、ピクセルの観点から考えるだけです。たとえば画面の大小は問題でなく、ただ背景をつねに画面の縦横サイズにピクセル単位で一致させればよいだけです。次のコードは単色の背景をアプリケーションのサイズに関わらず追加する例です。

```
var bg:Sprite = new Sprite();
bg.x = 0;
bg.y = 0;
bg.graphics.beginFill(0x006E59);
bg.graphics.drawRect(0, 0, this.stage.stageWidth,this.stage.stageHeight);
this.addChild(bg);
```

ステージの `stageWidth` と `stageHeight` プロパティは、このコンテンツのステージの縦横のサイズをピクセル単位で示すので、どのサイズのデバイスで動作するどんなサイズのアプリケーションにも対応する適切な背景が作成できます。

物理的な単位で考える

アセットのピクセル単位でのサイズ決めは、アセットのサイズが背景の場合のように別のものを基準に決められる場合には有効ですが、絶対サイズで決めなければならない場合にはうまくいきません。別の言い方をすると、背景がステージ全体のサイズである限り、背景の大小は問題にはなりません、フォントやボタンなどではもっと精度を上げてサイズを制御する必要があります。このときには物理的な単位、つまり PPI の観点から考えなければなりません。

インチ:アセットの縦横サイズが PPI で決められるようになると、アセットがレンダリングされる画面に関係なく、アセットの正確なサイズが制御できるようになります。たとえば巨大なデスクトップモニターでレンダリングされるときにも、小さなモバイル画面でレンダリングされるときにもつねに 3/4 インチ x 1/4 インチになるボタンを作成するには、その画面の PPI を使用します。

Note:ヒットターゲットは、つねに確実にヒットさせるには、1/4 インチ、つまり 7mm よりも小さくすべきではな

いという調査結果があります。ボタンの使い勝手を複数のデバイスで確実に保つ唯一の方法は、物理的な単位で考えることです。

現在の画面の PPI は、Capabilities.screenDPI プロパティで求められます。アセットは無論インチではなくピクセル単位でサイズを決めるので、PPI をピクセルに変換する必要があります。わたしは次のような簡単なユーティリティ関数を使っています。

```
/**
 * インチからピクセルへの変換
 */
private function inchesToPixels(inches:Number):uint
{
    return Math.round(Capabilities.screenDPI * inches);
}
```

次のコードは、どのデバイスでも 3/4 インチ x 1/4 に見えるスプライトを作成する例です。

```
var button:Sprite = new Sprite();
button.x = 20;
button.y = 20;
button.graphics.beginFill(0x 003037);
button.graphics.drawRect(0, 0, this.inchesToPixels(.75),this.inchesToPixels(.25));
button.graphics.endFill();
this.addChild(button);
```

メートル: アメリカ中心になりすぎないように、また小さな拡大縮小ではメートルシステムの方がすぐれているので、ミリメートルからピクセルに変換するコードも紹介しておきます。

```
/**
 * ミリメートルからピクセルへの変換
 */
private function mmToPixels(mm:Number):uint
{
    return Math.round(Capabilities.screenDPI * (mm / 25.4));
}
```

4. ダイナミックなレイアウト

ここまでで、アプリケーションをプログラミングによって複数の画面サイズに適応できる方法で設計できるようになり、アセットのサイズを決定するテクニックも覚えたところで、次はアセットのレイアウトに進みましょう。

異なる画面サイズに適応できるようにアセットをレイアウトする鍵は、何をハードコーディング(決め打ち)し、何を現在の画面のプロパティにもとづいて計算するかを知ることにあります。たとえば、アプリケーションの最上部にタイトルバーを作成したいときには、その x と y 座標を(0, 0)にすればよいことが分かります。これは、これらのプロパティでは値がハードコーディングできるということです。言い方を変えると、タイトルバーは、アプリケーションが動作するデバイスに関係なく、つねに左上隅に配置すればよいということです。次のコードはタイトルバー用の Sprite を作成し、その位置をハードコーディングする例です。

```
var titleBar:Sprite = new Sprite();  
titleBar.x = 0;  
titleBar.y = 0;
```

タイトルバーの位置はどのデバイスでも変わりませんが、タイトルバーの幅は変わります。高解像度のデバイスでは、幅は大きくする必要があります。

タイトルバーの幅は `stage.stageWidth` プロパティで簡単に決められますが、高さはどうでしょう？ 高さもその気になればピクセル単位でハードコーディングできますが、そのサイズはデバイスの解像度によって大幅に変わってしまいます。この場合には物理的な単位の観点から考える方が適切で、すべてのデバイスでタイトルバーに変わらない外見を持たせることができます。

次のタイトルバーを作成するコードは、以下の概念を例示するためのコードです。

- ・適切な場合には、絶対位置をハードコーディングする
- ・タイトルバーの幅は、`stage.stageWidth` を使ってピクセル単位でダイナミックに決める
- ・タイトルバーの幅は、すべてのデバイスで同じように見えるように、インチ単位で設定する

```
var titleBar:Sprite = new Sprite();  
titleBar.x = 0;  
titleBar.y = 0;  
titleBar.graphics.beginFill(0x003037);  
titleBar.graphics.drawRect(0, 0, this.stage.stageWidth,this.inchesToPixels(.3));
```

```
titleBar.graphics.endFill();
this.addChild(titleBar);
```

上の例はアセットのサイズを動的に設定する方法を示していますが、位置を動的に決めるにはどうすればよいのでしょうか？たとえばタイトルバーの位置はつねに左上隅にあるので簡単に決まりますが、x座標は0でもy座標がステージの高さで決まるフッターの配置はどうすればよいのでしょうか？

次のコードは、幅がアプリケーションの幅と同じで、画面の高さに関係なく必ず最下部に配置するフッターの作成方法の例です。

```
var footer:Sprite = new Sprite();
footer.graphics.beginFill(0x003037);
footer.graphics.drawRect(0, 0, this.stage.stageWidth, this.inchesToPixels(.3));
footer.graphics.endFill();
footer.x = 0;
footer.y = this.stage.stageHeight - footer.height;
this.addChild(footer);
```

5. 相対的な位置決め

アセットのレイアウトには主に次の3つの方法があります(初めの2つはすでに述べた方法です)。

1. アプリケーションの背景やタイトルバーで行ったように、位置をハードコーディングする方法
2. アプリケーションのフッターで行ったように、位置を計算で決める方法
3. ほかのアセットの位置にもとづいて計算して決める方法

アセットの位置を別のアセットの位置にもとづいて計算することを相対配置と言い、マルチスクリーンアプリケーションでは極めて重要になるテクニックです。前のタイトルバーの例では、タイトルバーを希望する位置につねに配置しレンダリングできましたが、タイトルの文字そのものについてはどうでしょう？

y位置はつねにハードコーディングできます。位置は上端から2、3ピクセル下になるでしょう。x座標はステージの幅とタイトル文字の幅にもとづいて計算することができます。しかしこの方法では次の2つの理由から、最良の結果をもたらすとは限らないのです。

1. タイトルバーの高さは画面のPPIによって変わるので、タイトルの文字は垂直方向センターに必ず来るとは限らない。
2. 何らかの理由で(たとえばフッターをなくして、タイトルバーを下端に置きたい)タイトルバーの

位置を変える場合には、タイトルの文字のコードも変更する必要が出てくる。

この問題は両方とも、タイトルの位置をタイトルバーを基準にして決めることで解決できます。これはわたしもよく起こしてしまう問題なので、わたしはそのための簡単なユーティリティ関数を用意しています。

```
/**
 * 表示オブジェクトを別の表示オブジェクトを基準にその真ん中に置く
 */
private function center(foreground:DisplayObject, background:DisplayObject):void
{
    foreground.x = (background.width / 2) - (foreground.width / 2);
    foreground.y = (background.height / 2) + (foreground.height / 2);
}
```

この center()関数を使うと、タイトルの配置は簡単に行えます。

```
var title:SimpleLabel = new SimpleLabel(
    "My Application", "bold", 0xffffff, "_sans", this.inchesToPixels(.15));
this.center(title, titleBar);
this.addChild(title);
```

6. 適応性のあるコンテンツ

タイトルバーのようなダイナミックなサイズ決めとレイアウトも大切ですが、実際のアプリケーションコンテンツはもっと大変です。たとえばメインコンテンツが正方形のグリッドで構成されるゲームを考えてみてください。そのゲームを複数のデバイスでプレイ可能にする最良のテクニックは何だと思いませんか？ 正方形は単に画面サイズにもとづいて拡大または縮小すればよいのでしょうか？ それとも行または列を追加、または削除すればよいのでしょうか？

いずれも妥当な方法で、ゲーム次第です。たとえばチェスやチェッカーゲームの場合には、行数と列数が決まっているので、画面サイズに関わらず行も列も変更できません。この場合には通常は、見た目を保つためにコンテンツをただ拡大または縮小する方法が最良でしょう。

ゲームの中には、ゲームプレイを画面サイズに適応できるものがあります。たとえばリアルタイムの戦略ゲームは、高解像度の方が表示できるタイルが増えるので、もっと大きな画面に拡張できるかもしれません。また小さな画面でこのゲームをプレイするときには、タイルが少しでも大きく詳細にレンダリングされるように、タイ

ルを減らす方法が最良かもしれません。この場合にはコンテンツを状況に合わせて変える必要があります。コンテンツは多種多様なので、たった1つでさまざまな画面サイズに適応できる便利な戦略はなく、公式もありませんが、標準的なテクニックはあります。以下はゲーム盤をさまざまな画面サイズに適応する論理です。

1. タイルやブロックのサイズを決めます。タイルにビットマップが含まれるときには、そのサイズはビットマップの縦横のピクセル値によってもう決まっています(ビットマップはベクターのように伸縮しないので)。タイルがベクターのときには、タイルの絶対サイズを基本に決めることになるかもしれません。
2. タイルのレイアウトに必要なスペースを決めます。このスペースは画面の全体からタイトルバーやナビゲーションなどを差し引いた量になるはずです。
3. そのスペースに収まる行数と列数を計算します。タイル間に空きを作りたいときにはこれも考慮に入れます。
4. 使用可能なスペースでタイルをレイアウトします。これは簡単そうに思えますが少し工夫がいります。なぜならゲームボードの上下にはまず間違いなく、水平と垂直方向の同じだけの余白を残したくなるからです。

下のコードは、ゲームボードの上下に余白を同じだけ残しつつ、割り当て可能なスペースにできるだけ多くの1/4インチのブロックをレイアウトする論理の例です。

```
// 収納可能な画面スペースにできるだけ多くのブロックを表示する
var BLOCK_SIZE:Number = .25;
var BLOCK_BUFFER:uint = 3;
var blockSize:uint = this.inchesToPixels(BLOCK_SIZE);
var blockTotal:uint = blockSize + BLOCK_BUFFER;
var cols:uint = Math.floor(this.stage.stageWidth / blockTotal);
var rows:uint = Math.floor((this.stage.stageHeight - titleBar.height) / blockTotal);
var blockXStart:uint =
    (this.stage.stageWidth - ((cols * blockSize) + ((cols - 1) * BLOCK_BUFFER))) / 2;
var blockX:uint = blockXStart;
var blockY:uint = ((this.stage.stageHeight + titleBar.height)
    - ((rows * blockSize) + ((rows - 1) * BLOCK_BUFFER))) / 2;
for (var colIndex:uint = 0; colIndex < rows; ++colIndex)
{
    for (var rowIndex:uint = 0; rowIndex < cols; ++rowIndex)
    {
```

```

        // private 関数を使ってブロックを描画
        var block:Sprite = this.getBlock(blockSize);
        block.x = blockX;
        block.y = blockY;
        this.addChild(block);
        blockX += blockTotal;
    }
    blockY += blockTotal;
    blockX = blockXStart;
}

```

次のコードは各ブロックを作成する関数のコードです。

```

/**
 * ゲームボードに追加する新しいブロックを作成する
 */
private function getBlock(blockSize:uint):Sprite
{
    var block:Sprite = new Sprite();
    block.graphics.beginFill(0xAAC228);
    block.graphics.drawRect(0, 0, blockSize, blockSize);
    block.graphics.endFill();
    block.cacheAsBitmap = true;
    return block;
}

```

Note: ブロックを作成するときには、cacheAsBitmap プロパティを true に設定します。モバイルアプリケーションの最適化に関するトピックは本記事の範囲を超えますが、頻繁に拡大縮小したり回転させる予定のない表示オブジェクトは、つねに cacheAsBitmap プロパティを true に設定するのがベストです。これによってデスクトップではパフォーマンスが向上しますが、デスクトップよりプロセッサ能力の劣るデバイスでは劇的な結果を生み出します。

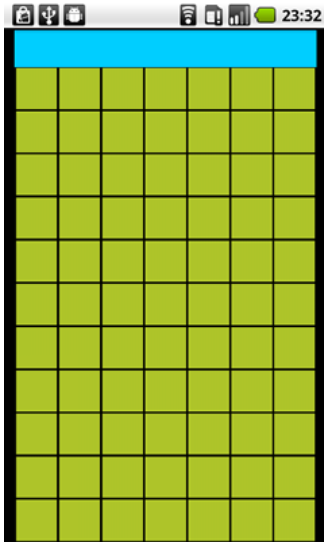


図3: タイルコードの実行結果。上の青い部分がタイトルバー。

7. フォントの管理

フォントはほとんどすべてのアプリケーションで鍵となる重要な要素で、複数の画面に対応するアプリケーションを設計するときには、ほかのアセットと同様の注意を払う必要があります。以下に複数のデバイスにうまく適応できるフォントの使用法のティップスを3つ挙げます。

- ・FTE(Flash Text Engine)を使う: わたしは FTE の使用を強くすすめます。これは Flash Player10 と Adobe AIR1.5、Flash Lite4 で導入された新しい Flash のテキストエンジンです。これは TextField オブジェクトよりもレンダリング性能がすぐれているだけでなく、テキストを高い精度で配置できる機能を備えています。ascent や descent、textWidth、textHeight といった TextLine のプロパティを使用すると、ピクセル精度の正確性でテキストの位置を決めることができます。

- ・コンポーネントの作成も視野に入れる: タイトルの例では、わたしが自作した SimpleLabel という名前の簡単なコンポーネントを使っています。これはカプセル化した FTE が容易に使用できる、テキスト作成のためのコンポーネントです。

- ・必要なら幅と高さの getter をオーバーライドする: わたしの SimpleText コンポーネントでは、DisplayObject の width と height プロパティをオーバーライドして、前の center()関数のようなユーティリティとテキストフィールドとの併用がうまく動作するようにしています。以下はその width と height の getter です。これによって良好な結果を得ることができます。

```
public override function get width():Number
```

```
{
    return this.textLine.textWidth;
}

public override function get height():Number
{
    return (this.textLine.ascent - 1);
}
```