Autodesk®
Scaleform®

Scale9Grid User Guide

本書では、Scaleform で Scale9Grid の機能を使用したサイズ変更可能なウィンドウ、パネル、ボタンの作成方法を説明しています。

著者: Maxim Shemanarev、Michael Antonov
バージョン: 1.01
最終更新日: 2008 年 3 月 21 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Disclaimer

Autodesk Scaleform の連絡先:

ドキュメント	Scale9Grid ユーザー ガイド
住所	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
ホームページ	www.scaleform.com
電子メール	mailto:info@scaleform.com
電話	(301) 446-3200
Fax	(301) 446-3199

目次

1. はじめに:Flash と Scaleform で Scale9Grid を使用する

Scale9Grid は Flash®で、ムービー クリップ用のコンポーネント スタイル スケーリングを指定するために使われます。Scale9Grid を使用すれば、開発者はよく使われるようなグラフィックやデザインエレメントを用いた直線的スケーリングではなく、インテリジェントにサイズ変更出来るムービー クリップ シンボルやユーザー インターフェイスのコンポーネントを作成できます。つまり、与えられた画面のエリアを有効に使うことができるのです。

Flash 用語では、Scale9Grid は「9 スライス スケーリング」とも呼ばれます。9 スライス スケーリングを有効にすると、ムービー クリップは理論上、グリッドのようなオーバーレイが付いた 9 個のセクションに分割されます。9 個の領域はそれぞれ独立してサイズ変更することができます。ムービー クリップの視覚的な整合性を損なわないように、コーナーはサイズ変更せず、コーナー以外の領域をサイズ変更します（引き伸ばしの逆）。

Flash Studio では、9 スライス スケーリングはムービー クリップの [シンボル プロパティ] ダイアログで、チェックボックスをオンにすると使用できます。また、MovieClip.scale9Grid と Button.scale9Grid のプロパティは ActionScript で確認できるので、ピースごとのスケーリングをプログラム的にコントロールできます。Flash における Scale9Grid の使用については、Flash Studio のマニュアルを参照してください。

Adobe Flash Player では Scale9Grid 機能に制限があり、実用的な場面では使いにくくなっています。たとえば、標準の Flash Player はイメージ スライスをサポートしていないため、自由回転など任意の変形を含む Scale9Grid を処理することができません。Scaleform は Scale9Grid をもれなくサポートする一方で、Flash との互換性も保っています。Adobe Flash と Scaleform における Scale9Grid 機能について、以下の表に分類します:

Scale9Grid 機能	Adobe Flash Player	Scaleform
変形	X/Y 方向の拡大/縮小のみ	任意の変形
ビットマップ	境界ボックスをベースにした均等スケーリング	Scale9Grid による自動タイリング
グラデーションとイメージ塗りつぶし	境界ボックスをベースにした均等スケーリング	境界ボックスをベースにした均等スケーリング
サブシンボル (ネスト化されたムービー クリップ、ボタン、グラフィック)	サポートされていない	サポート
ネスト化された Scale9Grids	サポートされていない	サポートされていない
内包するシンボルの変形	サポート	サポート
テキスト	通常通りのサイズ変更	通常通りのサイズ変更

Scaleform は Flash との下位互換性を備えているため、Flash で動作する Scale9Grid 機能は Scaleform でも同様に動作することになります。ただし、Flash でサポートされていない機能は思ったように動作しないかもしれませんが、より実用的になっています。たとえば、サブシンボル（ネスト化されたムービー クリップ）のサポートは、インタラクティブにサイズ変更可能なウィンドウを作成する際に不可欠です。2 つの Player の相違点については、本書で詳しく説明します。

開発者に Scale9Grid の実践例を示すため、本書では数多くのサンプル FLA/SWF ファイルを参照しています。

2. 変形

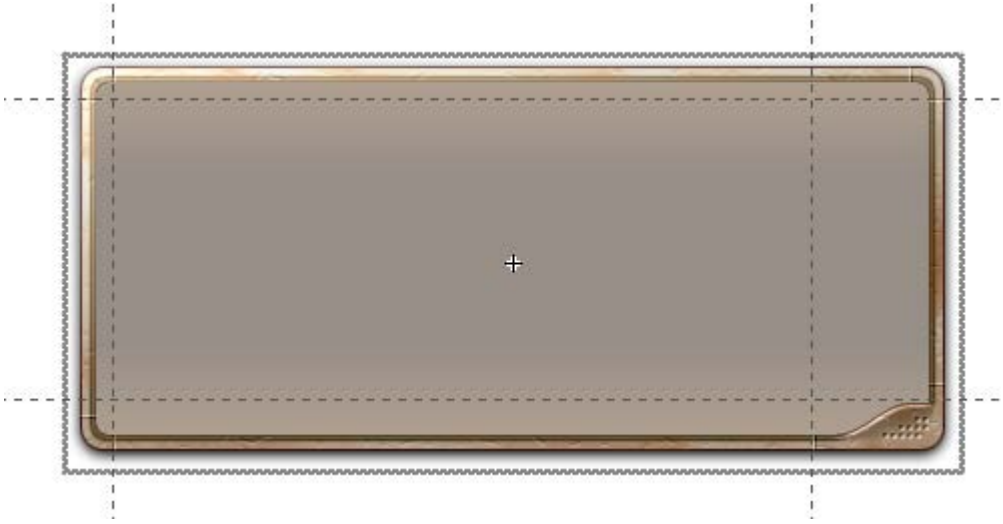
Flash とは異なり、Scaleform は Scale9Grid を適用したムービー クリップの任意の変形をサポートします。Flash では、参照したムービー クリップを回転や歪曲することはできません。ただし、スプライトの形でムービー クリップを含め、内包するスプライトに回転、または歪曲を適用することは可能です。内包するスプライトの変形は Flash でも Scaleform でもまったく同様に動作します。高い階層にあるシンボルの変形は、以下の例に示すように、同じ Scale9Grid を適用しても異なる外観を描画することができます。

3. ビットマップの処理

グラデーションとビットマップ図形による塗りつぶしは、Flash でも Scaleform でも同じように変形されますが、単一ビットマップの処理には違いがあります。Flash では、単一ビットマップを 9 スライス スケールのムービー クリップに配置すると、ユーザーが手動でビットマップを 9 つのピースに切り分けでもしないと、プレイヤーでは Scale9Grid の設定は無視されます。一方で、Scaleform Player は自動的にビットマップをスライスするので、ビットマップは Scale9Grid を使って正確にスケーリングされます。この操作により開発作業が大幅に簡素化され、アーティストが手動でイメージ カッティングをしなくても、サイズ変更可能なボタンやウィンドウの作成が可能になります。また、自動スライスにより EdgeAA でレンダリングしたコンテンツをつなぎ合わせた時に発生しやすい、アンチエイリアスのシームを除去します。自動スライスの動作の詳細については、以下の例で説明します。

アーティストが、以下のイメージのようなウィンドウ背景を表すビットマップを作成すると仮定します:

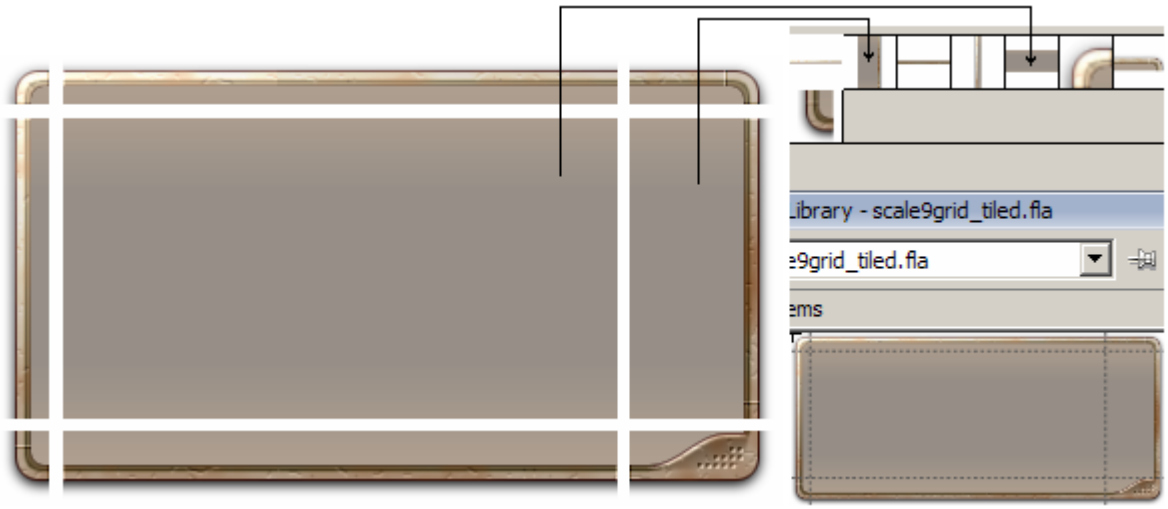
スケーリングしてもコーナーの形が変わらないように、Scale9Grid を適用します。この操作を設定するには、ムービー クリップを作成して、以下の手順で Scale9Grid を適用するのが最も分かりやすいでしょう:



ただし、この操作は Flash では思った通りに動作しません。作成したムービー クリップをスケーリングすると、Scale9Grid を適用していないような仕上がりになります:

Scaleform の場合は、期待した通りに動作し、コーナーの形はそのまま変わりません:

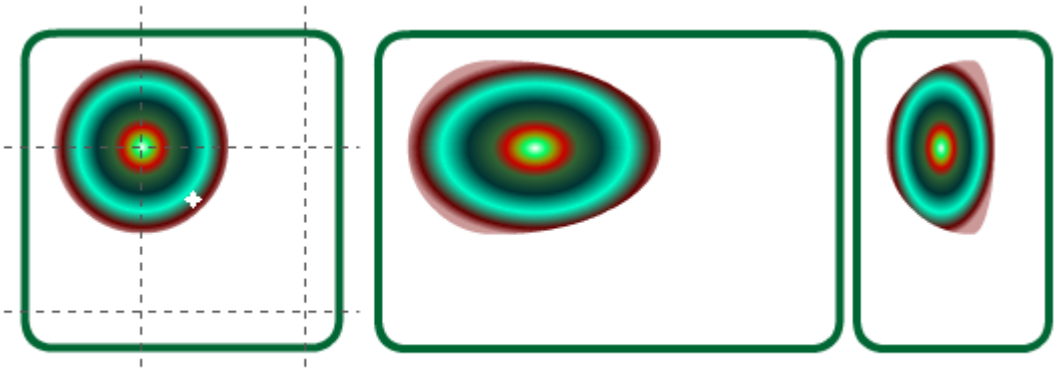
Adobe Flash は、境界ボックスを基準にイメージの平均スケールを計算します。つまり、9 スライス スケーリングでビットマップを操作することは可能ですが、Scale9Grid タイルと正確に一致する 9 つの独立したイメージ ピースが必要になるということです。



Scaleform の場合は、Flash ファイルの再生時にこの操作を自動的に実行します。

ただし、任意でイメージを回転、または歪曲している場合は、Scaleform は Flash と同様の処理を行います。上記の「手動でタイリングした」ウィンドウの例は、scale9grid_tiled.fla と scale9grid_tiled.swf を参照してください。

ビットマップとグラデーションの塗りつぶしスタイルを使った図形の場合は、Flash と Scaleform の表示はまったく同じです。グラデーションとビットマップの塗りつぶしスタイルはいずれも、Scale9Grid 変形を維持できないので、注意してください。変形はベクトル図形の輪郭にのみ適用され、塗りつぶしスタイルのコンテンツには適用されません。これが例です:

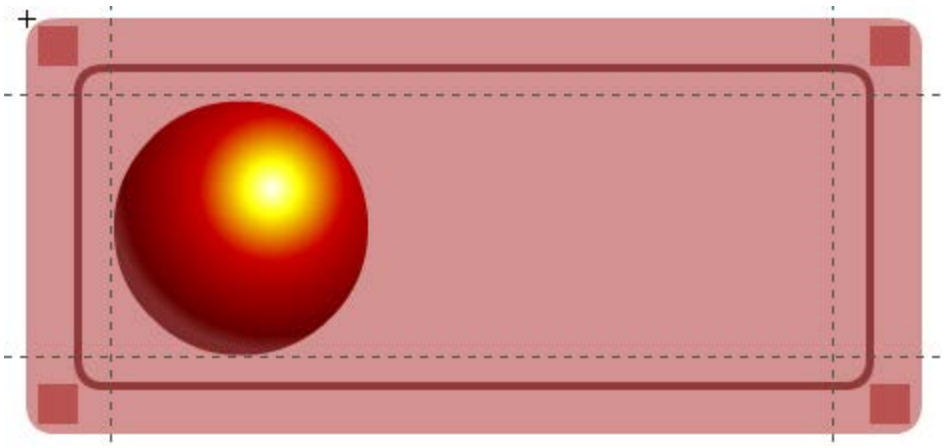


4. インタラクティブ ウィンドウ

この章では、インタラクティブにサイズ変更可能なウィンドウの例を提示し、ユーザーの入力によるサイズ変更を実行するのに必要な ActionScript ソースコードについて説明します。

4.1 Flash 互換の例

以下の例は、インタラクティブにサイズ変更可能なウィンドウの、Flash 互換バージョンの作成手順を示しています。このサンプルは、ファイル scale9grid_window1.fla と scale9grid_window1.swf を使用しています。Flash Studio で以下のようなムービー クリップを作成したと仮定します:



コーナーにある小さな正方形をドラッグするとウィンドウのサイズが変更でき、コーナー以外の面をドラッグするとウィンドウを移動できます。ここで問題になるのは、Flash と ActionScript には、ムービー クリップ内の様々な図形やレイヤーを区別するヒット テスト用のメカニズムがないことです。さらに、Flash では 9 スライス スケーリングを、ネスト化されたムービー クリップに適用することはできません。こうした矛盾があることから、サイズ変更ロジックのプログラムは非常に難しくなります。図形のヒット テストの代わりに、プログラマは座標をきちんとチェックする必要があります。以下は、対象となるムービー クリップに関連する ActionScript プログラムの例です:

```
import flash.geom.Rectangle;

var bounds:Object = this.getBounds(this);
for (var i in bounds) trace(i+" --> "+bounds[i]);
this.XMin = bounds.xMin;
this.YMin = bounds.yMin;
this.XMax = bounds.xMax;
this.YMax = bounds.yMax;
this.MinW = 120;
this.MinH = 100;
this.OldX = this._x;
```

```
this.OldY = this._y;
this.OldW = this._width;
this.OldH = this._height;
this.OldMouseX = 0;
this.OldMouseY = 0;
this.XMode = 0;
this.YMode = 0;


this.onPress = function()
{
	this.OldX = this._x;
	this.OldY = this._y;
	this.OldW = this._width;
	this.OldH = this._height;
	this.OldMouseX = _root._xmouse;
	this.OldMouseY = _root._ymouse;
	this.XMode = 0;
	this.YMode = 0;

	var kx = (this.XMax - this.XMin) / this._width;
	var ky = (this.YMax - this.YMin) / this._height;
	var x1 = this.XMin + 6  * kx;   // Left
	var y1 = this.YMin + 4  * ky;   // Top
	var x2 = this.XMax - 26 * kx;   // Right
	var y2 = this.YMax - 25 * ky;   // Bottom
	var w  = 20 * kx;
	var h  = 20 * ky;
	var xms = this._xmouse;
	var yms = this._ymouse;

	if (xms >= x1 && xms <= x1+w)
	{
		if (yms >= y1 && yms <= y1+h)
		{
			this.XMode = -1;
			this.YMode = -1;
		}
		else
		if (yms >= y2 && yms <= y2+h)
		{
			this.XMode = -1;
			this.YMode =  1;
		}
	}
	else
	if (xms >= x2 && xms <= x2+w)
```

```
	{
		if (yms >= y1 && yms <= y1+h)
		{
			this.XMode =  1;
			this.YMode = -1;
		}
		else
		if (yms >= y2 && yms <= y2+h)
		{
			this.XMode =  1;
			this.YMode =  1;
		}
	}

	if (XMode == 0 && YMode == 0)
	{
		this.startDrag();
	}
}


this.onRelease = function()
{
	this.XMode = 0;
	this.YMode = 0;
	this.stopDrag();
}

this.onReleaseOutside = function()
{
	this.XMode = 0;
	this.YMode = 0;
	this.stopDrag();
}

this.onMouseMove = function()
{
	var dx = _root._xmouse - OldMouseX;
	var dy = _root._ymouse - OldMouseY;

	if (this.XMode == -1)
	{
		this._x     = this.OldX + dx;
		this._width = this.OldW - dx;
		if (this._width < this.MinW || _root._xmouse > this.OldX + this.OldW)
		{
			this._x     = this.OldX + this.OldW - this.MinW;
```

```
                this._width = this.MinW;
            }
        }
        if (this.XMode == 1)
        {
            this._width = this.OldW + dx;
            if (this._width < this.MinW || _root._xmouse < this.OldX)
                this._width = this.MinW;
        }
        if (this.YMode == -1)
        {
            this._y      = this.OldY + dy;
            this._height = this.OldH - dy;
            if (this._height < this.MinH || _root._ymouse > this.OldY + this.OldH)
            {
                this._y      = this.OldY + this.OldH - this.MinH;
                this._height = this.MinH;
            }
        }
        if (this.YMode == 1)
        {
            this._height = this.OldH + dy;
            if (this._height < this.MinH || _root._ymouse < this.OldY)
                this._height = this.MinH;
        }
}
```

このように、ロジックはかなり複雑です。ムービー クリップの最初の境界ボックスとその他の変数が必要です。動作をコントロールする重要な数値は XMode と YMode です。
モード値 '-1' はそれぞれ、ウィンドウの境界線の左、または上部をドラッグすることを意味します。数値 '1' は右、または下部をドラッグすることを表しています。

メインのヒット テストのロジックは onPress()関数に含まれています。マウスの座標軸は自動的に Scale9Grid 変形ロジックを維持することができないため、まず、スケーリングの係数 kx と ky を計算する必要があります。

```
        var kx = (this.XMax - this.XMin) / this._width;
        var ky = (this.YMax - this.YMin) / this._height;
```

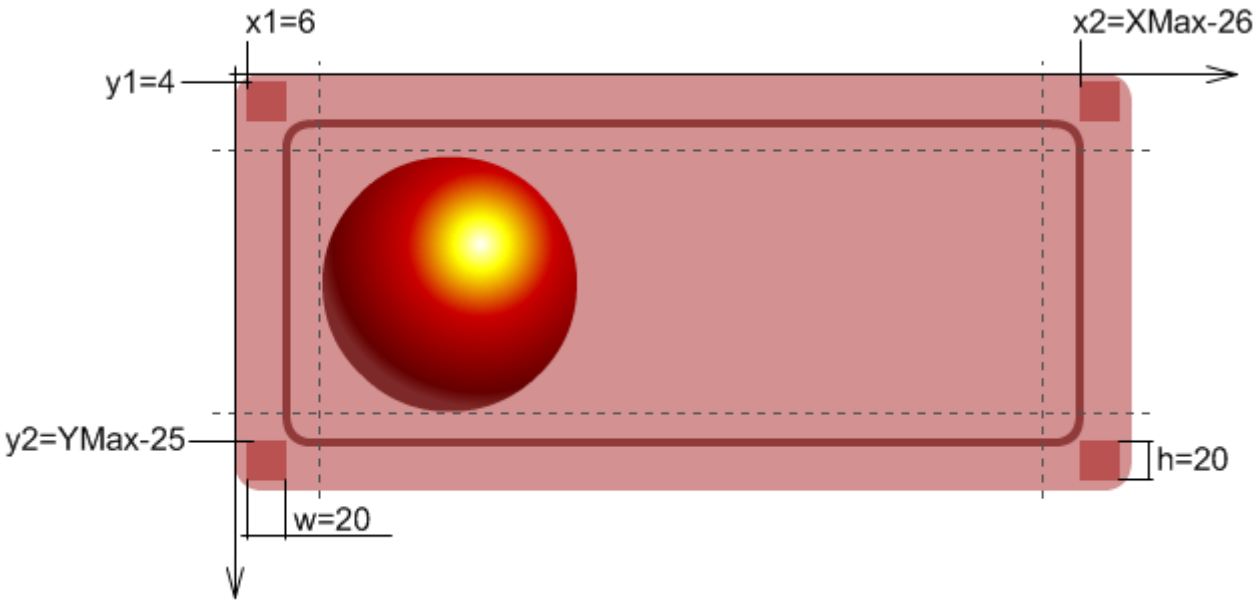
次に、ヒット テストの矩形（この場合は正方形）を計算します:

```
        左-上:        x1, y1, x1+w, y1+h
        右-上:        x2, y1, x2+w, y1+h
        左-下:        x1, y2, x1+w, y2+h
        右-下:        x2, y2, x2+w, y2+h

        var x1 = this.XMin + 6  * kx;   // Left
```

```
var y1 = this.YMin + 4  * ky;   // Top
var x2 = this.XMax - 26 * kx;   // Right
var y2 = this.YMax - 25 * ky;   // Bottom
var w  = 20 * kx;
var h  = 20 * ky;
```

定数 6、4、26、25、および 20 に注意してください。実際、これらの座標はムービー クリップの中の図形に正確に一致する必要があります。コーナーの正方形は必ずしも必要ではなく、削除してもかまいません。形を変更する場合、それに合わせて ActionScript コードも修正しなければならないのが、この方法の最大の欠点です。以下の図で、使用される定数の意味を説明します:



次の手順で、プログラム的に座標を検証し、それぞれのサイズ変更モードを割り当てます。

```
var xms = this._xmouse;
var yms = this._ymouse;
if (xms >= x1 && xms <= x1+w)
{
      ... and so on
```

言うまでもなく、コーナーの図形が複雑になれば、それに応じて複雑なロジックが必要になります。たとえば、以下のようなコーナーの場合、2 つの矩形をそれぞれチェックする必要があります:

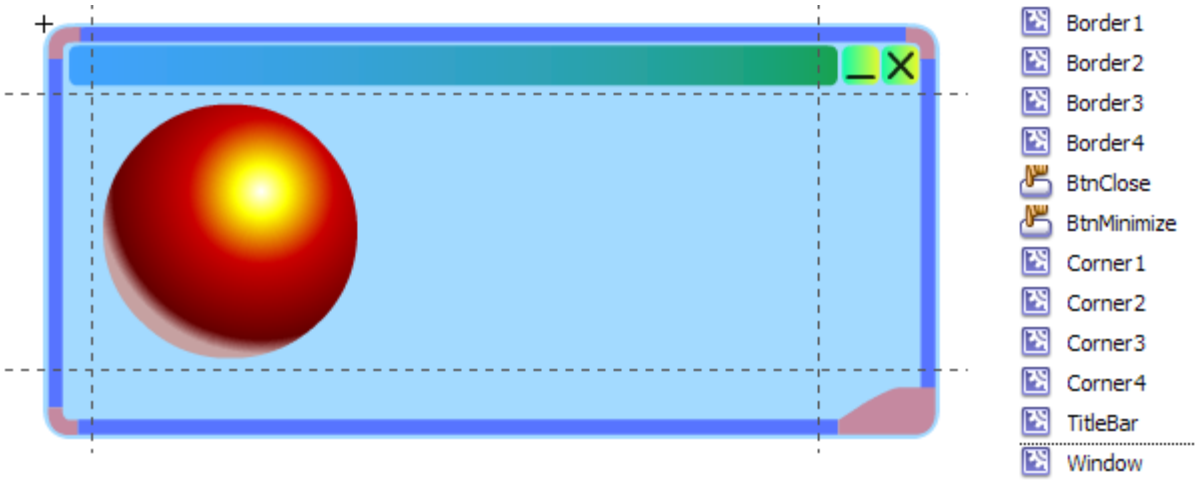
丸みのあるコーナーや不規則な形はさらに複雑になります。ここに示す例では、サイズ変更の境界線については扱いません。

これまで述べた以外の ActionScript ロジックは、最小幅や高さの制限を守りながら、ウィンドウの座標やサイズの変更に対応します。

イメージ背景のあるウィンドウの場合、Flash との完全な互換性を保つには手動のタイリングが必要になりますので注意してください。

4.2 高度な Scaleform の例

Scaleform では正しく動作し、Flash では動作しないさらに高度なサンプルを以下に示します。この違いは、Scaleform がネスト化されたムービー クリップを正しく変形できることから生じるものです。つまり、任意の不規則な図形に、マウスのヒット テスト機能を使用できることを意味します。このサンプルは scale9grid_window2.fla ファイルと scale9grid_window2.swf ファイルに含まれています。このサンプルのグラフィックはさらに複雑ですが、ハードコード化された座標を使用せず、簡単に再利用できる一般化された ActionScript コードをベースにした、高度な機能を提供します。



このサンプル Flash ファイルには、Border1 から Border4（青）、Corner1 から Corner4（ピンク）、TitleBar などの各種ムービー クリップとボタンが含まれています。Scaleform は Scale9Grid を基準にネスト化されたムービー クリップを正しく調整するので、そのようなムービー クリップに関数を割

り当てるだけで必要な作業が完了します。この ActionScript はシンプルでわかりやすいものになります。OnResize()関数は前のサンプルと同じなので注意してください。

```
import flash.geom.Rectangle;
//trace(this.scale9Grid);
this.MinW = 100;
this.MinH = 100;
this.XMode = 0;
this.YMode = 0;
this.OldX = this._x;
this.OldY = this._y;
this.OldW = this._width;
this.OldH = this._height;
this.OldMouseX = 0;
this.OldMouseY = 0;

function AssignResizeFunction(mc:MovieClip, xMode:Number, yMode:Number)
{
      mc.onPress          = function() { this._parent.StartResize(xMode, yMode);
      mc.onRelease        = function() { this._parent.StopResize(); }
      mc.onReleaseOutside = function() { this._parent.StopResize();    }
      mc.onMouseMove      = function() { this._parent.OnResize(); }
}

AssignResizeFunction(this.Border1,  0, -1);
AssignResizeFunction(this.Border2,  1,  0);
AssignResizeFunction(this.Border3,  0,  1);
AssignResizeFunction(this.Border4, -1,  0);
AssignResizeFunction(this.Corner1, -1, -1);
AssignResizeFunction(this.Corner2,  1, -1);
AssignResizeFunction(this.Corner3,  1,  1);
AssignResizeFunction(this.Corner4, -1,  1);
this.TitleBar.onPress          = function() { this._parent.startDrag(); }
this.TitleBar.onRelease        = function() { this._parent.stopDrag(); }
this.TitleBar.onReleaseOutside = function() { this._parent.stopDrag(); }

function StartResize(xMode:Number, yMode:Number)
{
      this.XMode = xMode;
      this.YMode = yMode;
      this.OldX = this._x;
      this.OldY = this._y;
      this.OldW = this._width;
      this.OldH = this._height;
      this.OldMouseX = _root._xmouse;
      this.OldMouseY = _root._ymouse;
}
```

```
function StopResize()
{
      this.XMode = 0;
      this.YMode = 0;
}

function OnResize()
{
      var dx = _root._xmouse - OldMouseX;
      var dy = _root._ymouse - OldMouseY;

      if (this.XMode == -1)
      {
            this._x     = this.OldX + dx;
            this._width = this.OldW - dx;
            if (this._width < this.MinW || _root._xmouse > this.OldX + this.OldW)
            {
                  this._x     = this.OldX + this.OldW - this.MinW;
                  this._width = this.MinW;
            }
      }
      if (this.XMode == 1)
      {
            this._width = this.OldW + dx;
            if (this._width < this.MinW || _root._xmouse < this.OldX)
                this._width = this.MinW;
      }
      if (this.YMode == -1)
      {
            this._y      = this.OldY + dy;
            this._height = this.OldH - dy;
            if (this._height < this.MinH || _root._ymouse > this.OldY + this.OldH)
            {
                  this._y      = this.OldY + this.OldH - this.MinH;
                  this._height = this.MinH;
            }
      }
      if (this.YMode == 1)
      {
            this._height = this.OldH + dy;
            if (this._height < this.MinH || _root._ymouse < this.OldY)
                this._height = this.MinH;
      }
}
```

この方法には 2 つの欠点があります。まず、Adobe Flash Player との互換性が失われることです。2 つ目の問題は、多くのムービー クリップを含むため、生成される描画プリミティブの数が増え、余分

なメモリを消費するということです。これは、シンプルでわかりやすいソリューションの代償と言えるかもしれません。描画プリミティブが多すぎて問題になる場合は、2 つの方法を組み合わせて、境界線とコーナーを、ウィンドウのフレームを示す 1 つの図形に置き換えることも可能です。この場合、ActionScript コードにロジックを追加する必要がありますが、最初のサンプルと比較すると複雑さと「脆弱さ」が解消されています。この方法の最大の利点は、境界線とコーナーが不規則なムービー クリップのヒット テストに、任意の形状を使用できるということです。図形とモーション トゥィーンも使用できます。

Scale9Grid で SWF ファイルを生成する場合、Flash Studio には潜在的バグがあることに留意しておいた方が良いでしょう。このバグは scale9grid_window2.fla で再現できます。[TitleBar] レイヤーを選択し、ウィンドウの中央に小さな矩形を描画すると、Scale9Grid に誤動作が生じます（正しくなくなります）。これは Flash Studio が、ネスト化されたムービー クリップと、その他のグラフィックを同じレイヤーに入れることを許可しないことによるものです。この矩形を削除した後も、正しい Scale9Grid が復元できないというケースも発生しています。この場合は「保存して圧縮」操作で問題を解決できます。

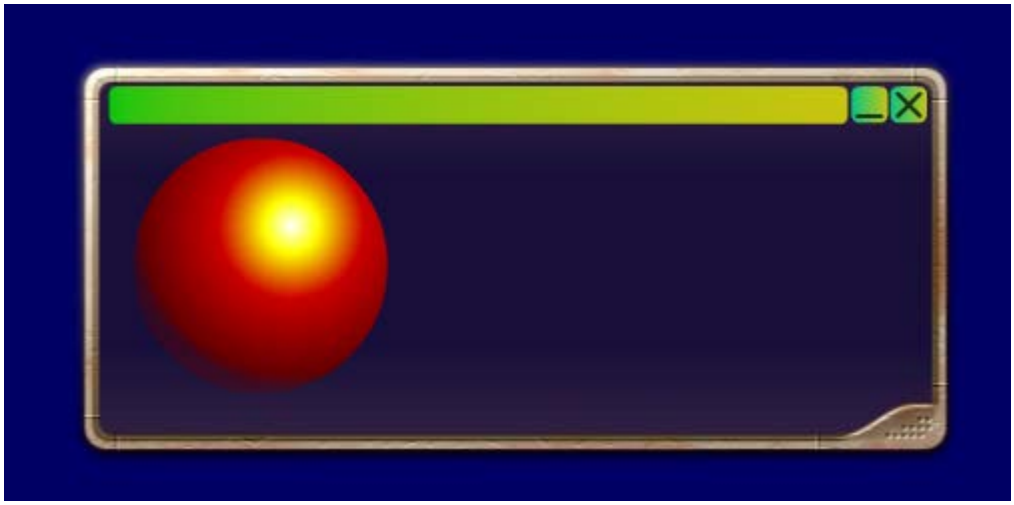
それでも問題が続く場合は、ActionScript から Scale9Grid を復元します。Scale9Grid が正しく動作している間に、トレースするだけです。

```
trace(this.scale9Grid);
```

それでもバグが発生し、除去できない場合は、以下のコードで復元を実行します:

```
this.scale9Grid = new Rectangle(24, 35, 363, 138);
```

ビットマップ ベースの背景サンプルについては、scale9grid_window3.fla と scale9grid_window3.swf を参照してください。このファイルでは、コーナーと境界線のムービー クリップは、ヒット テスト機能をサポートすることだけが目的なので、完全に透明になっています。背景イメージのコーナーの形状を再現しているにすぎません。



本書で前述したように、Scaleform では単一のイメージを背景として使用できますが、Flash ではスライスして手動でタイリングしなければ使用できません。また、上記の例のようなコーナーや境界線

は Adobe Flash Player では正しく動作しません。Adobe Flash Player ではヒット テスト変形を正しく維持できないからです。同様に、Scale9Grid クリップの回転と歪曲も、標準の Flash Player では正しく動作しません。Scaleform では、サイズ変更可能なウィンドウの作成を、できる限りシンプルで効率的にできるよう工夫し、Scale9Grid で正確に変形をサポートできるように努めました。