Autodesk®
Scaleform®

Scale9Grid 用户手册

本文件详细介绍了如何在 Scaleform 中使用 Scale9Grid 功能来创建尺寸可调的窗口、面板和按钮。

作者： Maxim Shemanarev, Michael Antonov
版本： 1.01
最后编辑日： 2008 年 3 月 21 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

如何联系 Autodesk Scaleform：

文件	Scale9Grid User Guide
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
网址	www.scaleform.com
电子邮件	info@scaleform.com

直线电话	(301) 446-3200
传真	(301) 446-3199

目录

1 介绍： 在 Flash 和 Scaleform 中使用 Scale9Grid

Scale9Grid 被用在 Flash®中，用于影片剪辑中各种元素的扩展。Scale9Grid 使得开发人员所制作的影片剪辑和用户界面元素可以智能化地改变其尺寸。而不象平面设计元素中通常使用的线性扩展。

在 Flash 的术语中，Scale9Grid 的另外一个名称是九宫格扩展（9-Slice Scaling）。一旦选中了九宫格扩展的选项，影片剪辑被一个网格划分成为九个部分，而其中的每个部分都将独立进行扩 展。为了保持整个影片剪辑在视觉效果上的整体感，边角不被扩展，但是图形的其它部分都被扩展（这和传统上的延伸不同）。

在 Flash Studio 中，在影片剪辑的“Symbol properties”对话框中提供了九宫扩展（9-Slice Scaling）的选框。此外，在 ActionScript 中提供了 MovieClip.scale9Grid 和 Button.scale9Grid 的属性，方便对分段扩展进行编程控制。如需了解 Flash 中对于 Scale9Grid 使用的更多信息，请察看 Flash Studio 提供的文档。

Adobe Flash 播放器中对于 Scale9Grid 给出了一些限制，使得它的实际应用变得相对困难。举例来说，标准的 Flash 播放器中不支持图片划分，因此无法在自由转动或其它的转换中支持 Scale9Grid。Scaleform 给予 Scale9Grid 更好和更稳定的支持，并且和 Flash 有着很好的兼容性。Scale9Grid 在 Adobe Flash 和 Scaleform 中的一些特性简要列于下表：

Scale9Grid 特性	**Adobe Flash 播放器**	**Scaleform**
转换	基于 X/Y 轴	自由转换
位图	基于限界框的平均扩展	根据 Scale9Grid 自动扩展
渐变和图像填充	基于限界框的平均扩展	基于限界框的平均扩展
次级元素（嵌套影片剪辑、按钮和图片）	不支持	支持
嵌套 Scale9Grid	不支持	不支持
包围元素的转换	支持	支持
文字	常规扩展	常规扩展

Scaleform 对 Flash 向后兼容。这就意味着在 Flash 中可以使用的 Scale9Grid 功能在 Scaleform 中同样有效。但是，在 Flash 中不被支持的 Scale9Grid 特性可能会在 Scaleform 中有所不同，但总是变得更加实用。举例来说，次级元素（嵌套影片剪辑）对于创建尺寸可以互动调整的窗口而言非常 重要。两种不同播放器中对此解决方法上的差异在本文件中将予以讨论。

为了能为开发人员提供 Scale9Grid 的第一手实例，本文件中将引用很多 FLA/SWF 文件。这些文件可以在 Scaleform 开发人员中心网站上下载得到。

2 转换

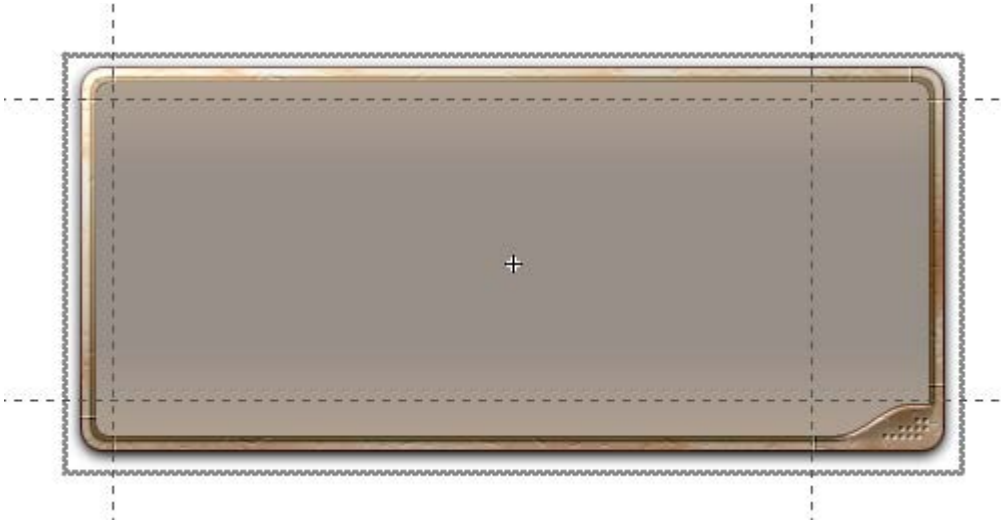
和 Flash 不同，Scaleform 使用了 Scale9Grid 可以支持影片剪辑的任意转换。在 Flash 中，参考的影片剪辑无法进行旋转或倾斜。但是，可以将结果剪辑包括在一个精灵（Sprite）中，对这个精灵再使用旋转或倾斜的功能。包围精灵的转换方法在 Flash 和 Scaleform 中都是相似的。对于高层级的元素进行转换即可以在相同 Scale9Grid 下绘制出不同的图形，具体见下面的例子：

3 处理位图

虽然在 Flash 和 Scaleform 中，对于渐变和位图填充的转换是相似的，但是独立位图的处理却是有所不同的。在 Flash 中，当一个独立的位图被置于一个九宫扩展的影片剪辑中时，播放器会忽视任何有关 Scale9Grid 的设置，除非用户将该位图进行了九宫划分。Scaleform 播放器会自动对位图进行九宫划分，以便能通过 Scale9Grid 进行正确扩展。这项功能大大简化了开发人员的工作，并且在不需要人工进行图片划分的情况下就能自动创建尺寸可调的按钮和窗口。此外，自动划分的功能可以免除对 EdgeAA 处理结果缝合时产生的抗锯齿缝隙。自动划分在下面的例子中进行了具体的介绍。

假设设计师创建了一个如下的位图，代表的是窗口的背景：

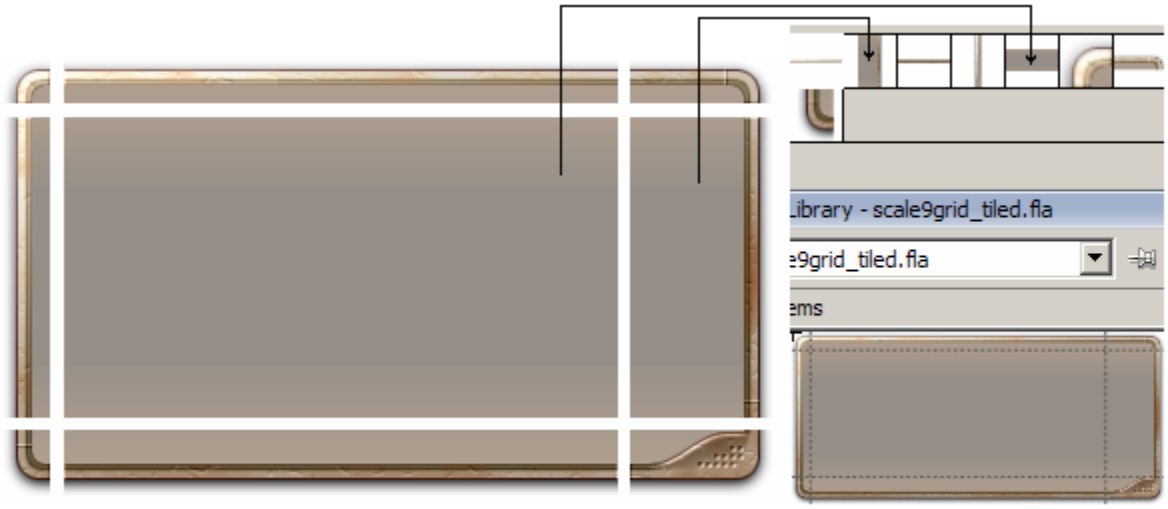
然后，设计师使用 Scale9Grid，这样边角处的图形在扩展时可以得到保留。最简单的方法是创建一个影片剪辑，并如下这样对其使用 Scale9Grid：



但是，在 Flash 中，这样做得不到我们期望的效果。如果你对结果剪辑进行扩展，则扩展后的效果就和没有使用 Scale9Grid 的一样：

在 Scaleform 中，这样操作可以得到理想效果，即边角图形可以得到保留：

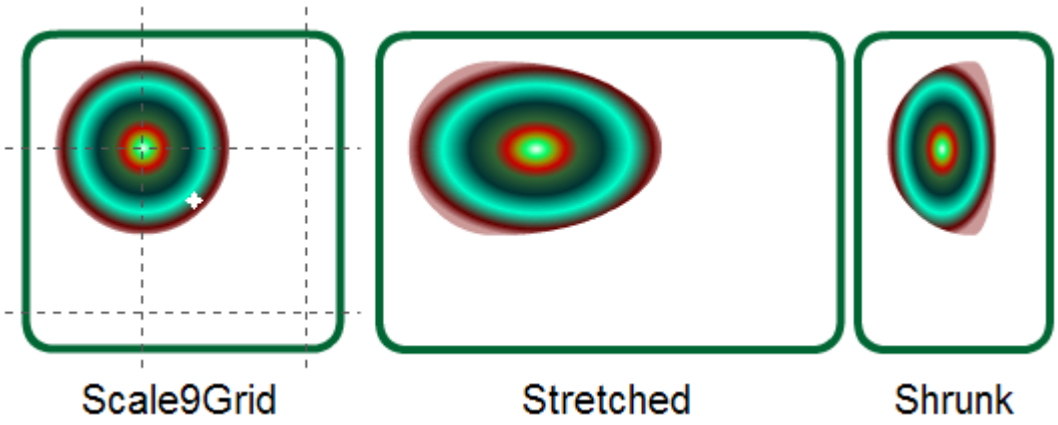
Adobe Flash 根据图形的包围框来计算平均扩展。这就意味着，还是可以在 Flash 中对位图进行九宫扩展的，但是需要将图片划分成和 Scale9Grid 完全对应的 9 个图形：



Scaleform 在播放 Flash 文件时可以自动完成这项操作。

如果对图形进行任意旋转或倾斜，则 Scaleform 的处理效果和 Flash 相似。对上述的“手工划分”窗口的举例见文件 scale9grid_tiled.fla 和 scale9grid_tiled.swf。

带渐变和位图填充的图形在 Flash 和 Scaleform 中的显示是相似的。请注意，无论是渐变还是位图填充都无法保留 Scale9Grid 转换的效果。实际上，转换仅作用在了矢量的图形轮廓上，没有作用在填充内容上，举例如下：

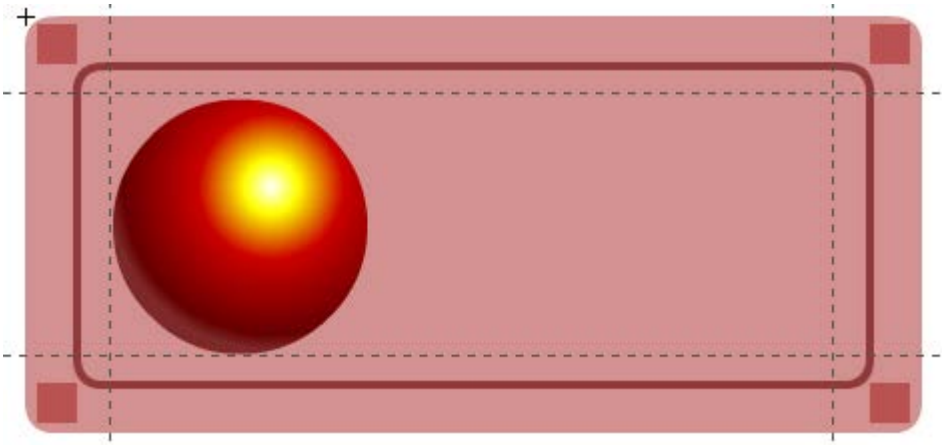


4 交互式窗口

这个部分将提供交互式尺寸可调窗口的实例，向你给出允许用户输入的 ActionScript 源代码。

4.1 Flash 兼容的例子

下面的例子显示了如何创建一个兼容 Flash 的交互式尺寸可调窗口。本例子对应的文件为 scale0grid_window1.fla 和 scale9grid_window1.swf。假设我们在 Flash Studio 中创建了下面这样一个影片剪辑：



整个的设想是通过拖动角上的方框来改变窗口的尺寸，通过拖动窗口上其它任意位置来移动窗口。这里的一个主要问题是 Flash 和 ActionScript 中不提供点击测试（hit-testing）方面的功能，该功能可以用来区别一个影片剪辑中存在的不同图形和层次。此外，九宫扩展同样不适用于 Flash 中的嵌套剪辑。这就使得尺寸调整逻辑的编程变得很困难。作为对点击测试功能的替代，编程人员不得不在程序中检查图形的座标。下面这个例子就是和上述给出的影片剪辑对应的 ActionScript 程序。

```
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;
	}
}
```

正如你所见，编程逻辑相当复杂。我们需要影片剪辑的包围框和其它的一些变量。其中用来控制移动的重要变量是 XMode 和 Ymode。变量值为“-1”表示我们拖动了窗口的左边界或上边界。同样，变量值为“1”表示我们拖动了窗口的右边界或下边界。

主要的点击测试逻辑包括在 onPress ( )函数中。首先，我们必须计算扩展系数 kx 和 ky，因为鼠标座标不会自动保存 Scale9Grid 转换逻辑：

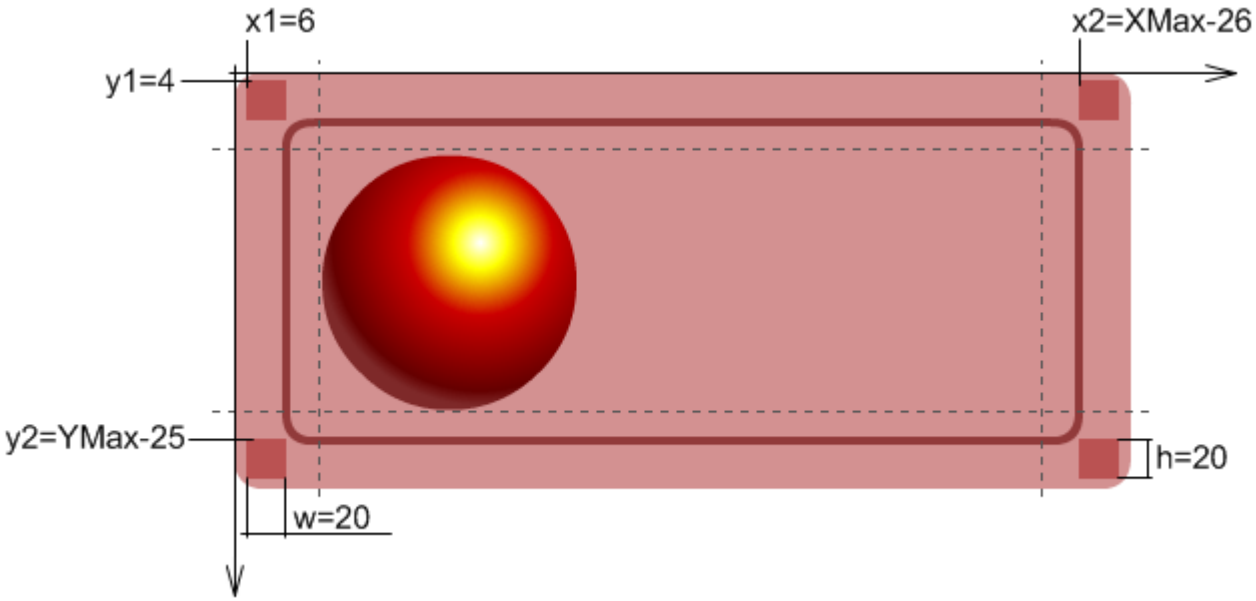
```
var kx = (this.XMax - this.XMin) / this._width;
var ky = (this.YMax - this.YMin) / this._height;
```

然后，我们计算点击测试矩形（在本例中是正方形），即：

Left-Top: x1, y1, x1+w, y1+h
Right-Top: x2, y1, x2+w, y1+h
Left-Bottom: x1, y2, x1+w, y2+h
Right-Bottom: 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;
```

请注意常量值 6、4、26、25。它们是准确对应影片剪辑中图形的座标值。事实上，处于角上的方框并非必需的，可以去除。这个方法的最大的缺点是：如果改变图形，则你需要针对性地修改 ActionScript 代码。下面这个图片显示了所使用常量代表的含义：



在下面一步中，我们通过程序来核对座标，并且对应地进行尺寸调整。

```
var xms = this._xmouse;
```

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

显然，在角上图形越是复杂，则编程中的逻辑也就越复杂。举例来说，如果角上是下面这样的图形，则就需要对两个矩形都进行编程：

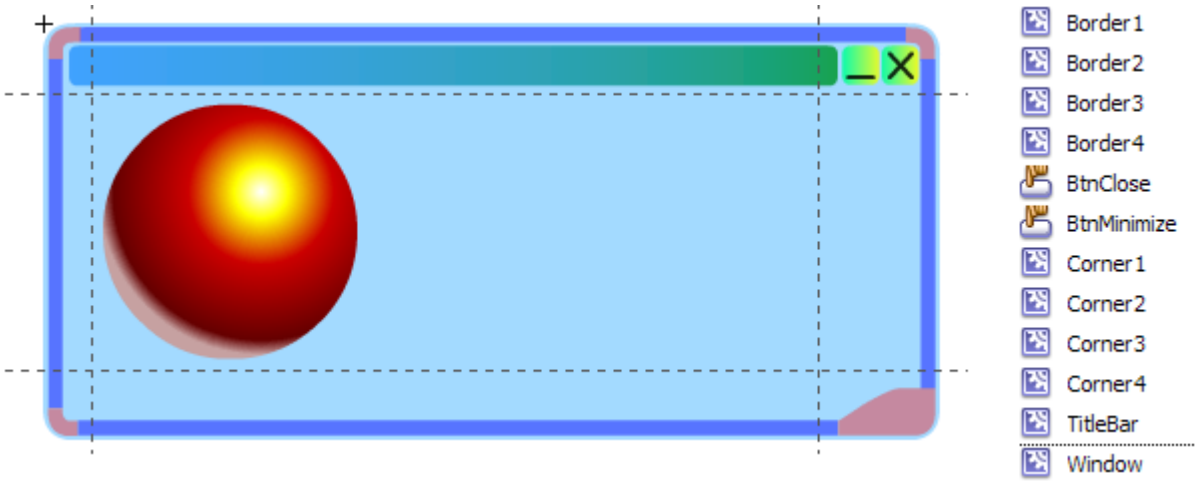
如果角上是圆形或不规则的其它图形，则编程就会变得非常复杂。此外，这个例子无法通过边界调节尺寸。

ActionScript 逻辑的剩下部分主要用于改变窗口座标和尺寸，同时确保不得超过高度和宽度方面的最小限制。

请注意，如果窗口背景是一个图案，则需要手工进行划分，以便可以和 Flash 兼容。

4.2 高级的Scaleform 举例

下面这个更加高级的例子可以在 Scaleform 中使用，但是在 Flash 中则无效。它所基于的原理是 Scaleform 可以自动且正确转换嵌套影片剪辑，这就意味着你可以针对任何随意的和不规则的图形都使用鼠标的点击测试功能。对应的举例文件为 scale9grid_window2.fla 和 scale9grid_window2.swf。这个例子中使用的图案更加复杂，功能性更加强大，但是所基于的却是普通的可以再使用的 ActionScript 代码，且不需要使用到“硬编码”的座标值。



举例的 Flash 文件中包括了相对独立的影片剪辑和按钮，如 Border1 至 Border4（蓝色）、Corner1 至 Corner4、TitleBar 等等。鉴于 Scaleform 可以通过 Scale9Grid 来正确调整嵌套影片剪辑的大小，你所需要做的就是为影片剪辑分配函授。ActionScript 变得简单和清楚。请　　注意，这里的 OnResize ( ) 函数和上面例子中的一样。

```
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;
	}
}
```

这个方法有两个缺点。首先，它和 Adobe Flash 播放器不兼容。第二个问题是它里面包括很多影片剪辑，导致产生过多的绘图图元和占用过多内存。这是你为简化和通用解决方案付出的代价。如果不希望有过多的绘图图元，则可以考虑将上面两种方法合而为一，使用一个窗口的图形框来替代边界和角。这样需要在 ActionScript 代码中增加编程逻辑，但是应该相比第一个例子而言还是要简单和“强壮”。这样做的一个主要的好处是点击测试的影片剪辑可以是随意的图形，带有不规则的边角。

这里有必要提出的是，Flash Studio 在通过 Scale9Grid 生成 SWF 文件时存在一个 BUG。你可以通过 scale9grid_window2.fla 这个文件再生成这个 BUG。如果你选择“TitleBar”层，然后在窗口正中绘制一个小小的矩形，Scale9Grid 就会出现错误。可能 Flash Studio 不允许在相同层里面同时有嵌套剪辑和其它图片。在某些时候，在你删除那个矩形后，Scale9Grid 仍旧无法正常工作。“Save and Compact”的功能可以用来应对这种情况。

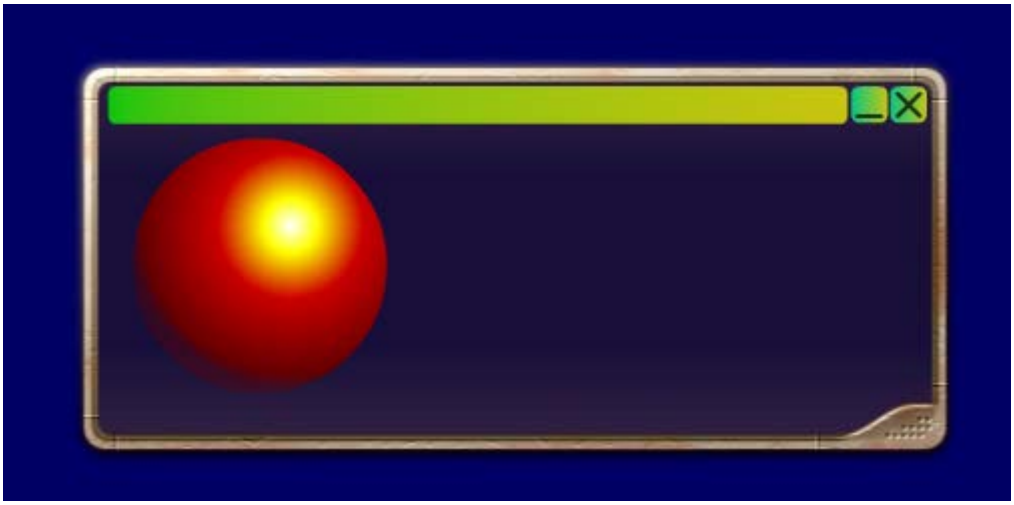
如有上面这种情况，你仍可以通过 ActionScript 来恢复 Scale9Grid。你可以在其正常时加上如下代码进行跟踪：

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

如出现 BUG，并且无法清除，则你可以使用下面代码来恢复：

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

如需背景是图案的例子，请参见 scale9grid_window3.fla 和 scale9grad_window3.swf。在这个例子中，边界和角上的影片剪辑完全透明，因为它们存在的唯一理由是支持点击测试功能。透明后它们仅仅是重现了背景的图案。



正如本文前面所说，在 Scaleform 中你可以将背景作为一个图形来进行处理，但是在 Flash 中你必须手动加以划分。此外，边界和角上的影片剪辑功能在 Adobe Flash 播放器中无法实现，因为其不能正确保存点击测试转换。同样，Scale9Grid 剪辑的旋转和倾斜在标准的 Flash 播放器中也无法正常工作。但是，使用了 Scaleform 后，我们可以以最简单和高效的方式来创建尺寸可调的窗口，同时也能确保 Scale9Grid 对图形转换的正确支持。