Autodesk®
Scaleform®

Scale9Grid 用戶手冊

本檔詳細介紹了如何在 Scaleform 中使用 Scale9Grid 功能來創建尺寸可調的視窗、面板和按鈕。

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

如何聯繫 Autodesk Scaleform：

文件	Scale9Grid User Guide
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網址	www.scaleform.com
電子郵件	info@scaleform.com
直線電話	(301) 446-3200
傳真	(301) 446-3199

目录

介紹： 在 Flash 和 Scaleform 中使用 Scale9Grid

Scale9Grid 被用在 Flash®中，用於影片剪輯中各種元素的擴展。Scale9Grid 使得開發人員所製作的影片剪輯和使用者介面元素可以智慧化地改變其尺寸。而不象平面設計項目中通常使用的線性擴展。

在 Flash 的術語中，Scale9Grid 的另外一個名稱是九宮格擴展（9-Slice Scaling）。一旦選中了九宮格擴展的選項，影片剪輯被一個網格劃分成為九個部分，而其中的每個部分都將獨立進行擴 展。為了保持整個影片剪輯在視覺效果上的整體感，邊角不被擴展，但是圖形的其它部分都被擴展（這和傳統上的延伸不同）。

在 Flash Studio 中，在影片剪輯的“Symbol properties”對話方塊中提供了九宮擴展（9-Slice Scaling）的選框。此外，在 ActionScript 中提供了 MovieClip.scale9Grid 和 Button.scale9Grid 的屬性，方便對分段擴展進行程式設計控制。如需瞭解 Flash 中對於 Scale9Grid 使用的更多資訊，請察看 Flash Studio 提供的文檔。

Adobe Flash 播放機中對於 Scale9Grid 給出了一些限制，使得它的實際應用變得相對困難。舉例來說，標準的 Flash 播放機中不支援圖片劃分，因此無法在自由轉動或其它的轉換中支持 Scale9Grid。Scaleform 給予 Scale9Grid 更好和更穩定的支持，並且和 Flash 有著很好的相容性。Scale9Grid 在 Adobe Flash 和 Scaleform 中的一些特性簡要列於下表：

Scale9Grid 特性	**Adobe Flash 播放機**	**Scaleform**
轉換	基於 X/Y 軸	自由轉換
點陣圖	基於限界框的平均擴展	根據 Scale9Grid 自動擴展
漸變和圖像填充	基於限界框的平均擴展	基於限界框的平均擴展
次級元素（嵌套影片剪輯、按鈕和圖片）	不支持	支持
嵌套 Scale9Grid	不支持	不支持
包圍元素的轉換	支持	支持
文字	常規擴展	常規擴展

Scaleform 對 Flash 向後相容。這就意味著在 Flash 中可以使用的 Scale9Grid 功能在 Scaleform 中同樣有效。但是，在 Flash 中不被支持的 Scale9Grid 特性可能會在 Scaleform 中有所不同，但總是變得更加實用。舉例來說，次級元素（嵌套影片剪輯）對於創建尺寸可以互動調整的視窗而言非常 重要。兩種不同播放機中對此解決方法上的差異在本文件中將予以討論。

為了能為開發人員提供 Scale9Grid 的第一手實例，本檔中將引用很多 FLA/SWF 文件。這些檔可以在 Scaleform 開發人員中心網站上下載得到。

轉換

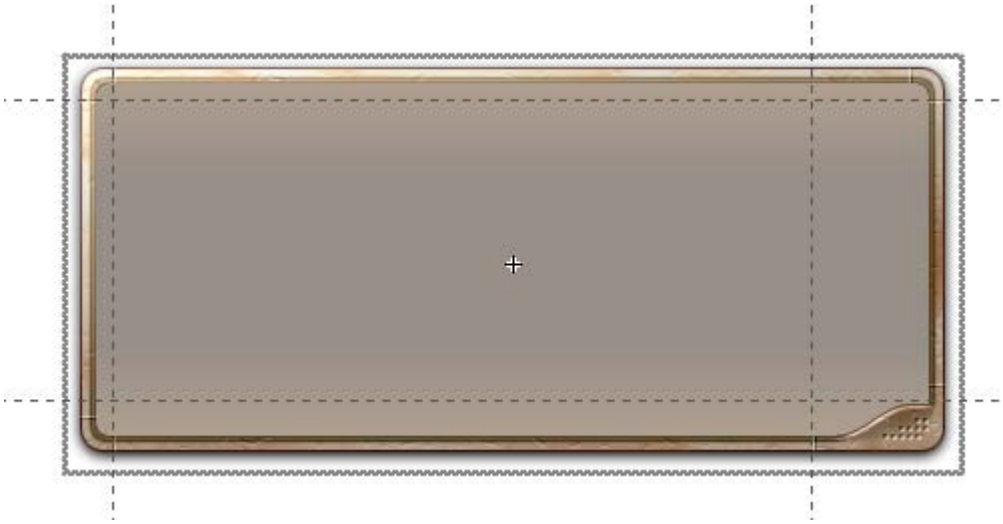
和 Flash 不同，Scaleform 使用了 Scale9Grid 可以支援影片剪輯的任意轉換。在 Flash 中，參考的影片剪輯無法進行旋轉或傾斜。但是，可以將結果剪輯包括在一個精靈（Sprite）中，對這個精靈再使用旋轉或傾斜的功能。包圍精靈的轉換方法在 Flash 和 Scaleform 中都是相似的。對於高層級的元素進行轉換即可以在相同 Scale9Grid 下繪製出不同的圖形，具體見下面的例子：

處理點陣圖

雖然在 Flash 和 Scaleform 中，對於漸變和點陣圖填充的轉換是相似的，但是獨立點陣圖的處理卻是有所不同的。在 Flash 中，當一個獨立的點陣圖被置於一個九宮擴展的影片剪輯中時，播放機會忽視任何有關 Scale9Grid 的設置，除非用戶將該點陣圖進行了九宮劃分。Scaleform 播放機會自動對點陣圖進行九宮劃分，以便能通過 Scale9Grid 進行正確擴展。這項功能大大簡化了開發人員的工作，並且在不需要人工進行圖片劃分的情況下就能自動創建尺寸可調的按鈕和視窗。此外，自動劃分的功能可以免除對 EdgeAA 處理結果縫合時產生的抗鋸齒縫隙。自動劃分在下面的例子中進行了具體的介紹。

假設設計師創建了一個如下的點陣圖，代表的是視窗的背景：

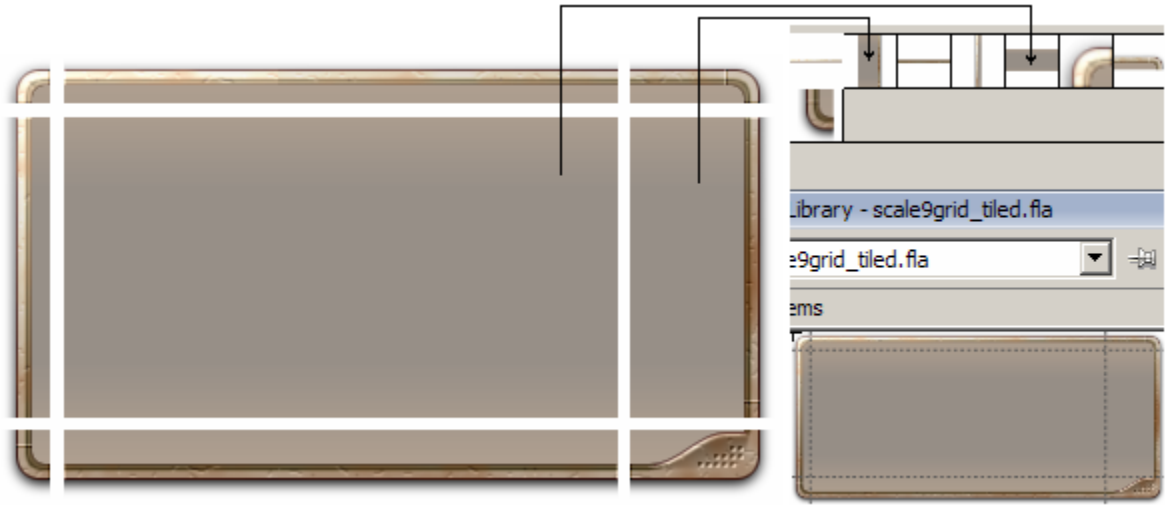
然後，設計師使用 Scale9Grid，這樣邊角處的圖形在擴展時可以得到保留。最簡單的方法是創建一個影片剪輯，並如下這樣對其使用 Scale9Grid：



但是，在 Flash 中，這樣做得不到我們期望的效果。如果你對結果剪輯進行擴展，則擴展後的效果就和沒有使用 Scale9Grid 的一樣：

在 Scaleform 中，這樣操作可以得到理想效果，即邊角圖形可以得到保留：

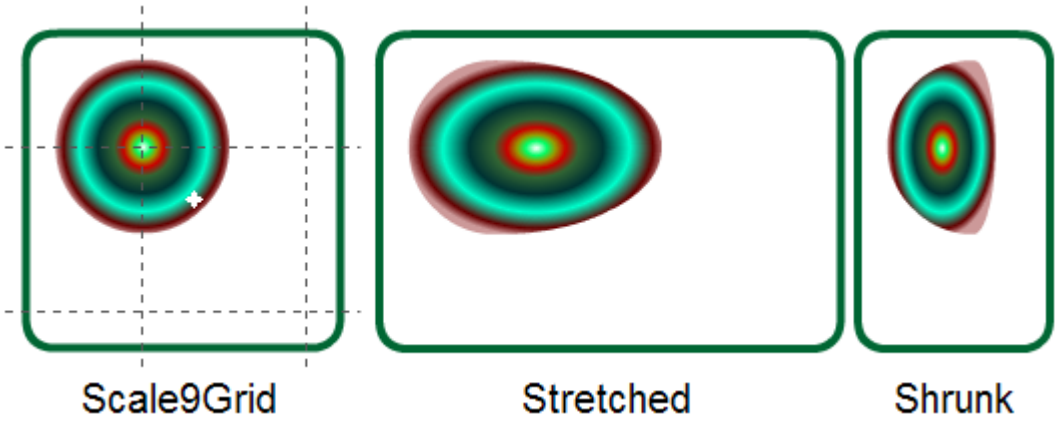
Adobe Flash 根據圖形的包圍框來計算平均擴展。這就意味著，還是可以在 Flash 中對點陣圖進行九宮擴展的，但是需要將圖片劃分成和 Scale9Grid 完全對應的 9 個圖形：



Scaleform 在播放 Flash 檔時可以自動完成這項操作。

如果對圖形進行任意旋轉或傾斜，則 Scaleform 的處理效果和 Flash 相似。對上述的“手工劃分”窗口的舉例見檔 scale9grid_tiled.fla 和 scale9grid_tiled.swf。

帶漸變和點陣圖填充的圖形在 Flash 和 Scaleform 中的顯示是相似的。請注意，無論是漸變還是點陣圖填充都無法保留 Scale9Grid 轉換的效果。實際上，轉換僅作用在了向量的圖形輪廓上，沒有作用在填充內容上，舉例如下：

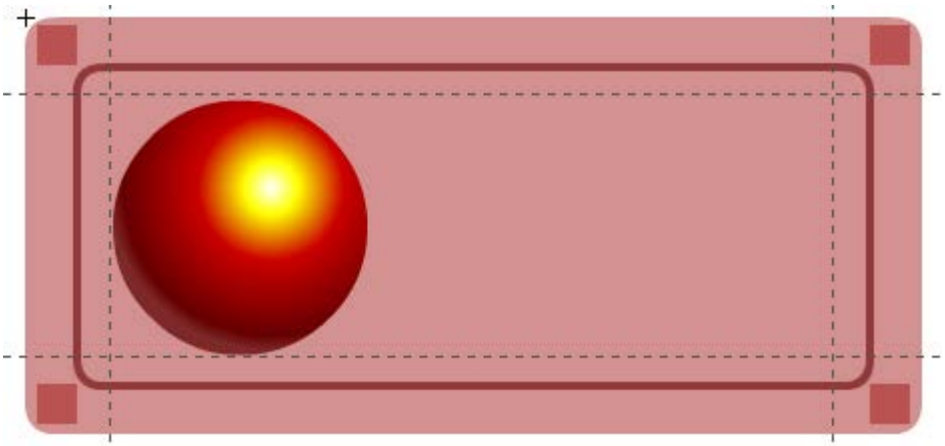


互動式視窗

這個部分將提供互動式尺寸可調視窗的實例，向你給出允許用戶輸入的 ActionScript 原始程式碼。

Flash 相容的例子

下面的例子顯示了如何創建一個相容 Flash 的互動式尺寸可調視窗。本例子對應的檔為 scale0grid_window1.fla 和 scale9grid_window1.swf。假設我們在 Flash Studio 中創建了下面這樣一個影片剪輯：



整個的設想是通過拖動角上的方框來改變視窗的尺寸，通過拖動視窗上其它任意位置來移動視窗。這裡的一個主要問題是 Flash 和 ActionScript 中不提供點擊測試（hit-testing）方面的功能，該功能可以用來區別一個影片剪輯中存在的不同圖形和層次。此外，九宮擴展同樣不適用於 Flash 中的嵌套剪輯。這就使得尺寸調整邏輯的程式設計變得很困難。作為對點擊測試功能的替代，程式設計人員不得不在程式中檢查圖形的座標。下面這個例子就是和上述給出的影片剪輯對應的 ActionScript 程式。

```
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;
	}
}
```

正如你所見，程式設計邏輯相當複雜。我們需要影片剪輯的包圍框和其它的一些變數。其中用來控制移動的重要變數是 XMode 和 Ymode。變數值為“-1”表示我們拖動了視窗的左邊界或上邊界。同樣，變數值為“1”表示我們拖動了視窗的右邊界或下邊界。

主要的點擊測試邏輯包括在 onPress ( )函數中。首先，我們必須計算擴展係數 kx 和 ky，因為滑鼠座標不會自動保存 Scale9Grid 轉換邏輯：

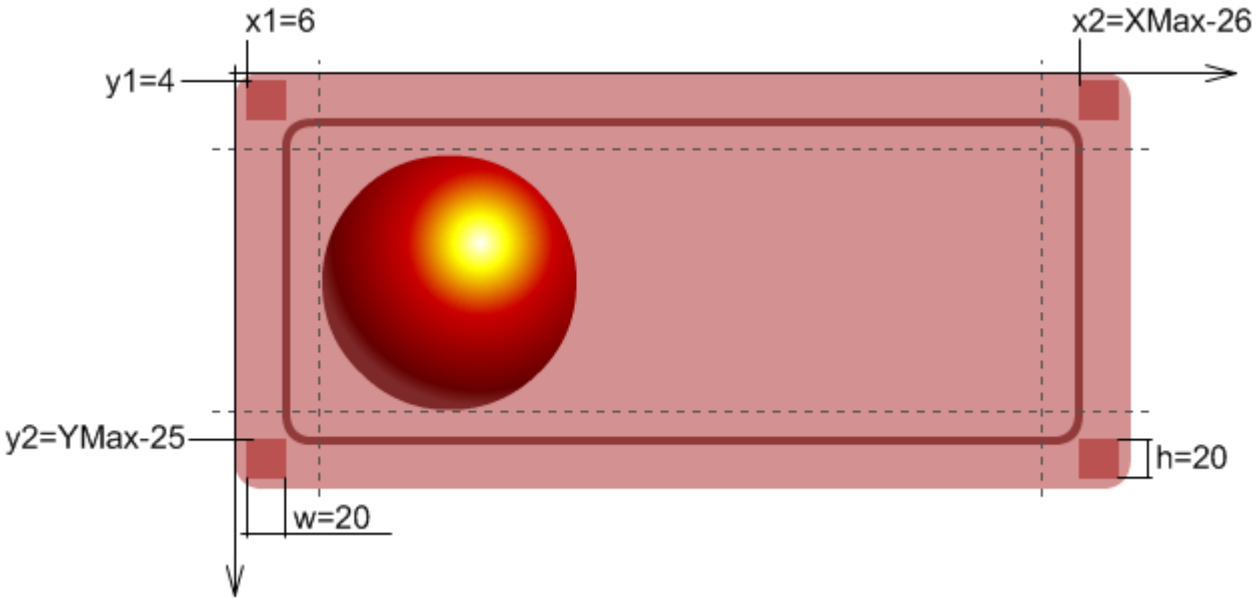
```
var kx = (this.XMax - this.XMin) / this._width;
var ky = (this.YMax - this.YMin) / this._height;
```

然後，我們計算點擊測試矩形（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;
```

請注意常量值 6、4、26、25。它們是準確對應影片剪輯中圖形的座標值。事實上，處於角上的方框並非必需的，可以去除。這個方法的最大的缺點是：如果改變圖形，則你需要針對性地修改 ActionScript 代碼。下面這個圖片顯示了所使用常量代表的含義：



在下面一步中，我們通過程式來核對座標，並且對應地進行尺寸調整。

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

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

顯然，在角上圖形越是複雜，則程式設計中的邏輯也就越複雜。舉例來說，如果角上是下面這樣的圖形，則就需要對兩個矩形都進行程式設計：

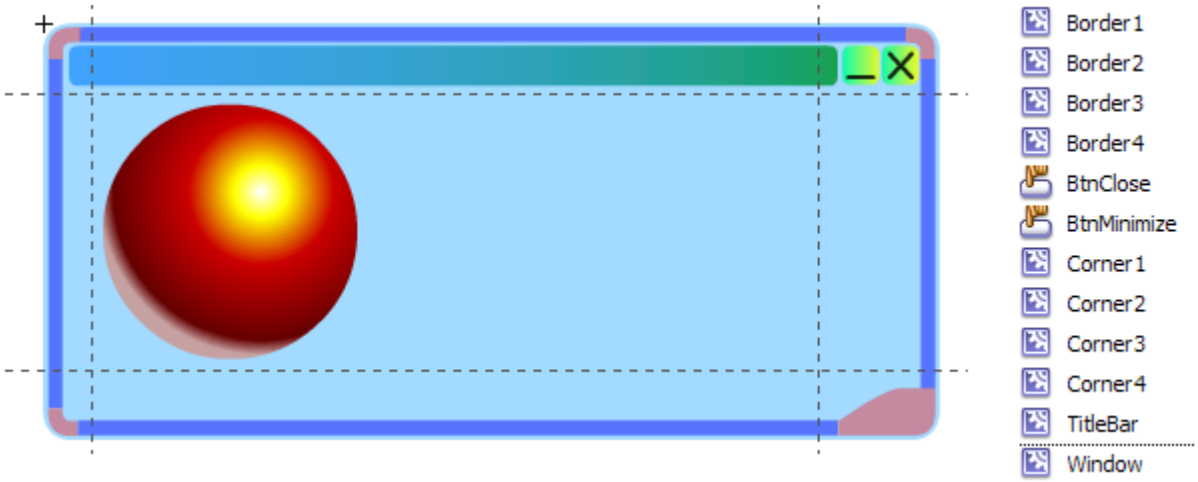
如果角上是圓形或不規則的其它圖形，則程式設計就會變得非常複雜。此外，這個例子無法通過邊界調節尺寸。

ActionScript 邏輯的剩下部分主要用於改變視窗座標和尺寸，同時確保不得超過高度和寬度方面的最小限制。

請注意，如果視窗背景是一個圖案，則需要手工進行劃分，以便可以和 Flash 相容。

高級的 Scaleform 舉例

下面這個更加高級的例子可以在 Scaleform 中使用，但是在 Flash 中則無效。它所基於的原理是 Scaleform 可以自動且正確轉換嵌套影片剪輯，這就意味著你可以針對任何隨意的和不規則的圖形都使用滑鼠的點擊測試功能。對應的舉例檔為 scale9grid_window2.fla 和 scale9grid_window2.swf。這個例子中使用的圖案更加複雜，功能性更加強大，但是所基於的卻是普通的可以再使用的 ActionScript 代碼，且不需要使用到“硬編碼”的座標值。



舉例的 Flash 檔中包括了相對獨立的影片剪輯和按鈕，如 Border1 至 Border4（藍色）、Corner1 至 Corner4、TitleBar 等等。鑒於 Scaleform 可以通過 Scale9Grid 來正確調整嵌套影片剪輯的大小，你所需要做的就是為影片剪輯分配函授。ActionScript 變得簡單和清楚。請　　注意，這裡的 OnResize ( ) 函數和上面例子中的一樣。

```
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;
	}
}
```

這個方法有兩個缺點。首先，它和 Adobe Flash 播放機不相容。第二個問題是它裡面包括很多影片剪輯，導致產生過多的繪圖圖元和佔用過多記憶體。這是你為簡化和通用解決方案付出的代價。如果不希望有過多的繪圖圖元，則可以考慮將上面兩種方法合而為一，使用一個視窗的圖形框來替代邊界和角。這樣需要在 ActionScript 代碼中增加程式設計邏輯，但是應該相比第一個例子而言還是要簡單和“強壯”。這樣做的一個主要的好處是點擊測試的影片剪輯可以是隨意的圖形，帶有不規則的邊角。

這裡有必要提出的是，Flash Studio 在通過 Scale9Grid 生成 SWF 檔時存在一個 BUG。你可以通過 scale9grid_window2.fla 這個檔再生成這個 BUG。如果你選擇“TitleBar”層，然後在視窗正中繪製一個小小的矩形，Scale9Grid 就會出現錯誤。可能 Flash Studio 不允許在相同層裡面同時有嵌套剪輯和其它圖片。在某些時候，在你刪除那個矩形後，Scale9Grid 仍舊無法正常工作。“Save and Compact”的功能可以用來應對這種情況。

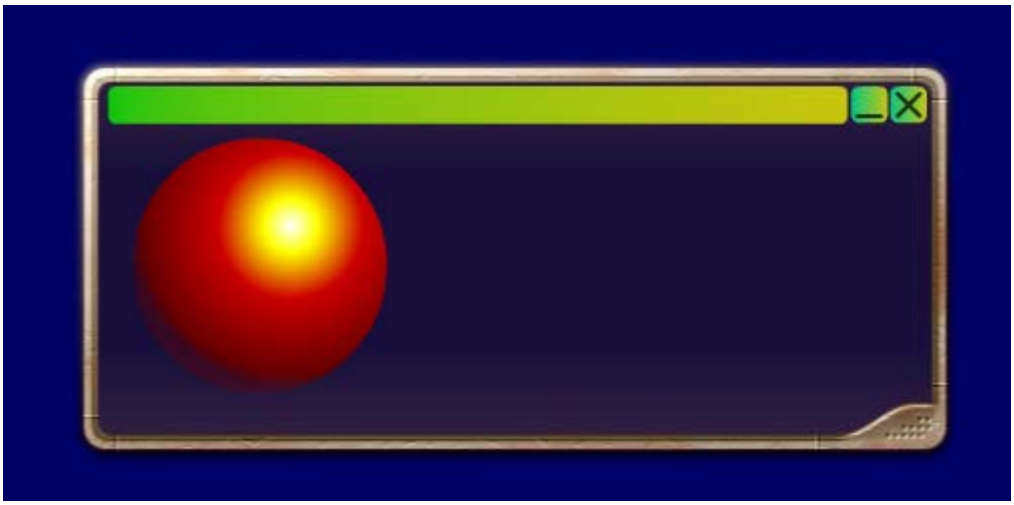
如有上面這種情況，你仍可以通過 ActionScript 來恢復 Scale9Grid。你可以在其正常時加上如下代碼進行跟蹤：

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

如出現 BUG，並且無法清除，則你可以使用下面代碼來恢復：

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

如需背景是圖案的例子，請參見 scale9grid_window3.fla 和 scale9grad_window3.swf。在這個例子中，邊界和角上的影片剪輯完全透明，因為它們存在的唯一理由是支持點擊測試功能。透明後它們僅僅是重現了背景的圖案。



正如本文前面所說，在 Scaleform 中你可以將背景作為一個圖形來進行處理，但是在 Flash 中你必須手動加以劃分。此外，邊界和角上的影片剪輯功能在 Adobe Flash 播放機中無法實現，因為其不能正確保存點擊測試轉換。同樣，Scale9Grid 剪輯的旋轉和傾斜在標準的 Flash 播放機中也無法正常工作。但是，使用了 Scaleform 後，我們可以以最簡單和高效的方式來創建尺寸可調的視窗，同時也能確保 Scale9Grid 對圖形轉換的正確支援。