Autodesk®
Scaleform®

Scaleform 3Di

本文说明如何使用 Scaleform 3.2 及更高版本的 3D 特性。

作者： Mustafa Thamer

版本： 2.03

上次编辑时间： 2012 年 5 月 9 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

如何联系 Autodesk Scaleform：

文档	Scaleform 3Di
地址	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
网站	www.scaleform.com
电子邮箱	info@scaleform.com
电话	(301) 446-3200
传真	(301) 446-3199

目录

1 概述

Scaleform® 3Di™ 使开发人员可以将其应用程序 UI 带到具有新的 3D Flash 渲染和动画功能的下一个维度。

从 Scaleform 3.2 开始，Scaleform 增加了一组简单而强大的 ActionScript 扩展，它们能够提供基本的 3D 功能。尽管 Scaleform 3.2 以 ActionScript 2.0 为基础，但还是另外提供了一组 3D AS 扩展 --- 与 AS 3.0 中提供的相似。

利用这些新的扩展（_z、_zscale、_xrotation、_yrotation、_matrix3d 和 _perspfov），开发人员可以为菜单、HUD 和游戏内 UI 创建绝妙的动画 3D 接口。这些 3D 扩展可应用于任何电影剪辑、按钮或文本框对象。

C++ 中也有同样的 3D 功能，作为 Scaleform Direct Access API 的组成部分。新的 API 允许开发人员通过伸缩、旋转或偏移电影剪辑、按钮或文本框，直接将其进行转换，保存为 3D GFx::Value。

此外，开发人员可以将自己的 3D 渲染对象或流式视频添加到 3D Flash 接口，以创造一种独特的下一代体验。

2 与 Flash 10 的相似之处

一般来说，Scaleform 提供的 3Di 支持与 Flash 10 中的 3D 非常相似，而且行为也可能相同。3Di 是使用一组内置到 Scaleform 中的基本 AS2 扩展实现的。

2.1 局限

与 Flash 10 中的 3D 支持相似，3Di 也有如下局限：

1. 不执行深度排序，因此，可能无法以正确的顺序绘制对象。默认情况下，绘画顺序仍然通过对顺序进行分层来指定。
2. 不使用隐面消除 (Back-face culling)。仍然绘制背对观众的对象。

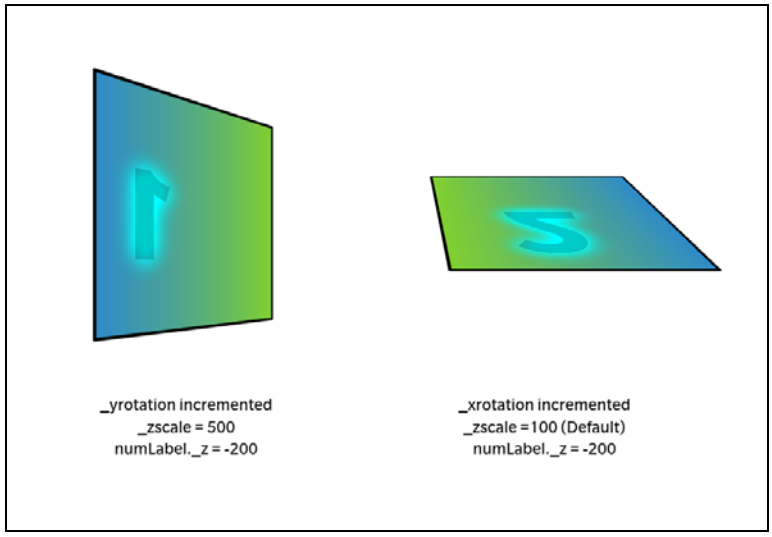

图 1：不选择 (Culled) 或画背面的背面向数目示例

2.2 实现

Scaleform 3Di 使用硬件加速三角形来渲染形状，而 Flash 10 将 3D 视图渲染成一个位图，然后将包含该位图的矢量矩形用作显示填充。

当一个对象同时具有 2D 和 3D 属性时，优先应用 3D 属性。此细节对于利用自由转换工具以及 3D 操作旋转一个对象时获得预期结果非常重要。Flash 10 具有相同的行为。

3 以 3D 显示 Flash

使用 Scaleform 3.2，有两种不同的方法可用来以 3D 形式显示 Flash 内容。

3.1 渲染到纹理

第一个选项是使用传统方法将 Flash 渲染成一种纹理，然后将该渲染纹理作为一个地图应用到场景中的一个 3D 表面上。在 Scaleform 3.2 之前，这是开发人员可以用 3D 可视化 Flash 的唯一方法。

“渲染到纹理”（Render to Texture）方法比 3Di 具有一个明显的优势。由于 Flash 内容以纹理方式映射到一个 3D 对象上，因而可以充分利用 3D 引擎的功能，并与 3D 世界的其余部分进行无缝混合。这样就可以进行正确的 Z（深度）排序、混合/透明以及剪辑（而不管是否想要对 UI 进行剪辑）。

另一方面，这种方法也存在若干与性能和内存相关的缺点，因为渲染纹理必需附加内存。而且，请注意，任何 Flash 交互必须通过游戏进行处理（通过以手动方式反向映射鼠标坐标并传递到 Scaleform）。最后，由于这种方法是基于纹理的，所以 Flash 内容将精确地显示，因而不会偏移具体电影剪辑或从彼此旋转。有关此方法的示例，请参阅“SWF 到纹理”(SWF to Texture) 演示，该演示可在 Scaleform SDK 浏览器中找到。

3.2 3Di

第二个选项（Scaleform 3.2 中的新特性）是使用 3Di --- 从 ActionScript 或 Direct Access API (C++)。与使用渲染纹理方法相比，3Di 具有许多优势。首先，不需要任何附加内存。其次，单个电影剪辑可以有其自己的嵌套式转换 – 每个都可以唯一地偏移或旋转，因而整个电影看起来不单调。最后，所有输入和交互都是由 Scaleform 自动处理的。应用程序只是像平常一样把输入事件传递到 Scaleform。

不过，3Di 确实也有一些局限性。特别是，由于它是由默认 Scaleform 渲染器处理的，并被视为一个 UI，因而不执行任何深度排序或隐面消除。因此，当 UI 对象在其他对象后面旋转时，绘制的顺序可能不正确。而且，混合 UI 元素与其他 3D 游戏对象无缝混合可能比较棘手，因为 UI 对象应匹配相同的相机、照明、后处理以及在用的其他图形效果。此外，许多 UI 元素是透明的，并要求特定的绘画混合顺序，而这一点是需要加以重视的。

4 3Di API

4.1 ActionScript 2 API

凭借 3Di，Flash 对象现在可以拥有另一个维度。增加第三个维度使用户可以查看对象向视点前进或后退。当对象移离相机时，它们由于透视投影而看起来较小。

默认情况下，对象是二维的，而且不使用或计算任何 3D 属性。不过，一旦设置一个 3D 属性（x 或 y 旋转、z 翻译或伸缩或 3d 矩阵），该对象就成为三维。在这一点上，自动创建一个 3D 转换矩阵，并指定给该对象。通过将该对象的 3d 矩阵设置为“空”(Null)（即，在 ActionScript 中，`foo._matrix3d = null`），可以使该对象返回到一种完全的 2D 状态。仅将旋转和 z 值设置为 0 将不会消除 3D 转换。

如上所述，3D 旋转允许围绕 3 个轴（x、y 或 z 轴）中的任何一轴旋转，这与只围绕 Z 轴旋转的 2D 情形相反。相似地，3D 伸缩增加一个额外的用于伸缩的轴，包含有 _zscale ActionScript 扩展。

Scaleform 中默认的 3D 坐标系和相机与 Flash 10 中的相同。它是一个右手坐标系，其中 +X 向右，+Y 向下，而 +Z 指向屏幕中（离开取景器）。默认相机面朝下对着 +Z 轴，FOV 角为 55 度。

可以将 3D 属性应用到下列 Flash 对象类型：

- 电影剪辑
- 文本框
- 按钮

为 3Di 增加了下列 ActionScript 扩展：

- **_z** - 设置对象的 Z 坐标（深度），默认值为 0。
- **_zscale** - 将 Z 向的对象伸缩设置一个百分比，默认值为 100。
- **_xrotation** - 设置对象围绕 X（横向）轴旋转，默认值为 0。
- **_yrotation** - 设置对象围绕 Y（纵向）轴旋转，默认值为 0。
- **_matrix3d** - 使用一组 16 个浮点数（4 x 4 矩阵）设置对象的完整 3D 转换。如果此值设置为 NULL，就会删除所有 3D 转换，而且对象将会变为 2D。
- **perspfov —** 在对象上设置透视视野 (FOV) 角度，有效角度必须 > 0 和 < 180，或者，要禁用透视并使用一个正交视图，则有效角度必须为 -1。如果不设置此值，对象就继承根 FOV 角，该角默认为 55 度。

使用这些新扩展需要在 ActionScript 中把全局变量 `gfxExtensions` 设置为 True（真）。

如果你计划修改一个电影剪辑实例的旋转，务请注意注册点的位置。默认情况下，当您创建一个电影剪辑符号时，注册点设置为左上角。如果您使用 _xrotation 或 _yrotation 将一个 3D 旋转应用到电影剪辑实

例，该对象就会围绕着注册点进行旋转。如果需要，可以将注册点设置为符号的中心。如果想要使用 3Di 将旋转应用到对象，务必谨慎设置注册点。

示例 1

围绕 Y 轴旋转一个电影剪辑 45 度：

```
_global.gfxExtensions = true;
mc1._yrotation = 45;
```

示例 2

沿着 X 轴以及一个 Z 标度连续旋转：

```
_global.gfxExtensions = true;
sq1._zscale = 500;

function rotateAboutXAxis(mc:MovieClip):Void
{
    mc._xrotation = mc._xrotation + 1;
    if (mc._xrotation > 360)
       {
        mc._xrotation = 0;
    }
}

onEnterFrame = function()
{
    rotateAboutXAxis(sq1);
}
```

示例 3

使用应用到根的 3D 翻译矩阵的 3D 转换：

```
function PixelsToTwips(iPixels):Number
{
      return iPixels*20;
}
```

```
function TranslationMatrix(tX, tY, tZ):Array
{
    var matrixC:Array = [ 1, 0, 0, 0,
                0, 1, 0, 0,
                0, 0, 1, 0,
                tX,tY,tZ,1];
    return matrixC;
}

this._matrix3d = TranslationMatrix(PixelsToTwips(100),PixelsToTwips(100),0);
```

4.2 来自 C++ 的 3Di

4.2.1 Matrix4x4

添加了一个名为 Matrix4x4 的新类以表示一个 4x4 行主要矩阵。此类具有用于创建和操纵各种 3D 矩阵类型（包括整个世界转换以及视图和投影）的所有适当函数。有关详细信息，请参阅 Render_Matrix4x4.h。

4.2.2 Direct Access

Direct Access 接口（经由 GFx::Value 类）已扩展为可支持 3Di。现在可以利用新的 API 来设置或查询 3D 属性。这些函数类似于 3Di ActionScript 扩展。

有关更多详细信息，请参阅 GFx_Player.h 中的 GFx::Value::DisplayInfo 类。

```
void    SetZ(Double z)
void    SetXRotation(Double degrees)
void    SetYRotation(Double degrees)
void    SetZScale(Double zscale)
void    SetFOV(Double fov)
void    SetProjectionMatrix3D(const Matrix4F *pmat)
void    SetViewMatrix3D(const Matrix3F *pmat)
bool    SetMatrix3D(const Render::Matrix3F& mat)

Double  GetZ() const
Double  GetXRotation() const
Double  GetYRotation() const
Double  GetZScale() const
Double  GetFOV() const
const Matrix4F* GetProjectionMatrix3D() const
const Matrix3F* GetViewMatrix3D() const
bool    GetMatrix3D(Render::Matrix3F* pmat) const
```

4.2.3 Render::TreeNode

Render::TreeNode 类接口添加了新的调用，以便于设置用于渲染 3D 电影的“视图”(View) 和“透视”(Perspective) 矩阵。尽管可以在单个显示对象上设置透视，但在电影根部设置往往方便一些，以便于立即影响到所有对象。

GFx_Player.h

```
   void     SetProjectionMatrix3D(const Matrix4F& m)
   void     SetViewMatrix3D(const Matrix3F& m);
```

4.2.4 示例：使用 Direct Access API 更改透视视野

```
   Ptr<GFx::Movie> pMovie = …;
   GFx::Value tmpVal;
   bool bOK = pMovie->GetVariable(&tmpVal, "_root.Window");
   if (bOK)
   {
         GFx::Value::DisplayInfo dinfo;
          bOK = tmpVal.GetDisplayInfo(&dinfo);
          if (bOK)
          {   // set perspectiveFOV to 150 degrees
              Double oldPersp = dinfo.GetFOV();
              dinfo.SetFOV(150);
              tmpVal.SetDisplayInfo(dinfo);
          }
   }
```

5 透视

上面的示例显示如何使用 Direct Access API 在电影上设置视野 (FOV)。透视和视图设置可应用到电影根部或个别显示对象上。如果一个对象返回的透视 FOV 值为 0，则它将从其父对象那里继承 FOV 值。

透视使距离观众较近的对象看起来更大，较远的对象则看起来更小。通过更改视野值，可以控制此效果的强度。此值越大，应用到沿着 z 轴移动的显示对象的失真度越强。FOV 值小的结果是伸缩度非常小，而此值大则使伸缩度大。此值必须大于 0 而小于 180，其中值为 1，则几乎不存在透视失真，而此值为 179 时，可创造一种失真的鱼眼透镜效果。

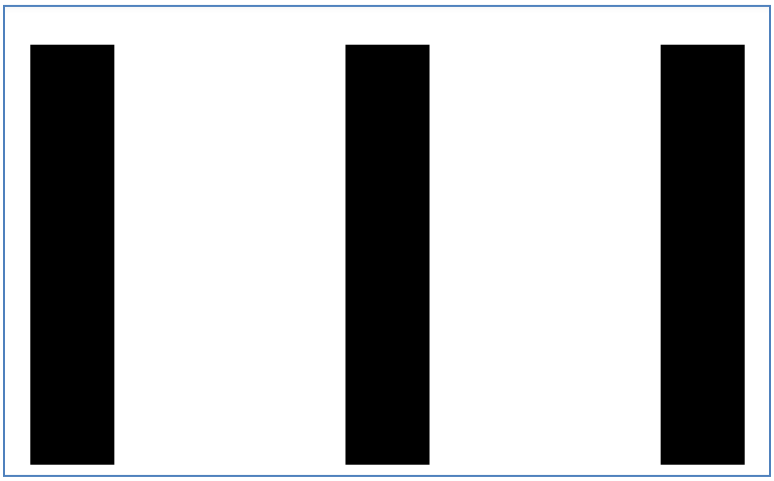

图 2：原始（未旋转）Flash 图像

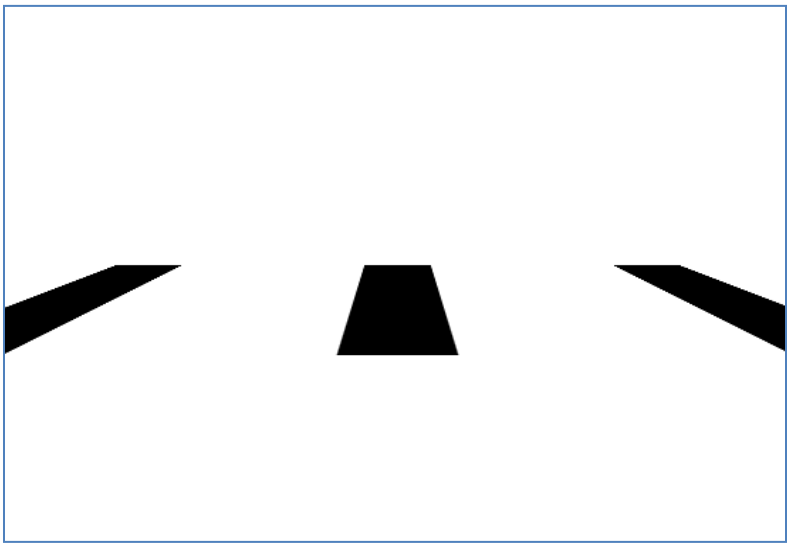

图 3：-80 和默认视角（55 度）的 X 轴旋转

过于低的 FOV 值 (5 以下) 会产生扭曲失真的不良效果. 为了完全避免此效果, 用户应该将对象设置为正射投影。 在 ActionScript 中，这可以通过把 property _perspfov 设置为 -1 来完成。在 C++ 中，它可以通过使用 Direct Access API 的两种不同的方法来完成:

// 1. 将 Perspective FOV 设置为 -1

```
GFx::Value tmpVal;
bool bOK = pMovie->GetVariable(&tmpVal, "root"); //"_root" for AS2
if (bOK)
{
   GFx::Value::DisplayInfo dinfo;
   bOK = tmpVal.GetDisplayInfo(&dinfo);
   if (bOK)
   {
      dinfo.SetPerspFOV(-1);
      tmpVal.SetDisplayInfo(dinfo);
   }
 }
```

// 或者，2. 将透视矩阵设置为某个正交矩阵

```
void MakeOrthogProj(const RectF &visFrameRectInTwips, Matrix4F *matPersp)
{
   const float nearZ = 1;
   const float farZ = 100000;
   float DisplayHeight = fabsf(visFrameRectInTwips.Height());
   float DisplayWidth = fabsf(visFrameRectInTwips.Width());
   matPersp->OrthoRH(DisplayWidth, DisplayHeight, nearZ, farZ);
}
GFx::Value tmpVal;
bool bOK = pMovie->GetVariable(&tmpVal, "root.mc");
if (bOK)
{
   GFx::Value::DisplayInfo dinfo;
   bOK = tmpVal.GetDisplayInfo(&dinfo);
   if (bOK)
   {
      Matrix4F perspMat;
      MakeOrthogProj(PixelsToTwips (pMovie->GetVisibleFrameRect()), &perspMat);
      dinfo.SetProjectionMatrix3D(&perspMat);
      tmpVal.SetDisplayInfo(dinfo);
  }
}
```

它还可以通过直接在电影对象上设置投影 (Projection) 矩阵来完成：

```
Ptr<Movie> pMovie = …
// compute ortho matrix
Render::Matrix4F ortho;
const RectF &visFrameRectInTwips = PixelsToTwips(pMovie->GetVisibleFrameRect());
const float nearZ = 1;
const float farZ = 100000;
ortho.OrthoRH(fabs(visFrameRectInTwips.Width()),  fabs(visFrameRectInTwips.Height()),
                                                    nearZ, farZ);
pMovie->SetProjectionMatrix3D(ortho);
```

5.1 在 AS3 中设置透视

在 ActionScript 3 中，您可以使用 PerspectiveProjection（透视投影）对象来调整一个对象的透视。

Example:

```
import flash.display.Sprite;

var par:Sprite = new Sprite();
par.graphics.beginFill(0xFFCC00);
par.graphics.drawRect(0, 0, 100, 100);
par.x = 100;
par.y = 100;
par.rotationX = 20;

addChild(par);

var chRed:Sprite = new Sprite();
chRed.graphics.beginFill(0xFF0000);
chRed.graphics.drawRect(0, 0, 100, 100);
chRed.x = 50;
chRed.y = 50;
chRed.z = 0;

par.addChild(chRed);

var pp3:PerspectiveProjection=new PerspectiveProjection();
pp3.fieldOfView=120;
par.transform.perspectiveProjection = pp3;
```

6 Stereoscopic 3D

Scaleform 3Di 可用来创建用于游戏和应用程序的 UI，此类游戏和应用程序利用立体 3D 成像技术。一般情况下，立体技术包含通过创建两个图像，彼此稍微偏移，一个图像针对一只眼睛，以此来增强深度错觉。这两个图像代表两个具有相同场景的透视图，使大脑可以感知深度。特殊眼镜，可能的情况下，与立体显示设备同步，可确保将恰当的图象显示到恰当的眼睛。

Scaleform 3Di 具有现成的立体 3D 系统，例如，PC 上使用的 NVIDIA 3D Vision Kit。NVIDIA 的解决方案可在驱动程序级自动工作，因此，无需更改代码，即可启用立体观看。这与其他系统（如游戏控制台）的情况不同。如果是那些系统，应用程序会需要调用 Display 两次，一次针对一只眼睛，并需要设置转换矩阵，以调整适应左右眼的视差位移 (Parallax Shift)。Scaleform 版本支持这一点，并提供一个示例 PS3 立体声播放器调用的 TinyPlayerStereo（可在 Scaleform 子目录中找到）：
Apps \Samples \GFxPlayerTiny \GFxPlayerTinyPS3GcmStereo.cpp

6.1 NVIDIA 3D Vision

6.1.1 3D Vision 安装

要用 NVIDIA 3D Vision Kit 立體渲染 3Di UI,首先要確保已經正確安裝該工具箱和關聯硬體。在 NVIDIA 控制台里的立體三維 3D 部分啟用立體立體三維 3D。可以通過從控制台運行立體三維測試 (Stereoscopic 3D Test) 來驗證安裝。

6.1.2 启动

接下來以全屏模式啟動應用程式或 Scaleform Player。自动化 3D Vision 支援僅適用于全屏應用程式以及需要运用到 Direct3D Scaleform Player,因此,應用程式必須以全屏模式啟動。在 Scaleform Player 运用上， 用户需要应用到 command line argument –f。 根據 NVIDIA 控制台的立體部分的設置,應用程式可以在禁用立體 3D 的情況下啟動。如果 3D 效果沒有立即出現,請按 (CTRL + T) 來切換到立體模式。

6.1.3 汇聚概要

一旦激活立体效果，调整设置就十分重要。屏幕深度的对象应具有零分离和无立体效果。相似地，相对较近和较远的对象必须具有适当的正负分离，以便于在屏幕内外出现。将 3D Vision 深度和汇聚（Convergence）值设置为对于应用程序最佳的状态，然后将它们保存到一个注册表概要之中，以便于将来使用。

正常情况下，在使用 3D Vision 时仅控制深度水平是可能的，要么通过旋转 IR 发射器上的滚轮，要么使用 (CTRL + F3) /(CTRL + F4) 减小或增大深度水平。这是正常操作，因为对于多数游戏来说，汇聚不需要调整，因为游戏概要中已经设置好一切。对于自定义应用程序，或在使用 Scaleform Player 的情况下，必须首先设置正确的汇聚。更改汇聚水平的默认键是 (CTRL + F5) 和 (CTRL + F6)，不过，请注意，默认情况下没有激活这些键。按照下列步骤激活此特性。

要控制汇聚水平，必须通过转到 “*Stereoscopic 3D*”，选择 “*Set up stereoscopic 3D*”，然后打开 “*Set Keyboard Shortcuts*” 来先从 NVIDIA 的控制面板 “启用高级游戏设置” (Enable the advanced in-game settings)。启用高级设置后，可以使用 (CTRL + F5) 和 (CTRL + F6) 来更改汇聚，也可以使用 (CTRL + F7) 组合键来保存选定的自定义设置。保存这些设置使用户下一次运行该应用程序时可以跳过汇聚配置步骤。

调整汇聚时，与深度不同的是，没有显示汇聚水平的图形表示屏幕。因此，更改汇聚时，必须认真监视屏幕上的更改情况。最好的方法是查看一个对象应处屏幕深度（或可能是中间/参考深度）的场景，然后按住 (CTRL + F5)，直到图像分离开始发生变化。这会持续 10 到 15 秒钟，因为汇聚是以非常小的增量调整的，而且起初并不明显。按住 (CTRL + F5) 或 (CTRL + F6)，直到屏幕深度的对象看上去没有图像分离显现（在不戴眼镜观看的情况下）。这将会把汇聚设置为 0 --- 把屏幕深度的对象用作参考点。此外，一旦认为选定的汇聚和深度数量正确，按 (CTRL + F7) 将会把那些设置保存到注册表，以便于应用程序将来使用。

NVIDIA 提供了一个 API，用来有计划地访问和控制立体 3D 行为，同时还提供了一些最佳做法文档，便于在设计立体 3D 游戏时使用。有关更多详细信息，请查阅下列链接：

API
http://developer.nvidia.com/object/nvapi.html.

立体设计
http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/3DVision_Develop_Design_Play_in_3D_Stereo.pdf.

6.2 Scaleform Stereo API

对于控制台，Scaleform 4.1 在 Render::HAL 中引入一个简单的 API，以支持立体 3D。初始化硬件的立体 3D 以及设置帧缓冲区和表面以支持 HDMI 1.4 帧包装的决定权在应用程序。

最初，应用程序初始化其在 Scaleform 渲染器中需要的立体声参数，如第 6.2.1 节所示。

Scaleform 4 具有将视窗设置为并排或上下渲染单独的立体图像的能力。这为需要垂直或水平并列显示立体图像的设备提供了便利。

此功能可通过调用带有适当标志的 **Render::Viewport::SetStereoViewport** 来实现。

下面介绍添加到 SF4 的新的“视窗立体”（Viewport Stereo) 选项标志：

```
// 只用于显示硬件中的立体；使用相同大小缓冲区，但每只眼睛
一半。
View_Stereo_SplitV        = 0x40,
View_Stereo_SplitH        = 0x80,
View_Stereo_AnySplit    = 0xc0,

void SetStereoViewport(unsigned display);
```

6.2.1 Scaleform Stereo 初始化

```
GRenderer::StereoParams s3DInfo;
s3DInfo.DisplayDiagInches = 46;      // 46 inch TV
s3DInfo.DisplayAspectRatio = 16.f / 9.f;
s3DInfo.Distortion = .75;         // 0 to 1
s3DInfo.EyeSeparationCm = 6.4;    // this will default 6.4cm
pRenderHAL->SetStereoParams(s3DInfo);
```

在显示遍历期间，应用程序应调用 Display 两次，一次针对一只眼睛，并在每次调用 Display 之前小心切换到正确的帧缓存区表面。Scaleform 将处理每只眼睛的视界偏移问题。例如：

```
pMovie->Advance(delta);

pRenderer->GetHAL()->SetStereoDisplay(Render::StereoLeft);
pMovie->Display();

pRenderer->GetHAL()->SetStereoDisplay(Render::StereoRight);
pMovie->Display();
```

7 示例文件

实际体验 3Di 的一个绝好方法是观看随 Scaleform SDK 一起提供的一些 3D Flash 示例文件（SWF 和 FLA）。

默认情况下，示例文件位于以下目录： *C:\Program Files (x86)\Scaleform\GFx SDK 4.2\Bin\Data\AS2 or AS3\Samples*

图 4：3DGenerator

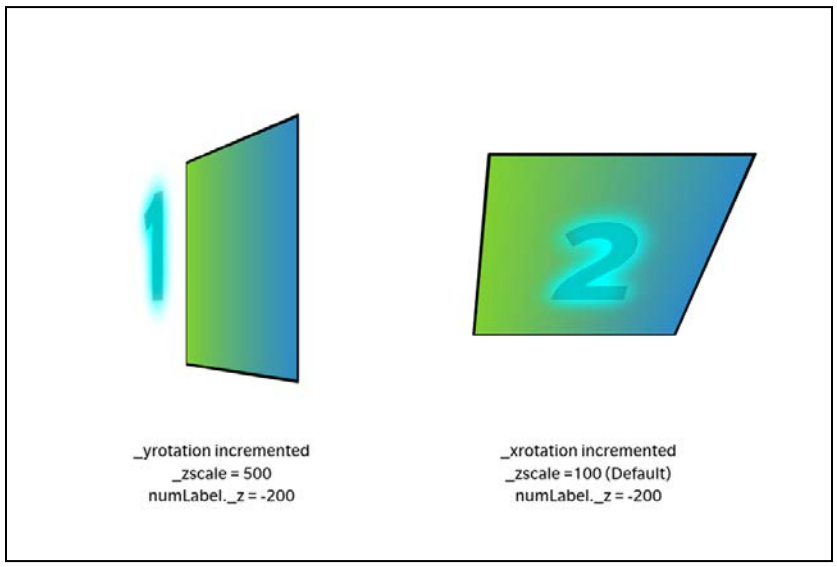

图 5：3D 方块


3D Flash - tilted using ActionScript, no render textures used
Scaleform GFx
3D Window
Send
Change Image
Enter Text
3D GFx Logo
GFx Logo
Scaleform Logo
Wireframe
Scaleform
GFx
Reset
Quit
Scaleform

图 6：3D 视窗