Autodesk®
Scaleform®

Scaleform 3D

本文說明如何使用 Scaleform 3.2 及更高版本的 3D 特性。

作者： Mustafa Thamer

版本： 2.03
上次編輯時間： 2012 年 5 月 9 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

如何聯繫 Autodesk Scaleform：

文檔	Scaleform 3D
位址	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網站	www.scaleform.com
電子郵箱	info@scaleform.com
電話	(301) 446-3200
传真	(301) 446-3199

目录

1 概述

Scaleform® 3Di™ 使開發人員可以將其應用程式 UI 帶到具有新的 3D Flash 渲染和動畫功能的下一個維度。

從 Scaleform 3.2 開始,Scaleform 增加了一組簡單而強大的 ActionScript 擴展,它們能夠提供基本的 3D 功能。儘管 Scaleform 3.2 以 ActionScript 2.0 為基礎,但還是另外提供了一組 3D AS 擴展 --- 與 AS 3.0 中提供的相似。
利用這些新的擴展(_z、_zscale、_xrotation、_yrotation、_matrix3d 和 _perspfov),開發人員可以為功能表、HUD 和遊戲內 UI 創建絕妙的動畫 3D 介面。這些 3D 擴展可應用於任何電影剪輯、按鈕或文字方塊物件。

c + + 中也有同樣的 3D 功能作為 Scaleform 直接訪問 API 的組成部分。新的 API 允許開發人員通過伸縮、旋轉或偏移電影剪輯、按鈕或文字方塊，直接將其進行轉換，保存為 3D GFx::Value。

此外,開發人員可以將自己的 3D 渲染物件或流式視頻添加到 3D Flash 介面,以創造一種獨特的下一代體驗。

2 與 Flash 10 的相似之處

一般來說,Scaleform 提供的 3Di 支援與 Flash 10 中的 3D 非常相似,而且行為也可能相同。3Di 是使用一組內置到 Scaleform 中的基本 AS2 擴展實現的。

2.1 局限

與 Flash 10 中的 3D 支援相似,3Di 也有如下局限:

1. 不執行深度排序,因此,可能無法以正確的順序繪製物件。預設情況下,繪畫順序仍然通過對順序進行分層來指定。
2. 不使用隱面消除 (Back-face culling)。仍然繪製背對觀眾的物件。

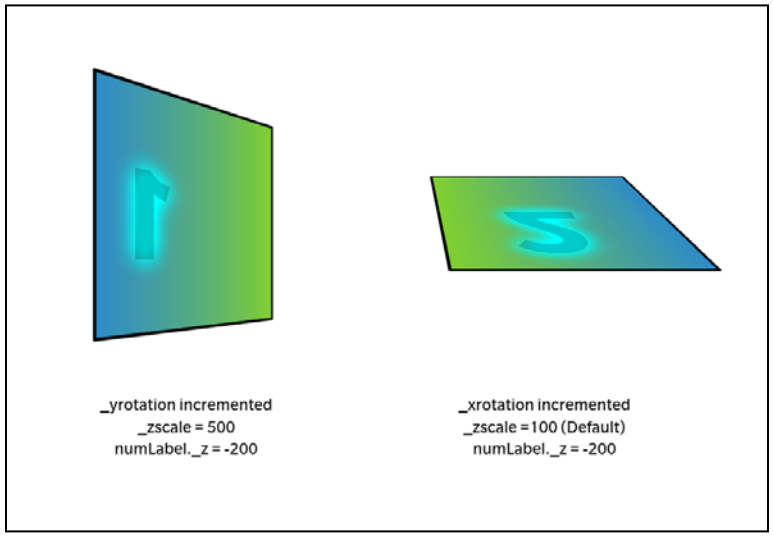

图 1：不选择 (Culled) 或画背面的背面向数目示例

2.2 實現

Scaleform 3Di 使用硬體加速三角形來渲染形狀,而 Flash 10 將 3D 視圖渲染成一個點陣圖,然後將包含該點陣圖的向量矩形用作顯示填充。

當一個物件同時具有 2D 和 3D 屬性時,優先應用 3D 屬性。此細節對於利用自由轉換工具以及 3D 操作旋轉一個物件時獲得預期結果非常重要。Flash 10 具有相同的行為。

3 以 3D 顯示 Flash

使用 Scaleform 3.2,有兩種不同的方法可用來以 3D 形式顯示 Flash 內容。

3.1 渲染到紋理

第一個選項是使用傳統方法將 Flash 渲染成一種紋理,然後將該渲染紋理作為一個地圖應用到場景中的一個 3D 表面上。在 Scaleform 3.2 之前,這是開發人員可以用 3D 視覺化 Flash 的唯一方法。

"渲染到紋理"(Render to Texture) 方法比 3Di 具有一個明顯的優勢。由於 Flash 內容以紋理方式映射到一個 3D 物件上,因而可以充分利用 3D 引擎的功能,並與 3D 世界的其餘部分進行無縫混合。這樣就可以進行正確的 Z(深度)排序、混合/透明以及剪輯(而不管是否想要對 UI 進行剪輯)。

另一方面,這種方法也存在若干與性能和記憶體相關的缺點,因為渲染紋理必需附加記憶體。而且,請注意,任何 Flash 交互必須通過遊戲進行處理(通過以手動方式反向映射滑鼠座標並傳遞到 Scaleform)。最後,由於這種方法是基於紋理的,所以 Flash 內容將精確地顯示,因而不會偏移具體電影剪輯或從彼此旋轉。有關此方法的示例,請參閱"SWF 到紋理"(SWF to Texture) 演示,該演示可在 Scaleform SDK 流覽器中找到。

3.2 3Di

第二個選項(Scaleform 3.2 中的新特性)是使用 3Di --- 從 ActionScript 或 Direct Access API (C++)。與使用渲染紋理方法相比,3Di 具有許多優勢。首先,不需要任何附加記憶體。其次,單個電影剪輯可以有其自己的嵌套式轉換 – 每個都可以唯一地偏移或旋轉,因而整個電影看起來不單調。最後,所有輸入和交互都是由 Scaleform 自動處理的。應用程式只是像平常一樣把輸入事件傳遞到Scaleform。

不過,3Di 確實也有一些局限性。特別是,由於它是由預設 Scaleform 渲染器處理的,並被視為一個 UI,因而不執行任何深度排序或隱面消除。因此,當 UI 物件在其他物件後面旋轉時,繪製的順序可能不正確。而且,混合 UI 元素與其他 3D 遊戲物件無縫混合可能比較棘手,因為 UI 物件應匹配相同的相機、照明、後處理以及在用的其他圖形效果。此外,許多 UI 元素是透明的,並要求特定的繪畫混合順序,而這一點是需要加以重視的。

4 3Di API

4.1 ActionScript 2 API

憑藉 3Di,Flash 物件現在可以擁有另一個維度。增加第三個維度使使用者可以查看物件向視點前進或後退。當物件移離相機時,它們由於透視投影而看起來較小。

預設情況下,物件是二維的,而且不使用或計算任何 3D 屬性。不過,一旦設置一個 3D 屬性(x 或 y 旋轉、z 翻譯或伸縮或 3d 矩陣),該物件就成為三維。在這一點上,自動創建一個 3D 轉換矩陣,並指定給該物件。通過將該物件的 3d 矩陣設置為"空"(Null)(即,在 ActionScript 中,foo._matrix3d = null),可以使該物件返回到一種完全的 2D 狀態。僅將旋轉和 z 值設置為 0 將不會消除 3D 轉換。

如上所述,3D 旋轉允許圍繞 3 個軸(x、y 或 z 軸)中的任何一軸旋轉,這與只圍繞 Z 軸旋轉的 2D 情形相反。相似地,3D 伸縮增加一個額外的用於伸縮的軸,包含有 _zscale ActionScript 擴展。

Scaleform 中預設的 3D 坐標系和相機與 Flash 10 中的相同。它是一個右手坐標系,其中 +X 向右,+Y 向下,而 +Z 指向螢幕中(離開取景器)。預設相機面朝下對著 +Z 軸,FOV 角為 55 度。

可以將 3D 屬性應用到下列 Flash 物件類型：

- 電影剪輯
- 文字方塊
- 按鈕

為 3Di 增加了下列 ActionScript 擴展：

- **_z** - 設置物件的 Z 座標(深度),預設值為 0。
- **_zscale** - 將 Z 向的物件伸縮設置一個百分比,預設值為 100。
- **_xrotation** - 設置物件圍繞 X(橫向)軸旋轉,預設值為 0。
- **_yrotation** - 設置物件圍繞 Y(縱向)軸旋轉,預設值為 0。
- **_matrix3d** - 使用一組 16 個浮點數(4 x 4 矩陣)設置物件的完整 3D 轉換。如果此值設置為 NULL,就會刪除所有 3D 轉換,而且物件將會變為 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。

使用這些新擴展需要在 ActionScript 中把全域變數 gfxExtensions 設置為 True(真)。

如果你計畫修改一個電影剪輯實例的旋轉,務請注意註冊點的位置。預設情況下,當您創建一個電影剪輯符號時,註冊點設置為左上角。如果您使用 _xrotation 或 _yrotation 將一個 3D 旋轉應用到電影剪輯實例,該物件就會圍繞著註冊點進行旋轉。如果需要,可以將註冊點設置為符號的中心。如果想要使用 3Di 將旋轉應用到物件,務必謹慎設置註冊點。

示例 1

圍繞 Y 軸旋轉一個電影剪輯 45 度：

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

示例 2

沿著 X 軸以及一個 Z 標度連續旋轉：

```
_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

使用應用到根的 3D 翻譯矩陣的 3D 轉換：

```
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 來自 C++ 的 3Di

4.2.1 Matrix4x4

添加了一個名為 Matrix4x4 的新類以表示一個 4x4 行主要矩陣。此類具有用於創建和操縱各種 3D 矩陣類型(包括整個世界轉換以及視圖和投影)的所有適當函數。有關詳細資訊,請參閱 Render_Matrix4x4.h。

4.2.2 Direct Access

Direct Access 介面(經由 GFx::Value 類)已擴展為可支援 3Di。現在可以利用新的 API 來設置或查詢 3D 屬性。這些函數類似于 3Di ActionScript 擴展。

有關更多詳細資訊,請參閱 GFx_Player.h 中的 GFx::Value::DisplayInfo 類。

```
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 類介面添加了新的調用,以便於設置用於渲染 3D 電影的"視圖"(View) 和"透視"(Perspective) 矩陣。儘管可以在單個顯示物件上設置透視,但在電影根部設置往往方便一些,以便於立即影響到所有物件。

GFx_Player.h

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

4.2.4 示例:使用 Direct Access API 更改透視視野

```
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 透視

上面的示例顯示如何使用 Direct Access API 在電影上設置視野 (FOV)。透視和視圖設置可應用到電影根部或個別顯示物件上。如果一個物件返回的透視 FOV 值為 0,則它將從其父物件那裡繼承 FOV 值。

透視使距離觀眾較近的物件看起來更大,較遠的物件則看起來更小。通過更改視野值,可以控制此效果的強度。此值越大,應用到沿著 z 軸移動的顯示物件的失真度越強。FOV 值小的結果是伸縮度非常小,而此值大則使伸縮度大。此值必須大於 0 而小於 180,其中值為 1,則幾乎不存在透視失真,而此值為 179 時,可創造一種失真的魚眼透鏡效果。

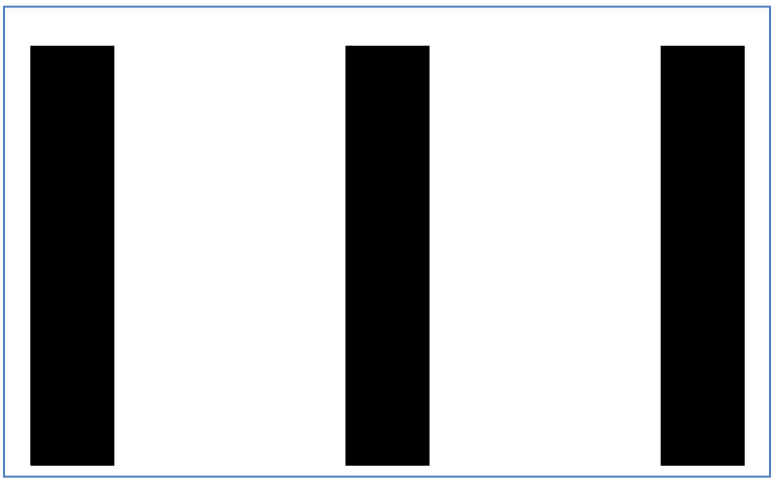

图 2：原始（未旋转）Flash 图像

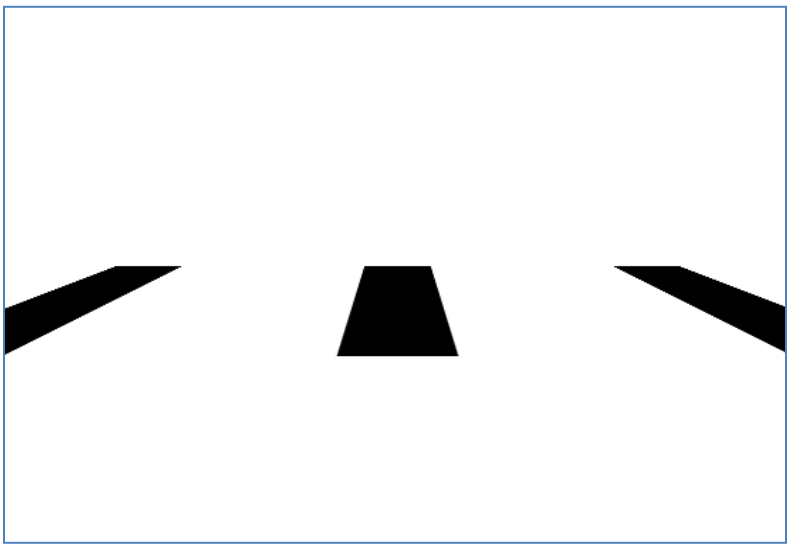

图 3：-80 和默认视角（55 度）的 X 轴旋转

過於低的 FOV 值 (5 以下) 會產生扭曲失真的不良效果. 為了完全避免此效果, 使用者應該將物件設置為正射投影。 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 中設置透視

在 ActionScript 3 中,您可以使用 PerspectiveProjection(透視投影)物件來調整一個物件的透視。

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 可用來創建用於遊戲和應用程式的 UI,此類遊戲和應用程式利用立體 3D 成像技術。一般情況下,立體技術包含通過創建兩個圖像,彼此稍微偏移,一個圖像針對一隻眼睛,以此來增強深度錯覺。這兩個圖像代表兩個具有相同場景的透視圖,使大腦可以感知深度。特殊眼鏡,可能的情況下,與立體顯示裝置同步,可確保將恰當的圖像顯示到恰當的眼睛。

Scaleform 3Di 具有現成的立體 3D 系統,例如,PC 上使用的 NVIDIA 3D Vision Kit。NVIDIA 的解決方案可在驅動程式級自動工作,因此,無需更改代碼,即可啟用立體觀看。這與其他系統(如遊戲主控台)的情況不同。如果是那些系統,應用程式會需要調用 Display 兩次,一次針對一隻眼睛,並需要設置轉換矩陣,以調整適應左右眼的視差位移 (Parallax Shift)。Scaleform 4 版本支援這一點,並提供一個示例 PS3 身歷聲播放機調用的 TinyPlayerStereo(可在 Scaleform 子目錄中找到):
Apps \Samples \GFxPlayerTiny \GFxPlayerTinyPS3GcmStereo.cpp

6.1 NVIDIA 3D Vision

6.1.1 3D Vision 安裝

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 主控台裡的立體三維 3D 部分啟用立體立體三維 3D。可以通過從主控台運行立體三維測試 (Stereoscopic 3D Test) 來驗證安裝。

6.1.2 啟動

接下來以全屏模式啟動應用程式或 Scaleform Player。自動化 3D Vision 支援僅適用於全屏應用程式以及需要運用到 Direct3D Scaleform Player,因此,應用程式必須以全屏模式啟動。在 Scaleform Player 運用上,使用者需要應用到 command line argument –f。 根據 NVIDIA 主控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 彙聚概要

一旦啟動立體效果,調整設置就十分重要。螢幕深度的物件應具有零分離和無立體效果。相似地,相對較近和較遠的物件必須具有適當的正負分離,以便於在螢幕內外出現。將 3D Vision 深度和彙聚 (Convergence) 值設置為對於應用程式最佳的狀態,然後將它們保存到一個註冊表概要之中,以便於將來使用。

正常情況下,在使用 3D Vision 時僅控制深度水準是可能的,要麼通過旋轉 IR 發射器上的滾輪,要麼使用 (CTRL + F3) /(CTRL + F4) 減小或增大深度水準。這是正常操作,因為對於多數遊戲來說,彙聚不需要調整,因為遊戲概要中已經設置好一切。對於自訂應用程式,或在使用 Scaleform Player 的情況下,必須首先設置正確的彙聚。更改彙聚水準的預設鍵是 (CTRL + F5) 和 (CTRL + F6),不過,請注意,預設情況下沒有啟動這些鍵。按照下列步驟啟動此特性。

要控制彙聚水準,必須通過轉到 “Stereoscopic 3D”,選擇 “Set up stereoscopic 3D”,然後打開 “Set Keyboard Shortcuts” 來先從 NVIDIA 的控制台“啟用高級遊戲設置”(Enable the advanced in-game settings)。啟用高級設置後,可以使用 (CTRL + F5) 和 (CTRL + F6) 來更改彙聚,也可以使用 (CTRL + F7) 複合鍵來保存選定的自訂設置。保存這些設置使使用者下一次運行該應用程式時可以跳過彙聚配置步驟。

調整彙聚時,與深度不同的是,沒有顯示彙聚水準的圖形表示螢幕。因此,更改彙聚時,必須認真監視螢幕上的更改情況。最好的方法是查看一個物件應處螢幕深度(或可能是中間/參考深度)的場景,然後按住 (CTRL + F5),直到圖像分離開始發生變化。這會持續 10 到 15 秒鐘,因為彙聚是以非常小的增量調整的,而且起初並不明顯。按住 (CTRL + F5) 或 (CTRL + F6),直到螢幕深度的物件看上去沒有圖像分離顯現(在不戴眼鏡觀看的情況下)。這將會把彙聚設置為 0 --- 把螢幕深度的物件用作參考點。此外,一旦認為選定的彙聚和深度數量正確,按 (CTRL + F7) 將會把那些設置保存到註冊表,以便於應用程式將來使用。

NVIDIA 提供了一個 API,用來有計劃地訪問和控制立體 3D 行為,同時還提供了一些最佳做法文檔,便於在設計立體 3D 遊戲時使用。有關更多詳細資訊,請查閱下列連結：

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

立體設計
http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/3DVision_Develop_Design_Play_in_3D_Stereo.pdf.

6.2 Scaleform Stereo API

對於主控台,Scaleform 4.1 在 Render::HAL 中引入一個簡單的 API,以支援立體 3D。初始化硬體的立體 3D 以及設置框架緩衝區和表面以支援 HDMI 1.4 幀包裝的決定權在應用程式。

最初,應用程式初始化其在 Scaleform 渲染器中需要的身歷聲參數,如第 6.2.1 節所示。

Scaleform 4 具有將視窗設置為並排或上下渲染單獨的立體圖像的能力。這為需要垂直或水準並列顯示立體圖像的設備提供了便利。

此功能可通過調用帶有適當標誌的 Render::Viewport::SetStereoViewport 來實現。

下面介紹添加到 SF4 的新的「視窗立體」(Viewport Stereo) 選項標誌：

```
//只用于顯示硬體中的立體;使用相同大小緩衝區,但每隻眼睛 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

```
Render::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);
```

在顯示遍歷期間,應用程式應調用 Display 兩次,一次針對一隻眼睛,並在每次調用 Display 之前小心切換到正確的幀緩存區表面。Scaleform 將處理每隻眼睛的視界偏移問題。例如：

```
pMovie->Advance(delta);

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

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

7 示例文件

實際體驗 3Di 的一個絕好方法是觀看隨 Scaleform SDK 一起提供的一些 3D Flash 示例檔(SWF 和 FLA)。

預設情況下,示例檔位於以下目錄： *C:\Program Files (x86)\Scaleform\GFx SDK 4.3\Bin\Data\AS2 or AS3\Samples*

图 4：3DGenerator

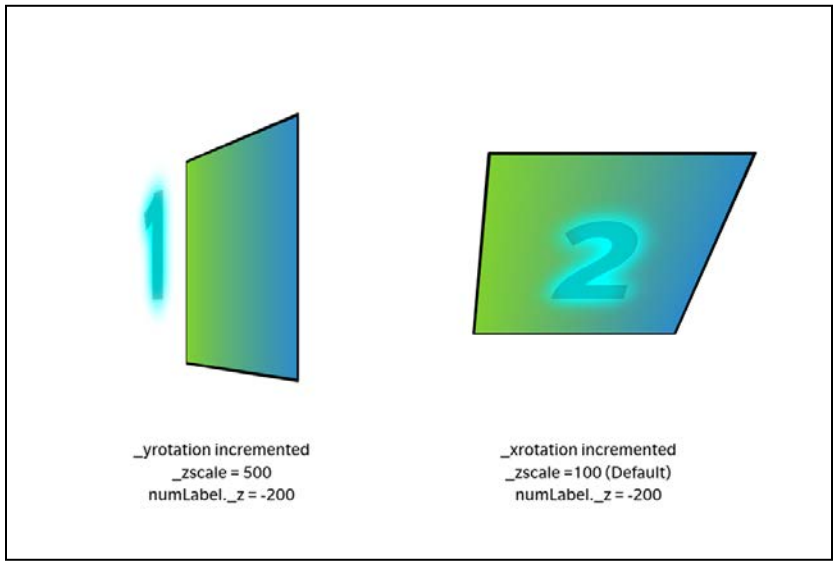

图 5：3D 方块


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

图 6：3D 视窗