Autodesk®
Scaleform®

Scaleform 3D

このドキュメントでは Scaleform 3.2 以上でサポートされている 3D 機能の使い方について説明しています。

作者 Mustafa Thamer
バージョン 2.03
最終更新日 2012 年 5 月 9 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Disclaimer

Autodesk Scaleform の連絡先：

ドキュメント名	Scaleform 3D
住所	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
ウェブサイト	www.scaleform.com
Email	info@scaleform.com
電話	+1 (301) 446-3200
Fax	+1 (301) 446-3199

目 次

1 概要

Scaleform® 3Di™ は、新しい 3D Flash レンダリングとアニメーション機能によって、アプリケーション UI を新たな次元へといざなうシステムです。

Scaleform は、Scaleform 3.2 の世代から、単純ながらも強力な ActionScript エクステンションを追加し、基本的な 3D 機能をサポートしてきました。Scaleform 3.2 は ActionScript 2.0 をベースにしていますが、AS 3.0 で利用可能なものと同様の 3D AS エクステンションがすでに提供されています。

新たに追加された拡張機能（_z、_zscale、_xrotation、_yrotation、_matrix3d、_perspfov）を使うことで、メニュー、HUD、およびゲーム中の UI を対象としたアニメーション付きの素晴らしい 3D インタフェースを作成することができます。これら 3D エクステンションは、すべてのムービークリップ、ボタン、およびテキストフィールド・オブジェクトに適用可能です。

同様の 3D 機能は Scaleform Direct Access API の一部として C++からも利用可能です。新 API を用いて拡大／縮小、回転、オフセットを与えることで、3D 内に GFx::Value として保持されるムービークリップ、ボタン、およびテキストフィールドを直接変換することができます。

さらに、開発した 3D レンダリング・オブジェクトまたはストリーミング・ビデオを 3D Flash インタフェースに追加して、次世代のエクスペリエンスを実現することも可能です。

2 Flash 10 との類似性

一般に Scaleform が提供する 3Di 機能は Flash 10 の 3D 機能と類似しており、同じように動作することが見込まれます。3Di は Scaleform 内に組み込まれている AS2 エクステンションの基本セットを使って実装されています。

2.1 制約事項

Flash 10 の 3D 機能と同じように、3Di には以下のような制約があります。

1. デプス・ソートは実行されないため、オブジェクトは正しい順番では描画されません。デフォルトでは、描画順はレイヤー順として指定されます。
2. 裏面（バックフェース）カリングは使用していません。観測者から見えないオブジェクトでも描画が行われます。

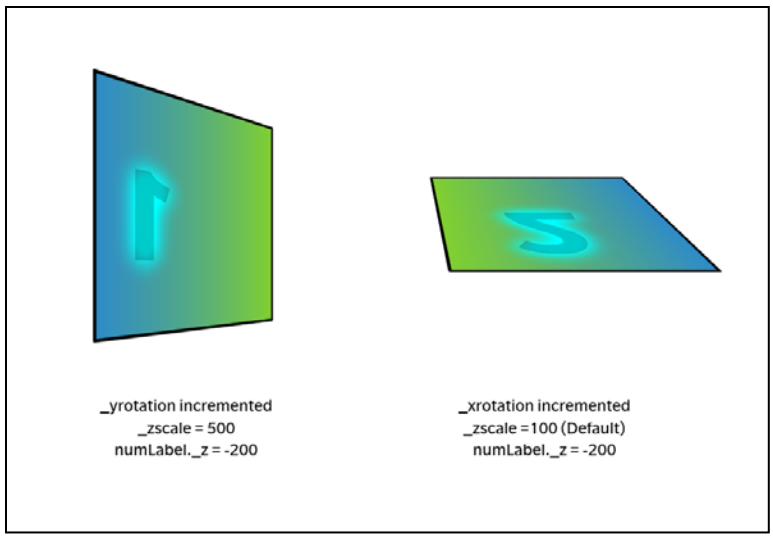

図 1: カリングされずにオブジェクトの後ろ側で描画される裏向き数字の例

2.2 実装

Scaleform 3Di はハードウェア・アクセラレータのトライアングルを使用して形状のレンダリングを行いますが、Flash 10 は 3D ビューをビットマップにレンダリングし、次に、表示のフィルとしてそのビットマップをベクター・レクトアングルに使用します。

オブジェクトが 2D と 3D の両方のプロパティを有する場合、2D プロパティの前に 3D プロパティが適用されます。自由変換ツールを使ったオブジェクトの回転や 3D 操作において期待した結果を得るには、このような詳しい動作を理解しておくことが重要です。Flash 10 の挙動も同じです。

3 3D での Flash の表示

Scaleform 3.2を使ってFlashコンテンツを3Dで表示するには、次の2つの方法のいずれかを使います。

3.1 レンダーからテクスチャへの変換

ひとつめの方法は、Flashをテクスチャにレンダリングし、そのレンダー・テクスチャをマップとしてシーン内の3D面に適用する従来の手法です。Scaleform 3.2よりも前のバージョンでFlashを3Dとしてビジュアライズするには、この方法しかありませんでした。

レンダーをテクスチャに変換する方法は3Diを使う方法に比べてメリットがあります。Flashコンテンツは3Dオブジェクト上にテクスチャ・マッピングされるため、3Dエンジンの能力を全面的に活用できるとともに、ほかの3D空間と自然なつながりが得られます。すなわち、適切なZ（デプス）ソーティング、ブレンディング／トランスペアレンシ、およびクリッピング（UIをクリップしようとする意図に関わらず）が得られます。

一方で、レンダリングしたテクスチャのために別途メモリが必要となるため、性能とメモリに関してはいくつかのデメリットがあります。また、すべてのFlashインタラクティブ性はゲーム側で扱わなければなりません（マウス座標をマニュアル作業にて逆マッピングし、それらをScaleformに渡すことによって）。また、この方法はテクスチャを使用しているため、Flashコンテンツは平面的に表示され、ゆえに個々のムービークリップを互いにずらしたり回転させることはできません。この方法の例としては、Scaleform SDK Browser内にある「SWF to Texture」デモを参照してください。

3.2 3Di

Scaleform 3.2から追加されたもうひとつの方法が3Diで、ActionScriptまたはDirectAccess API（C++）から使います。3Diにはレンダー・テクスチャ方式に比べて多くのメリットがあります。まず、追加メモリ領域を必要としません。また、各ムービークリップに対して個別に変換を適用することが可能であり、それぞれを個別にずらしたり回転できるため、ムービー全体が平面的に見えません。さらに、すべての入力とインタラクティブ性は自動的にScaleformによって処理されますので、アプリケーションは単純に入力イベントを通常と同じようにScaleformに渡すだけです。

ただし、3Diにはいくつかの制約があります。とくに、デフォルトScaleformレンダラーによって処理され、UIとして扱われるため、デプス・ソーティングあるいは裏面カリングは実行されません。そのため、UIオブジェクトを他のオブジェクトの後ろで回転させたとき、正しい順番で描画されない可能性があり得ます。また、同じカメラ、ライティング、後処理、あるいは他のグラフィカルなエフェクトとUIオブジェクトとを整合させなければなりませんが、UIエレメントと他の3Dゲーム・オブジェクトとを自然に混在させることが難しい場合があります。また、多くのUIエレメントはトランスペアレントであり、ブレンディングのために特定の描画順を必要とするため、なんらかの対応が必要です。

4 3Di API

4.1 ActionScript 2 API

3Di を使うと Flash オブジェクトには別の次元が与えられることになります。3 番目の次元が追加されることで、ビューポイントに対して前方または後方に移動するオブジェクトをユーザーに対して表示できます。オブジェクトがカメラから離れると、透視投影の原理によってオブジェクトは小さくなります。

デフォルトではオブジェクトは 2 次元であり、3D プロパティが使われたり計算されることはありません。ただし、一度でも 3D プロパティを設定すると（x 回転または y 回転、z トランスレーションまたはスケールまたは 3D マトリックス）、オブジェクトは 3 次元に変わります。このとき、3D 変換マトリックスが自動的に生成されオブジェクトに割り当てられます。3D マトリクッスを null（たとえば ActionScript にて foo._matrix3d = null）に設定すると、オブジェクトは 2D 状態に戻ります。回転および z 値のみを 0 に設定しても 3D 変換は削除されません。

2D のときは z 軸の回転しかできませんでしたが、上述のとおり 3D 回転がサポートされたため、x、y、z の 3 軸のいずれでも回転させることが可能になりました。同様に、_zscale ActionScript エクステンションを使うことで、3D スケールも新しい軸を対象に行うことができるようになりました。

デフォルトの 3D 座標系と Scaleform 内のカメラは Flash 10 と同じです。時計回り系になっていて、+X で右、+Y で下、+Z で奥行き方向（観測者から離れる）になります。デフォルトのカメラは FOV 角度 55°にて+Z 軸を見下ろしています。

3D プロパティは次のタイプの Flash オブジェクトに適用可能です。

- Movie clips ムービークリップ
- Text fields テキストフィールド
- Buttons ボタン

3Di では次の ActionScript エクステンションが追加されました。

- **_z** - オブジェクトの Z 座標（デプス）を設定します。デフォルト値は 0 です。
- **_zscale** - Z 方向のオブジェクトのスケール（倍率）をパーセントを単位に設定します。デフォルト値は 100 です。
- **_xrotation** - X 軸（水平）を中心としたオブジェクトの回転を設定します。デフォルト値は 0 です。
- **_yrotation** - Y 軸（垂直）を中心としたオブジェクトの回転を設定します。デフォルト値は 0 です。
- **_matrix3d** - 16 個の浮動小数点のアレイ（4×4 のマトリックス）を使ってオブジェクトの完全 3D 変換を設定します。この値を NULL にセットするとすべての 3D 変換は削除されオブジェクトは 2D に戻ります。
- **_perspfov** -オブジェクト上の Field Of View パースペクティブ角を設定します。有効な角度は 0 より大きく 180 未満です。-1 を設定するとパースペクティブを無効にして正投影ビューを使用します。設定しない場合、オブジェクトはデフォルトが 55°に設定されているルートの FOV 角を継承します。

これら新しい拡張機能を使用するには、ActionScript にてグローバル変数 gfxExtensions を true に設定する必要があります。

ムービークリップ・インスタンスの回転を変更したい場合は登録ポイントの位置を確認してください。デフォルトでは、ムービークリップ・シンボルを作成したとき、登録ポイントは左上隅に設定されます。ムービークリップ・インスタンスに_xrotation または_yrotation を使って 3D 回転を適用すると、オブジェクトは登録ポイントを中心に回転します。登録ポイントは、必要に応じて、シンボルの中心に設定することができます。3Di を使ってオブジェクトに回転を適用したい場合は、登録ポイントを注意深く設定してください。

例 1
Y 軸を中心にムービークリップを 45 度回転させる。

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

例 2
X 軸を中心に連続的に回転させるとともに、Z スケールを適用する：

```
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);
}
```

例 3
ルートに適用される 3D トランスレーション・マトリックスを使って 3D 変換を行う。

```
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 C++から 3Di を利用する

4.2.1Matrix4x4

4×4 の列優先（row major）マトリックスを表現する目的で、Matrix4x4 という名称の新しいクラスが追加されました。このクラスは、汎用ワールド変換のほかビューやプロジェクションなど、さまざまなタイプの 3D マトリックスを生成および操作するために必要なすべての関数を備えています。詳細は Render_Matrix4x4.h を参照してください。

4.2.2Direct Access

3Di サポートを目的に、（GFx::Value クラスを介した）Direct Access インタフェースが拡張されています。新しい API を使って 3D プロパティの設定またはクエリが可能です。これら関数は 3Di ActionScript エクステンションに類似しています。

詳細は GFx_Player.h 内の GFx::Value::DisplayInfo クラスを参照してください。

```
   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
```

4.2.3Render::TreeNode

3D でのムービーのレンダリングに必要なビューとパースペクティブ・マトリックスを設定する新しいコールが、Render::TreeNode クラス・インタフェースに追加されました。パースペクティブは個々の表示オブジェクトに対して設定できますが、ムービーのルートに対して設定するとすべてのオブジェクトに効果が及びます。

GFx_Player.h

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

4.2.4例：Direct Access API を使ったパースペクティブ FOV の変更

```
   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 パースペクティブ

前記のサンプルコードは、Direct Access API を使って、ムービー上にフィールド・オブ・ビュー（FOV）を設定する方法を示したものです。パースペクティブとビューの設定は、ムービーのルートまたは個々のディスプレイ・オブジェクトに適用されます。オブジェクトがパースペクティブ FOV 値として 0 を返したときは、その親から FOV 値を継承することになります。

パースペクティブによって、閲覧者に近いオブジェクトは大きく、閲覧者から遠いオブジェクトは小さくなります。フィールド・オブ・ビュー値を変えることで、この効果の度合いをコントロールできます。値を大きくすると z 軸方向に沿ったディスプレイ・オブジェクトに強い歪みが適用されます。小さな FOV 値を与えると遠近感は小さくなり、大きな FOV 値を与えると遠近感は強くなります。値は 0 よりも大きく、180 よりも小さくなければなりません。1 はほとんどパースペクティブ歪みがなく、179 は魚眼レンズのような歪みを生み出します。

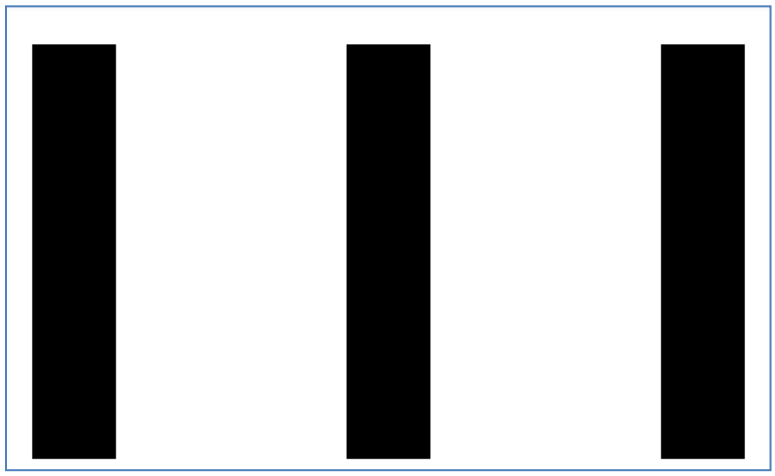

図 2: オリジナルの（回転していない）Flash イメージ

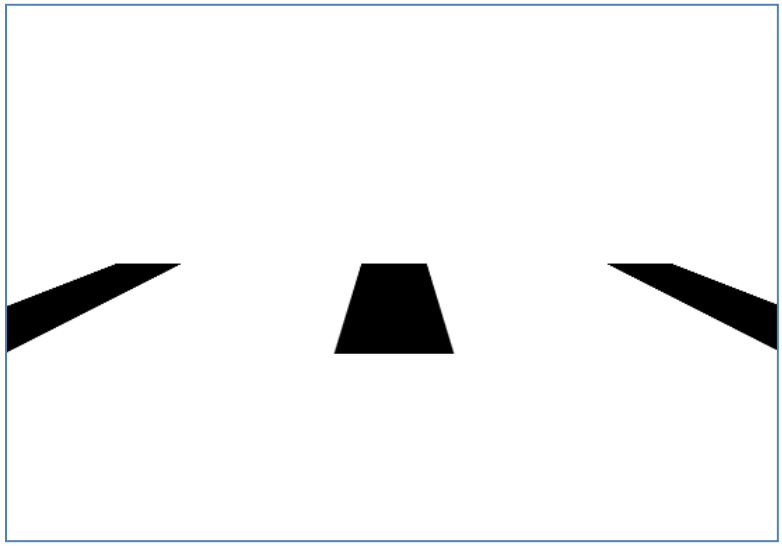

図 3: X 方向に-80 の回転を与え、パースペクティブはデフォルト（55°）

ローＦＯＶバリュー（下記５をご参照）は歪んだエフェクトという結果になることがあります。パースペクティブなエフェクトを完全に除外するためには orthographic projection を使用しオブジェクトにセットしたほうが良いかもしれません。ActionScript では property _perspfov を-1 に設定します。C++では Direct Access API を使ったふたつの方法があります。

// 1. Perspective FOV を-1 に設定する

```
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);
   }
 }
```

// または、2. パースペクティブ マトリックスを正投影マトリックスに設定する

```
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);
  }
}
```

ムービー オブジェクト上の Projection マトリックスを直接設定してもかまいません。

```
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 AS3 でのパースペクティブの設定

ActionScript 3 では PerspectiveProjection オブジェクトを使用してオブジェクトのパースペクティブを調節できます。

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 3D での立体画像

Scaleform 3Di を使うと、ステレオ 3D イメージを利用したゲームやアプリケーション用の UI を作成することができます。一般にステレオ技法は、それぞれが片方の眼に相当するわずかに離れた 2 つのイメージで生じる奥行きの錯覚を利用しています。これら 2 つのイメージは同じシーンを対象に異なるパースペクティブを表現したもので、脳に奥行きを知覚させる働きがあります。ステレオ・ディスプレイデバイスと、デバイスと同期した特殊なメガネを使うことで、適切な眼に適切なイメージが届けられます。

Scaleform 3Di は、パソコン上で動作する NVIDIA の 3D Vision Kit のような市販のステレオ 3D システム上でも適切に動作します。NVIDIA のソリューションは、ドライバレベルで自動的に動作するため、コードを変更することなくステレオ機能を有効にすることができます。このような特徴はゲームコンソールなど他のシステムでは見られません。それらシステムでは、アプリケーションは Scaleform Display をそれぞれの眼に対して 1 回ずつ合計で 2 回コールするとともに、左目と右目の視差に合致するように変換マトリックスを設定する必要があるでしょう。Scaleform はこの機能をバージョン 4.0 でサポートしています。PS3 ステレオプレーヤーのサンプルは、Scaleform の *Apps\Samples\GFxPlayerTiny\GFxPlayerTinyPS3GcmStereo.cpp* サブディレクトリに収録した TinyPlayerStereo にて提供しています。

6.1 NVIDIA 3D Vision

6.1.13D Vision のセットアップ

NVIDIA の 3D Vision Kit を使って 3Di の UI をステレオでレンダリングする場合、まず最初にキットと関連ハードウェアが適切にインストールされていることを確認してください。NVIDIA のコントロール パネルを開き Stereoscopic 3 D セクションの’Stereoscopic を有効にする’をクリックします。セットアップを検証するには、NVIDIA コントロールパネルから Stereoscopic 3D Test を起動します。

6.1.2起動

次に、アプリケーションまたは D3D9 Scaleform Player をフルスクリーン・モードで起動します。オートマチック 3D Vision はフルスクリーン・アプリケーションしかサポートしていませんし Direct3D Scaleform Player が必要なので、アプリケーションはフルスクリーン・モードで起動しなければなりません。Scaleform Player の場合はコマンドライン argument –f のパッシングが必要です。NVIDIA コントロールパネルのステレオ・セクションの設定によっては、アプリケーションはステレオ 3D が無効の状態で起動する場合があります。3D の効果が即座に見られない場合は、CTRL＋T を入力してステレオ・モードにトグルしてください。

6.1.3コンバージェンス・プロファイル

ステレオエフェクトを有効にした場合は設定の調整が重要になります。画面奥にあるオブジェクトは距離が離れていてはならずステレオ効果は必要ありません。同様に、お互いが近接していて、かつ、

遠くにあるオブジェクトも、画面の外側または内側に現れるように、適切な正および負のセパレーションを有しなければなりません。3D Vision のデプス（眼の間隔）とコンバージェンス（焦点距離）をアプリケーションにとって最適な値に設定するとともに、将来の使用に備えてレジストリ・プロファイルに保存してください。

3D Vision を使用した場合は一般に、IR トランスミッタのスクロールホイールを回すか CTRL＋F3 キー／CTRL＋F4 キーを押すことで、デプス・レベルの増減のみが可能です。ほとんどのゲームではゲームのプロファイルによってすべてが設定されているため、コンバージェンスは調整の必要がなく、デプス・レベルしか調整できなくても問題ありません。カスタム・アプリケーションあるいは Scaleform Player を使った場合は、正しいコンバージェンスを最初に設定する必要があります。デフォルトでコンバージェンス・レベルを変更するキーの組み合わせは CTRL＋F5 と CTRL＋F6 です。ただし、デフォルトではこれらのショートカットキーは有効にはなっていません。この機能を有効にするには以下の手順に従ってください。

コンバージェンス・レベルを制御するには、NVIDIA コントロールパネルの「*Stereoscopic 3D*」から「*Set up stereoscopic 3D*」を選び、「*Set Keyboard Shortcuts*」を開いて、「Enable the advanced in-game settings」を有効にします。この設定を有効にすると、CTRL＋F5 と CTRL＋F6 からコンバージェンスを変更できるとともに、選択したカスタム設定を CTRL＋F7 で保存することができます。設定を保存しておけば、アプリケーションを次回起動したときに、ユーザーはコンバージェンスの設定をする必要がありません。

コンバージェンスを調整しても、デプスの調整とは違って、コンバージェンス・レベルを表す情報は画面上に何も表示されません。そのため、コンバージェンスを変更するときは、画面上の変化を注意深く観察する必要があります。画面の奥（あるいは中央／基準デプス）にオブジェクトがあるシーンで、イメージのセパレーションが起こるまで CTRL＋F5 キーを押下するのが適切な方法です。コンバージェンスの調整はきわめて小さな変化で最初はほとんど気が付かないため、この操作には 10 秒から 15 秒程度はかかるでしょう。（メガネなしで見たときに）画面奥のオブジェクトのセパレーションがなくなるまで CTRL＋F5 と CTRL＋F6 を押下します。以上の操作では、画面奥のオブジェクトを基準点とし、そのオブジェクトのコンバージェンスをゼロに設定します。ここでも同じように、適正とみなされるコンバージェンスとデプス量が得られた段階で、CTRL＋F7 を押して次回の利用に備えて設定を保存してください。

NVIDIA はステレオ 3D の動作をプログラムからアクセスおよびコントロールする API を公開しています。また、ステレオ 3D のゲーム設計に関するベスト・プラクティスをまとめたドキュメントを公開しています。詳細は以下のリンクを参照してください。

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

ステレオでの設計
http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/3DVision_Develop_Design_Play_in_3D_Stereo.pdf.

6.2 Scaleform ステレオ API

コンソールに対して、Scaleform 4.1 では、ステレオ 3D をサポートするために Render::HAL 内に簡単な API を導入しました。この API は、ステレオ 3D 用のハードウェアの初期化をアプリケーションに提供するとともに、フレームバッファとサーフェスをセットアップして HDMI 1.4 フレームパッキングをサポートします。

最初にアプリケーションは、セクション 6.2.1 に示すように、アプリケーションが Scaleform レンダラー内で必要とするステレオパラメータを初期化しなければなりません。

Scaleform 4 では別々のステレオ画像を横に並べて、あるいは上下に並べてレンダーするようにビューポートを設定できます。この機能はステレオ画像を垂直、水平にタイルして表示する必要のあるデバイスで便利です。

これは **Render::Viewport::SetStereoViewport** を適切なフラッグと共に呼び出して行います。

SF4 には各アイに次の新たなビューポート ステレオ オプションのフラッグが追加されました。

```
// ディスプレーハードウェアのステレオのみ。同じバッファーサイズを使用するが、各アイには半分ずつ。
View_Stereo_SplitV        = 0x40,
View_Stereo_SplitH        = 0x80,
View_Stereo_AnySplit    = 0xc0,

void SetStereoViewport(unsigned display);
```

6.2.1Scaleform Stereo 初期化

```
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);
```

ディスプレイを走査している間、アプリケーションは Display をそれぞれの眼に対して 1 回ずつ合計で 2 回コールするとともに、それぞれの Display コールの前に正しいフレームバッファサーフェスに切り替えなければなりません。Scaleform は両目の視差オフセットを処理します。たとえば、

```
pMovie->Advance(delta);

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

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

7 サンプルファイル

3Di の実際の動作を見るには Scaleform SDK に同梱される 3D のサンプル Flash ファイル（SWF と FLA）を参考にしてください。

サンプルはデフォルトでは次のディレクトリに収録されています。
C:\Program Files (x86)\Scaleform\GFx SDK 4.4\Bin\Data\AS2or AS3\Samples

図 4: 3DGenerator

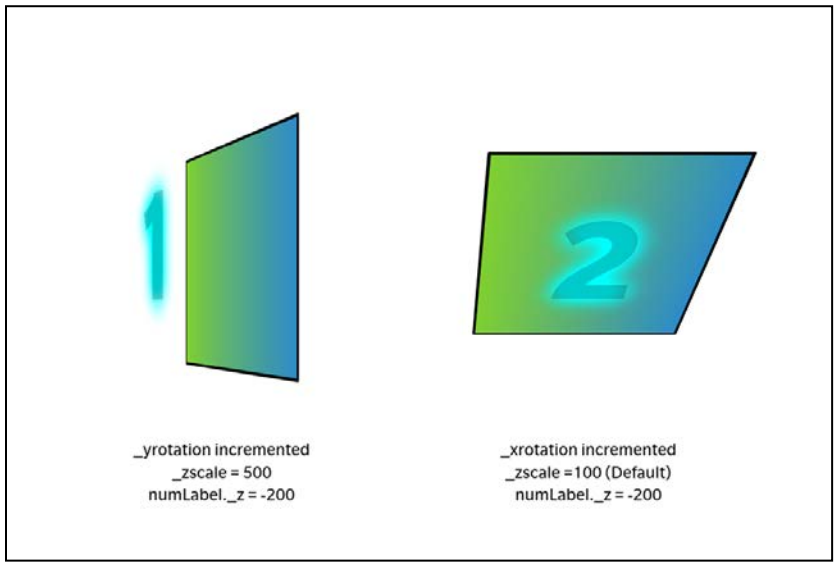

図 5: 3D スクエア


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

図 6: 3D ウィンドウ