ESライブラリ＆＆
ゲームプログラミング

○　ピクセルシェーダー編 - 第２回　フェードイン・アウト　○

フェードイン・アウト

・現在の画面が徐々に消えていったり、次の画面が徐々に現れたりする画面エフェクト
・現在の画面と次の画面を任意の比率で合成する
・スプライトを半透明で描画したり、ピクセルシェーダーで合成後の色値を計算する

概要

フェードイン・アウトは、徐々に画像が現れたり消えたりする効果のことで、場面切り替えのもっとも基本的なエフェクトです。ほとんどのゲームに使われています。

２つのシーンをなめらかに切り替えるように、現在のシーンが徐々に薄くなり、新しいシーンが徐々に濃くなっていきます。
プログラムでは、アルファブレンドを利用した方法で実現します。２つのシーンを合成する際、透明度を少しずつ変化させることによって再現できます。

Direct3Dでは、スプライト(テクスチャ)またはピクセルシェーダーによるピクセル値の演算によって実現することができます。

スプライトによるフェードイン・アウト

Direct3DXのスプライト(テクスチャ)は、透明度を指定して描画することができます。現在表示している画像のフェードイン・アウトしたい領域に、次の画面の画像や特定色で塗りつぶしたスプライトを半透明合成すれば、フェードイン・アウトの１フレームが完成します。あとは、時間経過に合わせて透明度を変化させれば、簡単にフェードイン・アウトが行えます。
半透明合成は、フェードイン・アウト以外にも、シーンの雰囲気に合わせた色合いや模様(夕焼けの色や霧模様など)を施したスプライトを画面に半透明合成することにより、よりリアルなシーンを表現することができます。

ピクセルシェーダーによるフェードイン・アウト

ピクセルシェーダーでも、ポストエフェクトとしてフェードイン・アウトが行えます。具体的には、「半透明合成」を計算で行うものです。

基本的な計算式は、以下のようになります。

出力色 = フェードの強さ * [フェードの色] + (1.0 - フェードの強さ) * [画像の色]
※フェードの強さは0.0f(フェードなし＝もとの画像)～1.0f(フェードの画像のみ)です

アルファブレンドの色値を計算する方法とまったく同じです。フェードの色と現在の画像の色を任意の比率で合成します。フェードの強さ(比率)を徐々に変えることにより、なめらかに次の画像へ切り替わります。

課　　題

ピクセルシェーダーを使って画面にフェードイン・アウトを適用してみましょう。

(１)画面全体を半分の明るさ(元の画像 * 0.5 + 黒 * 0.5 = 元の画像 * 0.5 ※黒は0のため)にしてみます。フェードイン・アウトのもっとも基本的なアルゴリズムです。
次のプログラムを"Fade.fx"として作成し、プロジェクトの"Shader"フォルダに保存しましょう。

－Fade.fx－

```
//-----------------------------------------------------------------------------
// File: Fade.fx
//
// The effect file for the HLSL sample.
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
sampler tex0 : register(s0);

//-----------------------------------------------------------------------------
// pass0 PixelShader Main Function
//-----------------------------------------------------------------------------
float4 PS_P0_Main(float2 UV : TEXCOORD0) : COLOR0
{
   float4   Color = tex2D(tex0, UV);

   ※ ここは各自考えましょう ※

   return Color;
}

//-----------------------------------------------------------------------------
// Techniques
//-----------------------------------------------------------------------------
technique Fade
{
   pass P0
   {
      VertexShader = NULL;
      PixelShader  = compile ps_2_0 PS_P0_Main();
   }
}
```

ピクセルシェーダーは、テクスチャ座標(ＵＶ座標)を受け取り、画面のピクセルに対応するテクスチャの色値を取得(tex2D関数)し、「元の画像 * 0.5」させます。

(２)(１)で作成したエフェクトファイルを読み込むように、プログラムを変更しましょう。

(３)レンダリング用の作業領域を生成します。まず、変数の宣言が必要です。ヘッダーファイルに以下のプログラムを追加しましょう。

```
RENDERTARGET    offscreen;
```

(４)作業領域を生成します。以下のプログラムをLoadContent関数の適切な場所に追加しましょう。

```
// オフスクリーンターゲット生成
offscreen = GraphicsDevice.CreateRenderTarget(1280, 720, SurfaceFormat_RGBA8888,
                                              DepthFormat_Unknown);
```

(５)「作業領域へのレンダリング→ピクセルシェーダーを用いて画面へ転送」を行うようにプログラムを変更します。CGameMain::Draw関数を以下のように変更しましょう。

```
void CGameMain::Draw()
{
    GraphicsDevice.Clear(Color_Black);

    // TODO: Add your drawing code here
    GraphicsDevice.BeginScene();

    GraphicsDevice.SetRenderTarget(offscreen);      // 作業領域をレンダリング出力先に設定
    GraphicsDevice.Clear(Color_Black);              // 作業領域のクリア
```

```
    // 2D描画
    SpriteBatch.Begin();
    SpriteBatch.Draw(bgSpr, Vector3(0.0f, 0.0f, 1.0f), 0.25f);
    SpriteBatch.End();

    // 3D描画
    GraphicsDevice.SetRenderState(CullMode, CullMode_None);

    // プレイヤー
    m_pPlyModel->SetScale(PLY_SCALE);
    m_pPlyModel->SetRotation(plyRot);
    m_pPlyModel->SetPosition(plyPos);
    m_pPlyModel->Draw();

    // 隕石
    for(int i = 0; i < METEO_MAX; i++) {
        m_pMeteoModel->SetPosition(meteoPos[i]);
        m_pMeteoModel->SetScale(meteoSize[i]);
        m_pMeteoModel->Draw();
    }

    // オフスクリーンターゲット→スクリーン
    GraphicsDevice.RenderTargetToBackBuffer(offscreen, effect);

    GraphicsDevice.EndScene();
}
```

(６)画面全体を１／４の明るさ(もとの画像 * 0.25)になるように、シェーダーを変更しましょう。

(７)シェーダーに「もとの画像の何倍にするか」を格納するグローバル変数を設定し、プログラム側で比率を自由に変更できるようにしましょう。

(８)時間経過にともない、比率が少しずつ変わるようにしましょう。

(９)フェードの色をプログラム側から設定できるようにしましょう。
ヒント　saturateという関数があります

応用問題　単色だけでなく、画像とのフェードイン・アウトを行うようにしましょう。

ヒント１　「sampler tex0 : register(s0);」というグローバル変数は、ステージ０に設定されたテクスチャを読み取れます
ヒント２　「sampler tex1 : register(s1);」とすると、ステージ１に設定されたテクスチャを読み取れるようになります
ヒント３　「GraphicsDevice.SetTexture(1, スプライト->GetTexture());」とすると、ステージ１にスプライト(のテクスチャ)を設定できます