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１回　ピクセルシェーダー　○

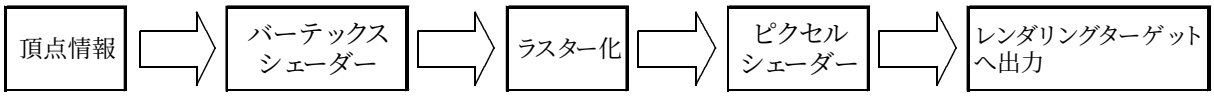
ピクセルシェーダー

・バーテックスシェーダー(座標変換)、ラスター化の後の最終的なピクセルの色を決定する
・ピクセルシェーダーには、画面に表示される直前のピクセル情報が送られてくる
・フェードイン・フェードアウトや色値反転、セピア調などのエフェクトが実現できる

・エフェクトファイルには、ピクセルシェーダーのみ記述することができるが、それでは座標変換が行われない → ワールド変換もカメラも反映されない
・ピクセルシェーダーのみ使う場合でも、T&L固定機能と同じ動作をするバーテックスシェーダーが必要となる → 各種座標変換と頂点の法線＋ライトの角度から頂点色を計算する…ちょっと難しい
・T&L固定機能とピクセルシェーダーを組み合わせるには、レンダリングターゲットを工夫する

概要

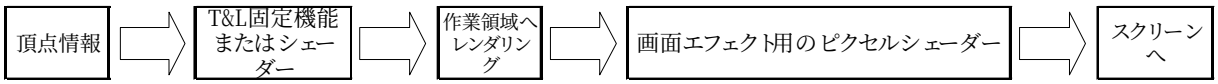
エフェクトファイルを用いて頂点を描画するおおまかな流れは、以下のようになっています。



DirectX9の仕様では、T&L固定機能を使いつつ、バーテックスシェーダーのみ、ピクセルシェーダーのみ使用するといったことができるようになっていますが、HLSLで記述したエフェクトファイルの場合は、T&L固定機能が働かないため、バーテックスシェーダーでの座標変換が必要です。一度作成すれば使い回すことができますが、各種座標変換とマテリアル、照明演算といったT&L固定機能と同じような機能を持ったシェーダープログラムが必要になります。簡単に使用できるT&L固定機能で座標変換と頂点色計算を行い、最終的な色値計算のみピクセルシェーダーを用いてエフェクトを行うという場合は、プログラムでの工夫が必要となります。

ESライブラリでの対応

ESライブラリでは、T&L固定機能またはシェーダーを用いてレンダリングし、最終的な画面エフェクトをピクセルシェーダーで行えるようするため、以下の仕組みが利用できます。



Direct3Dでは、レンダリングの出力先(レンダリングターゲット)を画面以外の任意のテクスチャに設定できます。これを利用し、T&L固定機能またはシェーダーを用いて作業領域のテクスチャへシーンをいったんレンダリングします。さらに、できあがったテクスチャをエフェクト用のシェーダーでディスプレイへ出力する、という流れです。作業領域へレンダリングした時点で、座標変換からピクセル化までのひととおりの処理が終わっているため、バーテックスシェーダーは不要です。よって、ピクセルシェーダーのみのエフェクトファイルでも、問題なく画面エフェクトを用いてディスプレイに出力できます。よって、ピクセルシェーダーでは、画面エフェクトのプログラムのみ記述することができます。このような一度描画した画像にエフェクトを加えることをポストエフェクトと呼びます。

課　　題

ピクセルシェーダーを使って画面にエフェクトを適用してみましょう。

(１)画面の色を反転するエフェクトをピクセルシェーダーで再現してみましょう。次のプログラムを"Inverse.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。

－Inverse.fx－

```
//-----------------------------------------------------------------------------
// File: Inverse.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);

   Color.rgb = 1.0f - Color.rgb;

   return Color;
}

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

ピクセルシェーダーでは、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反転しています。反転は、最大値が1.0、最小値が0.0のため、1.0から値を引くことにより行えます(「Color.rgb = 1.0f - Color.rgb;」の部分)。40%の反対は60%、という考えと同じです。ピクセルシェーダーから出力(return)された値がそのまま画面に表示されるため、1.0から引いた値を出力すれば、画面に反転した色が表示されることになります。

ここで、「ピクセルに対応するテクスチャ」とは、「作業領域のテクスチャ」(裏の裏の画面)のことです。このエフェクトを用いるには、レンダリングの出力先を通常の画面ではなく「作業領域のテクスチャ」にします。作業領域へレンダリングし、作業領域から画面エフェクト用のピクセルシェーダーをとおして、画面に出力するという流れです。ピクセルシェーダーでは、ＵＶ座標と色値の参照しか行えないため、このようなやや複雑な流れになります。

(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();
}
```

「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」は、やや複雑な処理ですが、ライブラリが吸収しています。
GraphicsDevice.SetRenderTarget関数を呼び出すと、レンダリング先を変更できます。以降のスプライトを含むすべての3D描画命令は、設定先へ出力されます。設定先から画面への転送は、GraphicsDevice.RenderTargetToBackBuffer関数です。引数にレンダリングターゲットとエフェクトを渡すと、指定されたエフェクトを用いて画面に転送されます。

(６)プログラムを実行し、実行結果を確認しましょう。

(７)モノトーンエフェクトを適用してみましょう。

反転エフェクトの色値計算部分を変更するだけで、モノトーンエフェクトがかけられます。モノトーンは、モノトーン用の係数とピクセル色値の内積を計算するだけで適用できます。シェーダーには、内積計算用のdot関数が用意されているので、変数をdot関数に渡すだけでできます。

```
float4 PS_PO_Main(float2 UV : TEXCOORD0) : COLOR0
{
   float4 Color = tex2D(tex0, UV);
   float4 Mono  = float4(0.298912, 0.586611, 0.114478, 0.0);

   Color.rgb = dot(Color.rgb, Mono);
   Color.a   = 1.0;
```

```
   return Color;
}
```

上記のようにピクセルシェーダーを変更し、実行結果を確認しましょう。

(８)セピア調エフェクトを適用してみましょう。

モノトーンエフェクトの色値計算部分をさらに変更すれば、セピア調の色調にもできます。モノトーンエフェクトの色値にセピア調エフェクト用の定数を乗算するだけで、セピア調にできます。

```
float4 PS_P0_Main(float2 UV : TEXCOORD0) : COLOR0
{
   float4 Color = tex2D(tex0, UV);
   float4 Mono  = float4(0.298912, 0.586611, 0.114478, 0.0);
   float4 Sepia = float4(0.941176, 0.784313, 0.293103, 1.0);

   Color.rgb    = dot(Color.rgb, Mono);
   Color.a      = 1.0;
   Color.rgb   *= Sepia;

   return Color;
}
```

上記のようにピクセルシェーダーを変更し、実行結果を確認しましょう。

(９)ナイトスコープ風のエフェクトを適用してみましょう。

```
float4 PS_P0_Main(float2 UV : TEXCOORD0) : COLOR0
{
   float4 Color = tex2D(tex0, UV);
   float4 Mono  = float4(0.298912, 0.586611, 0.114478, 0.0);
   float4 Night = float4(0.5, 0.75, 0.25, 1.0);

   Color.rgb    = dot(Color.rgb, Mono);
   Color.a      = 1.0;

   Color.rgb   *= Night;

   return Color;
}
```

シェーダーのグローバル変数に「float Brightness = 1.0;」のような明るさを管理する変数を宣言し、"Color.rgb *= Night;"の部分を「Color.rgb *= Night * Brightness;」と変更すると、プログラム側から明るさを制御できるようになります。

(10)ピクセルシェーダーを以下のように変更してみましょう。

```
float4 PS_P0_Main(float2 UV : TEXCOORD0) : COLOR0
{
   float4 Color = tex2D(tex0, UV);
   float4 Akai  = float4(1.000000, 0.743137, 0.547059, 1.0);

   Color.rgb   *= Akai;

   return Color;
}
```

※「千と千尋の神隠し」DVD風の色調になります