ESライブラリ＆＆
ゲームプログラミング

○　シェーダー共通編 - 第１回　プログラマブルシェーダー　○

プログラマブルシェーダー

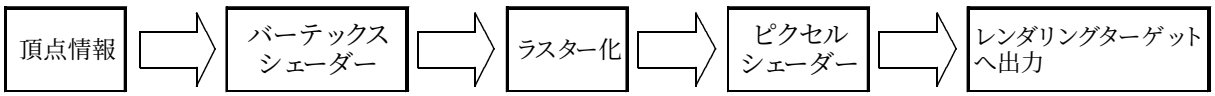
・GPUのグラフィックパイプラインをシェーダーと呼ばれるプログラムによって自由に構築できる
・シェーダーには、頂点演算を行うバーテックスシェーダーと、ピクセルの色値を計算する
　ピクセルシェーダーがある
・DirectX11では、いくつかのシェーダーが追加され、固定機能が廃止された

概要

プログラマブルシェーダーは、頂点データが画面に出力されるまでの一連の処理をプログラムで記述したものです。シェーダーはGPUで直接実行され、従来のT&L固定機能では不可能だったリアリティあふれる映像の構築が可能になります。PS3やXbox360でもシェーダーが導入されています。

プログラマブルシェーダーには、頂点の座標変換および色の演算を行うバーテックスシェーダーと、最終的に画面に表示される色値を計算するピクセルシェーダーがあります。これらのシェーダーを用いると、リアルな影の再現、バンプマップによる物体表面の凹凸の表現、ライトブルーム、高さフォグ、ボリュームフォグ、ディスプレースメントマップ、色調変換などといった技術を使い、リアリティあふれる映像が再現できます。

DirectXでシェーダー用いて頂点を描画するおおまかな流れは、以下のようになります。



バーテックスシェーダー

バーテックスシェーダーは、頂点の変換方法と頂点色を計算するための手順を定義するためのシェーダーです。3Dグラフィクスでは、数値の頂点情報に座標変換と色値計算を行い、画面のどこに何色でレンダリングするのかを計算します。バーテックスシェーダーで計算された頂点情報は、ラスター化を経てピクセルシェーダーに渡ります。

ピクセルシェーダー

ピクセルシェーダーは、それぞれのピクセルに関して、画面に表示される最終的な色の決定方法を記述するものです。頂点の色情報をさらに加工したり、テクスチャをほどこしたり、ピクセルを自在に操ることができます。バーテックスシェーダーは頂点データを処理するのに対し、ピクセルシェーダーは色値そのものの処理を行います(なので、ＵＶ座標の変更は出来ますが、ピクセルそのものを移動することはできません)。

HLSL

HLSL(High Level Shader Language)は、マイクロソフトによって開発されたDirect3Dで使われるＣ言語風のシェーディング言語です。バーテックスシェーダーとピクセルシェーダーはHLSLで記述します。

シェーダ開発ツール

NVIDIAとATIから、シェーダーを統合環境で開発できるツールが無償で配布されています。NVIDIAは「FX Composer」、ATIは「RenderMonkey」です。それぞれ下記URLからダウンロードできます。

・FX Composer :http://developer.nvidia.com/fx-composer
・RenderMonkey:http://developer.amd.com/archive/gpu/rendermonkey/pages/default.aspx

ESライブラリでの対応

ESライブラリでは､コンパイル前後のシェーダーファイルを読み込んで描画に用いることができます。シェーダーはGraphicsDevice.CreateEffectFromFile関数で読み込みます。コンパイル前のテキストファイルもGraphicsDevice.CompileEffectFromFile関数でコンパイルし、読み込むことができます。
返されたEFFECT型の変数をモデルや頂点バッファのDraw関数へ渡すと、シェーダーを使用して描画されます。シェーダーを使いたくない場合は渡さなければ良いので、これまでのT&L固定機能とシェーダーをモデルによって使い分けながら描画することができます。

課　　題

ピクセルシェーダーを使ってみましょう。

(１)以下のプログラムを"Bule.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。

－Blue.fx－

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

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
float4 g_OutColor = {0.0f, 0.0f, 1.0f, 1.0f};   // RGBA

// PixelShader Main
float4 PS_Main() : COLOR0
{
    return g_OutColor;
}

//-----------------------------------------------------------------------------
// Techniques
//-----------------------------------------------------------------------------
technique BluePixel
{
    pass P0
    {
        VertexShader = NULL;
        PixelShader  = compile ps_1_1 PS_Main();
    }
}
```

シェーダーが記述されたファイルをエフェクトファイルと呼び、拡張子.fxが多く使われています。このエフェクトファイルでは、"BluePixel"というテクニックが定義されています。

BluePixelテクニックでは、１つのパスがあり、"P0"という名前がつけられています。このパスでは、バーテックスシェーダーはNULL(つまり'なし')、ピクセルシェーダーとして「PS_Main関数」を実行するように設定しています(手前の"compile ps_1_1"は、ピクセルシェーダーのバージョン1.1として扱ってください、という意味です。低いバージョンを指定すると、対応できる環境が多くなり、バージョンを上げると高機能になります)。

シェーダーの実行が開始されると、上記で指定したバーテックスシェーダー→ピクセルシェーダーの順にプログラム(関数)が実行され、最終的な色値が出力されます。BluePixelテクニックでは、グローバル変数g_Colorで定義した色を単に出力しているだけとなっています。

(２)シェーダーを管理する変数を宣言します。以下のプログラムをヘッダーファイルの適切な場所に追加しましょう。

```
// エフェクト
EFFECT   effect;
```

(３)シェーダーを読み込みます。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。

```
    // エフェクト生成
    effect = GraphicsDevice.CreateEffectFromFile( TEXT("FX\\Blue.fxc") );
```

ESライブラリでは、"Shader"フォルダのすべてのエフェクトファイルがビルド時に自動的にコンパイルされ、コンパイル済みエフェクトファイル.fxcとして"FX"フォルダに出力されるように設定しています。毎回コンパイルされるので、コンパイルする必要ないエフェクトファイルは、拡張子を変更しておくか、別のフォルダに待避してください。

(４)GameMain::Draw関数を以下のように変更しましょう。

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

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

    for(UINT i = 0; i < effect->Begin(); i++) { … ①
        effect->BeginPass(i); … ②

        // 2D描画
        SpriteBatch.Begin(SpriteBlendMode_AlphaBlend, SpriteSortMode_BackToFront);
        SpriteBatch.Draw(m_pBGSprite, Vector3(0.0f, 0.0f, 1.0f), 0.25f);
        SpriteBatch.End();

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

        // プレイヤー
        plyMdl->SetScale(PLY_SCALE);
        plyMdl->SetRotation(plyRot);
        plyMdl->SetPosition(plyPos);
        plyMdl->Draw(); … ③

        // 隕石
        for(int i = 0; i < METEO_MAX; i++) {
            meteoMdl->SetPosition(meteoPos[i]);
            meteoMdl->SetScale(meteoSize[i]);
            meteoMdl->Draw(); … ③
        }

        effect->EndPass(); … ④
    }
    effect->End(); … ⑤

    GraphicsDevice.EndScene();

    // ダメージ描画
    (省略)

    // スコア描画
    (省略)
}
```

シェーダーを用いた描画は、

① シェーダー使用開始(Begin関数)　→
② パスの開始　(BeginPass関数)　→
③ シェーダーを使って描画したいモデルを描画 →
④ パス終了　(EndPass関数)　→
⑤ シェーダー終了

という順に行います。Begin関数を呼び出した時点でシェーダーが有効になり、T&L固定機能が使われなくなります。なお、パスとはシェーダーの処理単位のことで、バーテックスシェーダーとピクセルシェーダーをまとめたものです。
プログラムからは、BeginPass～EndPassによって、パス単位で実行を制御できます。１つのファイルに複数のパスを記述しておき、複数のパスを用いて描画したり、モデルによってパスを使い分けたりすることができます。複数のパスをまとめたものをテクニックと呼びます。エフェクトファイルにいくつパスが入っていても正常に動作するように、ループにしています(特定のパスのみ実行することも出来ます)。

"Blue.fx"では、"P0"がパスとして登録されています。今回のエフェクトファイルには、パスは１つしかありませんが、パスをまとめて"BluePixel"というテクニック名がつけられています。

(５)プログラムを実行し、実行結果を確認しましょう。
※(４)には１ヵ所正しくない場所があるので、環境によっては正常に動作しない場合があります
※※スプライトは内部でT&L固定機能を使っているため、エフェクトがかかりません

(６)エフェクトファイルを変更し、緑になるようにしましょう。