ESライブラリ＆＆
ゲームプログラミング

○　バーテックスシェーダー編 - 第３回　アンビエント　○

アンビエント

・アンビエントは、反射方向がなく、モデル全体に均等に色を与える
・ライトの色の影響を受けるが、ライトの角度や頂点の向き(法線)の影響は受けない

概要

アンビエントは、光の反射方向がない反射で、モデル全体に均等に色を与えます。ライトからの角度に依存しない反射ですが、ライトの色の影響は受けます。そのため、ライトがないと色がつきません。ライトと頂点の向きを考慮する必要がなく、ライトとマテリアルのアンビエント色のみで計算することができるので、簡単に実装することができます。

アンビエント

アンビエントの計算方法は、以下のようにとても簡単なものです。

アンビエント ＝ ライトのアンビエント色 × マテリアルのアンビエント反射

ライトとマテリアルの色(発光率と反射率)を掛け合わせる、というものです。たとえば、ライトのアンビエントがRGB(1.0, 1.0, 1.0)と白色、マテリアルのアンビエント反射率がRGB(1.0, 0.0, 0.0)と赤しか反射できない、という場合、

RGB(1.0, 1.0, 1.0) × RGB(1.0, 0.0, 0.0) ＝ RGB(1.0, 0.0, 0.0)

となり、頂点の色は「赤」になります。計算は、RGB各成分が独立して乗算されます。1.0に近ければ、その成分が強く発光している、もしくは強く反射できることを表し、0.0になると発光していない、吸収している、ということになります。

プログラムでは、シェーダーのグローバル変数としてライトとマテリアルのアンビエントを格納する変数を宣言し、プログラム側から色を渡してもらいます。角度の影響を一切受けないため、２つ色を乗算し、それを頂点色とします。エミッシブやディフューズなど、ほかの属性がある場合は、加算して合成します。

課　題

座標変換とアンビエント照明演算を行うバーテックスシェーダーを作成しましょう。

(１)以下のシェーダープログラムを"Ambient.fx"として作成し、Meteoプロジェクトの"Shader"フォルダに保存しましょう。

―Ambient.fx―

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

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
float4x4    g_WVP;

float4      g_LightAmbient;
float4      g_ModelAmbient;
```

```
//-----------------------------------------------------------------------------
// VertexShader Output Structure
//-----------------------------------------------------------------------------
struct VS_OUTPUT
{
    float4 Pos      : POSITION;
    float4 Color    : COLOR0;
};

//-----------------------------------------------------------------------------
// pass0 VertexShader Main Function
//-----------------------------------------------------------------------------
VS_OUTPUT VS_P0_Main(float4 Pos : POSITION)
{
    VS_OUTPUT   Out = (VS_OUTPUT)0;

    Out.Pos   = mul(Pos, g_WVP);

    // Vertex Color
    Out.Color = g_LightAmbient * g_ModelAmbient;

    return Out;
}

//-----------------------------------------------------------------------------
// Techniques
//-----------------------------------------------------------------------------
technique Ambient
{
   pass P0
   {
      VertexShader = compile vs_1_1 VS_P0_Main();
      PixelShader  = NULL;
   }
}
```

グローバル変数としてライトのアンビエント色とマテリアルのアンビエント反射を宣言しています。(「float4 g_LightAmbient」と「float4 g_ModelAmbient」)

処理内容は、まず座標変換を行い、射影座標にします。アンビエントは、計算式どおり

```
Out.Color = g_LightAmbient * g_ModelAmbient;
```

と、そのままコード化しています。前回のエミッシブも考慮する場合は、グローバル変数にエミッシブ色を追加し、その変数をOut.Colorに加算します。最後にその結果を出力(return Out)して終了です。

(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)シェーダーのグローバル変数に値を設定し、エフェクトを使って描画しましょう。

ライトとマテリアルのアンビエント設定が必要です。以下のプログラムを(２)の後に追加しましょう。

```
// エフェクトへライトのアンビエント色を設定
effect->SetFloat4("g_LightAmbient", Color(1.0f, 1.0f, 1.0f));

// エフェクトへマテリアルに対応するグローバル変数名を通知する
effect->※ここは各自考えましょう※;
```

ヒント：エフェクトへのマテリアル変数名の通知は、InitMaterial関数で行います。引数は、左からディフューズ、アンビエント、スペキュラー、エミッシブ、スペキュラーの強さ、テクスチャの順になっています。使用しないマテリアルは、NULLを指定します。

(４)モデルのアンビエント色を設定します。以下のプログラムを適切な場所に追加しましょう。

```
// マテリアル設定
Material   mtrl;

// プレイヤー
mtrl.Ambient = Color(0.3f, 0.3f, 0.3f);
plyMdl  ->SetMaterial(mtrl);

// 隕石
mtrl.Ambient = Color(0.1f, 0.1f, 0.1f);
meteoMdl->SetMaterial(mtrl);
```

描画などの部分は、前回と変更ありません。

(５)プログラムを実行し、動作を確認しましょう。座標変換が行われ、色がつけば成功です。

(６)ライトの色を変え、描画色が変更されるか確認しましょう。ライトの色によって描画色が変更されれば成功です。

応用問題：アンビエントだけでなく、エミッシブも考慮されるようにしましょう。