ESライブラリ＆＆
ゲームプログラミング
○　３Ｄ編 - 第１０回　影　○

影

・影(Shadow)は、物体が光を遮ることでできる
・陰(Shade)は、ライトで照らしてレンダリングすればできるが、影はできない
・リアルタイムに生成する影は負荷が高く完全ではないため、さまざまな方法が考案されている
・影は、臨場感が増すだけでなく遮蔽物の存在を確認できるなど、空間把握の手助けにもなる

概要

影は、物体が光を遮ることでできます。3Dグラフィクスでは、光は物体を照らして陰をつけることはしますが、光そのものは描画されません。影も同様に、描画されることはありません。

説得力の高い、臨場感あふれる映像を再現するには、影は欠かすことのできない要素の１つです。しかし、完全な影をリアルタイムに生成し描画する方法は、まだ考案されていません。また、リアルな影を再現しようとすると、負荷が高くなってしまいます。

影を生成する主な方法には、まる影、平面投影、投影テクスチャマッピング、シャドウボリューム、シャドウマップなどがあります。

まる影

丸い影のテクスチャを貼り付けたポリゴンをモデルの足下に描く方法です。負荷が非常に低く、位置関係を立体的に伝えることができます。まる影も発展しており、まる影１つを描画するものから、モデルの姿勢によって伸縮するもの、モデルのパーツごとにまる影を連結して描画するもの、光源の方向を考慮するものもあります。
あらゆるモデルの陰が"まる"をもとに描かれるため、リアリティはほとんどありませんし、セルフシャドウ(自分自身への影付け)もできませんが、負荷の低さから現在でも使われることがあります。また、リアルな影よりもシンボル的な影が良い場合もあります。

平面投影シャドウ

モデルを平面に投影したものを影として描画する方法です。基本的な考え方は、光源の方向と平面の向きを考慮し、該当軸方向(たとえば、上から照らす場合はｙ軸)を０に縮小したモデルを描画するというものです。平面投影のための変換行列を生成すれば、簡単に描画できるのが特長です。
ただし、凹凸のあるものへの投影やセルフシャドウはできません。また、同じモデルを本体、影と２回描画することになるため、複雑な形状のモデルが増えれば比例して負荷が高くなります。投影用のモデルやLODで形状を簡略化するなどの工夫が必要です。

投影テクスチャマッピング(プロジェクションシャドウ)

光源から見たモデルを影としてテクスチャに描画し､地面など影を落としたいモデルへ貼り付ければ、凹凸があるものへ投影できるというものです。基本的には、平面投影をテクスチャに行い、それを地面などに貼り付けるという考え方です。プレイステーション２で多く使われた方法です。

シャドウボリューム

影を落とすオブジェクトの光源方向からみた輪郭辺を抽出し、輪郭辺を光源と逆方向に引き伸ばした領域をシャドウボリュームと呼びます。シャドウボリュームが影になる候補の領域です。
ｚテストとステンシルバッファを使って影のステンシル(型枠)を作ります。影が落とされるオブジェクトを描画した後、ｚテスト有効、ｚバッファ更新なしでシャドウボリュームの表ポリゴンと裏ポリゴンをステンシルバッファに描画します。表と裏の描画は、カリングのステートを変更して行います。

表を描画するときにはステンシルに＋１、裏を描画するときにはステンシルに－１していきます(この操作は、両面ステンシル対応であれば１度に行えます)。最終的に影が落ちるピクセルはｚテストにより表しか描画されず、影になるピクセルはステンシル値が１となります。

プロジェクションシャドウと異なり、複雑な形状のオブジェクトに対しても一定の負荷で影が生成できセルフシャドウも可能ですが、オブジェクトが複雑だと輪郭抽出時の負荷が高くなります。また、本体モデル、影生成のために裏カリング、表カリングと何度か描画しなければならないため、フィルレート(描画能力)を大量に消費してしまうのが欠点です。カメラがボリューム内に入ると影が破綻するという欠点もありますが、改良方法も多く考案されています。

シャドウマッピング

現在主流となっている方法です。光源から見たモデルの深度値をテクスチャに書き出し、通常のカメラから見たシーン描画時に「光源からの深度値(Ａ)」と「カメラからの深度値(Ｂ)」を比較して、Ａ＞Ｂなら影とみなして描画する方法です。この方法もセルフシャドウが可能です。欠点は、プログラムが複雑になる点と、DirectX GraphicsではHLSLでなければ対応できない点があげられます(モデルのレンダリング、影付けすべてHLSLで行う必要があります)。

ESライブラリでの対応

ESライブラリでは、平面投影の変換行列を生成するためにMatrix_CreateShadow関数を用意しています。また、平面投影シャドウが簡単に描画できるように各種ステートの変更を行うGraphicsDevice.BeginShadowRendering関数とGraphicsDevice.EndShadowRendering関数があります。さらに、頂点バッファ、モデル、アニメーションモデルにも平面投影変換行列から黒でモデル描画を行うDrawShadow関数が用意されています。

これらの関数とカリング、LODを組み合わせることにより、平面のみですが簡単に影を落とすことができます。

課　　題

影を描画しましょう。

(１)モデルやアニメーションモデルが描画されるシーンを準備しましょう。

(２)影を描画します。

１．影はすべてのモデル本体を描画した後に行います。まず、平面投影行列を生成します。DirectXにはヘルパー関数があり、それをもとに作成されたMatrix_CreateShadow関数が用意されています。これを呼び出すだけで、簡単に投影行列が生成できます。

```
Matrix_CreateShadow(光源種類，光の方向ベクトルまたは光源位置，
                    モデル座標，モデル座標から影を描画する平面位置へのオフセット，
                    平面方向ベクトル)
```

※光源種類がLight_Directional(ディレクショナルライト)の場合は「光の方向ベクトル」、それ以外の場合は光源位置と解釈して平面投影行列を生成します

２．影の描画前に、GraphicsDevice.BeginShadowRendering関数で各種ステートの設定を行います。

３．影の描画後に、GraphicsDevice.EndShadowRendering関数で各種ステートを元に戻せます。

４．影の描画自体は、頂点バッファ／モデル／アニメーションモデルのDrawShadow関数で行います。引数は、DrawShadow(平面投影行列，影の濃さ)です。地面などからの距離によって影の濃さを調整すれば、シーンの説得力が増します。

５．以上をまとめると、次のようなコードになります。

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

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


    // 本体描画
    model->Draw();
    anime->Draw(GameTimer.GetElapsedSecond());


    // 影描画
    GraphicsDevice.BeginShadowRendering();

    Matrix   mat;

    mat = Matrix_CreateShadow(Light_Directional, Vector3(0.0f, -1.0f, -1.0f),
                              model->GetPosition(),
                              Vector3(0.0f, -0.9f, 0.0f),
                              Vector3_Up);
    model->DrawShadow(mat, 0.75f);

    mat = Matrix_CreateShadow(Light_Point, Vector3(-100.0f, 100.0f, -15.0f),
                              anime->GetPosition(),
                              Vector3(0.0f, 0.1f, 0.0f),
                              Vector3_Up);
    anime->DrawShadow(mat, 0.75f);

    GraphicsDevice.EndShadowRendering();


    GraphicsDevice.EndScene();
}
```