ESライブラリ＆＆
ゲームプログラミング

○　３Ｄ編 - 第７回　レンダリングステート　○

レンダリングステート

・レンダリングステートは、レンダリング時の動作を制御するさまざまな値のこと
・レンダリングステートは、IDirect3DDevice9::SetRenderStateメソッドで変更

概要

レンダリングステートとは、T&L固定機能において、ジオメトリパイプラインを制御するさまざまな設定値のことです。代表的なステートに、深度バッファ、カリング、アルファブレンド、プリミティブの描画モード、フォグなどがあります。

レンダリングステートの設定は、IDirect3DDevice9::SetRenderStateメソッドで行います。ステートは一度設定すると変更されない限り維持され続けます(ただし、Resetメソッドを呼び出した場合は、すべての設定値が消去され、デフォルトの状態に戻ります)。

```
// レンダリングステートの変更(pD3DDeviceは、初期化済みのIDirect3DDevice9オブジェクト)
pD3DDevice->SetRenderState(D3DRS_DITHERENABLE, TRUE);          // ディザリング有効
pD3DDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);      // ポリゴンの裏を描画する
pD3DDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME); // ワイヤフレームモード
```

Direct3Dでは、レンダリングステートがどのようになっているのかをしっかりと管理し、2D／3D／半透明など、それぞれにあったステートを設定してから描画を行う必要があります。

よく変更されるステートは以下のものです。

レンダリングステート変数	意　　味	デフォルト値
D3DRS_ZENABLE	深度バッファのステート	D3DZB_FALSE
D3DRS_ZWRITEENABLE	深度バッファへの書き込み	TRUE
D3DRS_ZFUNC	深度値の比較	D3DCMP_LESSEQUAL
D3DRS_CULLMODE	ポリゴンの背面処理	D3DCULL_CCW
D3DRS_FILLMODE	塗りつぶしモード	D3DFILL_SOLID
D3DRS_SHADEMODE	シェーディングモード	D3DSHADE_GOURAUD
D3DRS_AMBIENT	環境光の色	0
D3DRS_SPECULARENABLE	スペキュラの有効／無効	FALSE
D3DRS_ALPHATESTENABLE	アルファテストの有効／無効	FALSE
D3DRS_ALPHAREF	アルファテストの基準値	0
D3DRS_ALPHAFUNC	アルファピクセルの受け取り	D3DCMP_ALWAYS
D3DRS_LIGHTING	照明演算の有効／無効	TRUE
D3DRS_ANTIALIAS	アンチエイリアスモード	D3DANTIALIAS_NONE
D3DRS_DITHERENABLE	ディザリングの有効／無効	FALSE
D3DRS_ALPHABLENDENABLE	アルファブレンドの有効／無効	FALSE
D3DRS_SRCBLEND	転送元のブレンディング係数	D3DBLEND_ONE
D3DRS_DESTBLEND	転送先のブレンディング係数	D3DBLEND_ZERO
D3DRS_BLENDOP	ブレンディングの方法	D3DBLENDOP_ADD
D3DRS_STENCILENABLE	ステンシル処理の有効／無効	FALSE
D3DRS_COLORWRITEENABLE	カラーチャンネルの書き込み	0x0F
D3DRS_WRAP0～7	テクスチャのラッピング動作	0

デバイスステート変数	意　　味	デフォルト値
D3DRS_FOGENABLE	フォグの有効／無効	FALSE
D3DRS_FOGTABLEMODE	ピクセルフォグのモード	D3DFOG_NONE
D3DRS_FOGVERTEXMODE	バーテックスフォグのモード	D3DFOG_NONE
D3DRS_FOGCOLOR	フォグの色	0
D3DRS_FOGSTART	フォグ開始点	0.0
D3DRS_FOGEND	フォグ終了点	1.0
D3DRS_FOGDENSITY	フォグ密度	1.0

ESライブラリでは、CDXGraphics9クラスのSetRenderState関数でレンダリングステートの設定を行うことができます。

深度バッファ(Zバッファ)

深度バッファは、画面の各ピクセルの深度を保持する領域です。深度バッファを用いると、描画する順番にかかわらず、奥のものは描画されず、手前のものが描画されるようになります。

3Dグラフィクスでは「奥にあるもの」より「手前にあるもの」が前に描画されなければなりません。奥にあるものは手前のものに隠される必要があり、2Dでは奥から描画するなど工夫して再現していました。3Dでも同様に、なんらかの方法でこの処理を行う必要があります。

ESライブラリでは、デフォルトで自動的に奥のものが隠されて描画されるようになっています。これは、画面の色データ以外に、深度バッファという奥行きのデータも保持しているため、より手前のピクセルしか描画されないようになっているからです。
ポリゴンのラスター化時に深度バッファが参照され、描画しようとしているピクセルがより「遠い」ピクセルの場合は上書きされないようになっています。「近い」ピクセルの場合は上書きされ、深度バッファも更新されます。

カリング

カリングとは、ポリゴンの向きによって描画するかしないかを指定するものです。デフォルトでは、視点から見て「頂点が反時計回り」になっているポリゴンは、「裏を向いている」という扱いになり、描画されません。これは、裏を向いているポリゴンはモデルの背面＝死角にあることが多いため、描画しない方がパフォーマンス向上につながるためです。

しかし、隙間のあるモデル、半透明モデル、一枚板などように、あえて裏面を描画しなければならないものもあります。このような場合は、"D3DRS_CULLMODE"の設定を行うことにより、カリングの動作を変更することができます。

```
// カリングモードの設定
pD3DDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);   // カリングなし
```

D3DRS_CULLMODEでは、以下の値を設定できます。

D3DCULL_NONE	カリングしません。すべての向きのポリゴンが描画されます
D3DCULL_CCW	反時計回りのポリゴンをカリングします(CounterClockwiseFace)
D3DCULL_CW	時計回りのポリゴンをカリングします(ClockwiseFace)

アルファブレンド

ブレンドとは、2つの色を混ぜて、それぞれの色を反映した新たな色を作成する処理です。このとき、ピクセルや頂点のアルファ値(透明度の値)をもとにブレンドを行うものをアルファブレンドと呼びます。アルファブレンドを行うと、奥のものが透けて見えるような効果を得ることができます。

アルファブレンドでは、「転送元の色」と「転送先の色」が混合されて最終的な色が計算されますが、どのように混ぜあわせるかを指定することができます。

基本式は以下のようになっています。

(転送元の色 × 転送元ブレンド割合)(ブレンド演算)(転送先の色 × 転送先ブレンド割合)

「転送元ブレンド割合」をD3DRS_SRCBLEND、「転送先ブレンド割合」をD3DRS_DESTBLEND、「ブレンド演算」をD3DRS_BLENDOPで設定します。

一般的な半透明処理は、「転送元ブレンド割合」が転送元のアルファ値をそのまま使用、「ブレンド演算」が足し算、「転送先ブレンド割合」が１－アルファ値(転送元と先のアルファ値を足すと１になるようになっています)を使用します。よって、

(転送元の色 × α) ＋ (転送先の色 × (１－α))

という式になります。これは、以下のコードになります。

```
// アルファブレンドを有効にする
pD3DDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);

// アルファブレンディングの設定([転送元*α]+[転送先*(1-α)])
pD3DDevice->SetRenderState(D3DRS_BLENDOP,   D3DBLENDOP_ADD);         // 下の２つを足す
pD3DDevice->SetRenderState(D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA);      // α値そのまま使用
pD3DDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);   // １－α(↑の反転)
                           ⋮
              (半透明モデルを描画する)
                           ⋮
pD3DDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, FLASE);       // アルファブレンド無効
```

アルファブレンドは、デフォルトで無効になっているので、AlphaBlendEnableプロパティを有効にする必要があります。

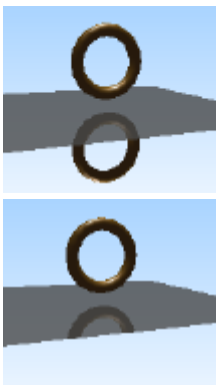
レンダリングステートの設定を工夫することにより、「加算ブレンディング」「乗算ブレンディング」「減算ブレンディング」など、様々な効果を用いたブレンドが行えます。

```
// 加算ブレンディング
pD3DDevice->SetRenderState(D3DRS_SRCBLEND,  D3DBLEND_ONE);
pD3DDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ONE);

// 乗算ブレンディング
pD3DDevice->SetRenderState(D3DRS_SRCBLEND,  D3DBLEND_ZERO);
pD3DDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_SRCCOLOR);
```

ステンシルバッファ・アルファテスト

ステンシルバッファは、モデルの影や映り込みを再現するときに有効な機能で、描画する領域を制限することができます。たとえばモデルの影(陰はマテリアルとライトで表現)は、影で暗くなる所をステンシルバッファによって描画できるようにしておけば、あとは半透明の黒を描画するだけで表現することができます。



▲上ステンシルなし
下ステンシルあり

アルファテストは、画面ピクセルのアルファ値によって、描画するかしないかを決めることができます。深度バッファの"深度"が"透明度"になったもので、深度ではなく透明度で描画するかどうかを決定したいときに使用します。多くの場合、深度バッファで不都合が出る場合にテクスチャと組み合わせて使用します。

たとえば、「細かい枝の隙間から奥のオブジェクトが見える」というシーンの場合、細かい枝はポリゴン数がかかってしまうので、四角形ポリゴン＋テクスチャで簡易的に再現します。このとき、奥にあるオブジェクトは、深度バッファによって手前にある四角形ポリゴンに隠されるため、描画されません。このまま「細かい枝」のテクスチャを貼っても、奥のオブジェクトは描画自体されていないため、枝の隙間から見ることはできません。

このような場合、テクスチャに透明度情報(アルファ値)を設定しておき、深度ではなくアルファ値によってピクセルを描画するかどうかを判定させます。こうすれば、奥のオブジェクトも枝の隙間から見えるように描画することができます。

▲深度バッファだけでは正しく描画できない

▲アルファテスト＋テクスチャで描画

```
// アルファテスト設定の例
pD3DDevice->SetRenderState(D3DRS_ALPHATESTENABLE, TRUE);          // アルファテスト有効
pD3DDevice->SetRenderState(D3DRS_ALPHAFUNC, D3DCMP_GREATEREQUAL);          // 比較関数
pD3DDevice->SetRenderState(D3DRS_ALPHAREF,  128);   // ↑とペアで使用。α値128以上で描画
pD3DDevice->SetRenderState(D3DRS_ZENABLE,   FALSE);                  // 深度バッファ無効
                        ⋮
                (モデルを描画する)
                        ⋮
// 深度バッファを有効にする
pD3DDevice->SetRenderState(D3DRS_ALPHATESTENABLE, FALSE);          // アルファテスト無効
pD3DDevice->SetRenderState(D3DRS_ZENABLE, TRUE);                     // 深度バッファ有効
```

アルファテストはそのほかに、「複雑な形に画像を切り取る」という効果にも使えます。

アルファブレンドと名称は似ていますが、まったく別の動作です。アルファブレンドは設定された透明度によって、プリミティブ自体を半透明にブレンドするもので、アルファテストはラスター化時に画面に書き込まれるピクセルの透明度によって、そのピクセルを描画するかどうか判定するものです。

アルファテストを使うと円形に描画することもできる▶

課　　題

レンダリングステートを変更してみましょう。

ESライブラリでは、レンダリングステートの設定をGraphicsDevice(CDXGraphics9クラス)のSetRenderState関数で行うことができます。引数は、Direct3Dと同じように、Microsoftが定義する"D3DRS_"から始まるD3DRENDERSTATETYPE列挙型または"XNA.h"で定義されるXNA風の定数を渡すことができます。

(１)塗りつぶしモードを変更してみます。以下のプログラムを適切な場所に追加し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(FillMode, FillMode_Point);
```

(２)(１)を以下のように変更し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(FillMode, FillMode_WireFrame);
```

(３)(２)を以下のように変更し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(FillMode, FillMode_Solid);
```

(４)シェーディングモードを変更し、フラットシェーディングにしてみます。(２)を以下のように変更し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(ShadeMode, ShadeMode_Flat);
```

(５)シェーディングモードをグーローシェーディングにしてみます。(３)を以下のように変更し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(ShadeMode, ShadeMode_Gouraud);
```

(６)本来は反時計回りのポリゴンのカリングを行いますが、逆に時計回りのポリゴンをカリングしてみます。(５)を以下のように変更し、どのようになるか確認しましょう。

```
GraphicsDevice.SetRenderState(CullMode, CullMode_CullClockwiseFace);
```

(７)「反時計回りのポリゴン」のカリングを行うように、レンダリングステートの変更を行いましょう。

(８)背面カリングを行わないようにプログラムを変更しましょう。