オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics 3D編 － 第１０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とは、T&L固定機能において、ジオメトリパイプラインをコントロールするさまざまな設定値のことです。
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。DirectX Graphicsでは、このようなステートは、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)。

IDirect3DDevice9::SetRenderStateメソッド

－説明－
SetRenderStateメソッドは、レンダリングステートパラメータを設定します。

－書式－
HRESULT SetRenderState(D3DRENDERSTATETYPE *State*, DWORD *Value*);

－パラメータ－
１つ目の引数(State)は、変更対象のレンダリングステート変数(D3DRENDERSTATETYPE列挙型)です。
２つ目の引数(Value)は、レンダリングステートの新しい値です。パラメータの意味は、１つ目の引数に依存します。

－戻り値－
関数が成功するとD3D_OK、引数の１つが無効な場合は、D3DERR_INVALIDCALLを返します。

```
// 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
```

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_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_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_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_CULLMODE	ポリゴンの背面処理	D3DCULL_CCW
D3DRS_SPECULARENABLE	スペキュラの有効／無効	FALSE
D3DRS_AMBIENT	環境光の色	0
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_STENCILENABLE	ステンシル処理の有効／無効	FALSE
D3DRS_CLIPPING	プリミティブのクリッピング	TRUE

デバイスステート変数	意　　味	デフォルト値
D3DRS_WRAP0～7	テクスチャのラッピング動作	0
D3DRS_TEXTUREPERSPECTIVE	テクスチャの遠近補正	FALSE
D3DRS_FOGENABLE	フォグの有効／無効	FALSE
D3DRS_RANGEFOGENABLE	範囲ベースフォグの有効／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

課　　題

CDXGraphics9クラスに、レンダリングステートを設定する機能を追加しましょう。

(１)レンダリングステートのステート値を保存する構造体を定義します。

レンダリングステートの設定値は、デバイスをリセットするとデフォルトに戻ってしまいます。リセットに対応させるため、設定値を保存するようにします。また、リセット以外にデフォルト値に戻す方法がないため、デフォルト値も保存しておきます。

以下の構造体を適切な場所に追加しましょう。

```
// ステート保存構造体
struct DXGSAVESTATE {
    DWORD   Value;      // 現在の設定値
    DWORD   DefValue;   // デフォルトの設定値
};
```

(２)デバイスのリセットに備え、CDXGraphics9クラスに変更されたステートを保存するメンバを追加します。以下のプログラムを適切な場所に追加しましょう。

```
// 変更されたレンダリングステートを格納するマップ
std::map<D3DRENDERSTATETYPE, DXGSAVESTATE>   m_RenderState;
```

mapとは、STLが提供するコンテナの１つで、キーと要素のペアを格納します。各要素は２分探索木で格納されるので、順次検索よりも平均して速く検索できます。setと同じく、キーの重複は許されていません(重複を許可したい場合は、multisetやmultimapを使います)。

このmapには、レンダリングステート変数をキーとしてステート値を保存します。保存したステート値は、リセットやデフォルト値に戻す場合に使用します。

(３)mapをコンパイルしたときの警告を押さえるため、以下のプログラムをDXGraphics9.hppの「#pragma once」の下に追加しましょう。

```
#pragma warning(disable:4786)
```

(４)レンダリングステートを設定するSetRenderState関数を作成します。次のプログラムを完成させ、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                        レンダリングステート設定                                     */
/*****************************************************************************************************/
void CDXGraphics9::SetRenderState(const D3DRENDERSTATETYPE inState, const DWORD inValue)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
```

```
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_SetRenderState)\n");
        return;
    }
#endif

    // ステートがマップに登録されているか調べる
    std::map<D3DRENDERSTATETYPE, DXGSAVESTATE>::iterator   it;
    it = m_RenderState.find(inState);
    if(it == m_RenderState.end()) {
        // 登録されていない場合
        DXGSAVESTATE   SaveState;
        SaveState.Value = inValue;
        m_pD3DDevice->GetRenderState(inState, &SaveState.DefValue);     // デフォルト値を保存

        if(inValue == SaveState.DefValue)
            return;     // 設定値がデフォルト値の場合は終了

        // マップに登録する
        m_RenderState.insert(std::pair<D3DRENDERSTATETYPE, DXGSAVESTATE>(inState, SaveState));
    } else {
        // 登録済みの場合
        if(it->second.DefValue != inValue)
            it->second.Value = inValue;     // 設定値の更新
        else
            m_RenderState.erase(it);        // デフォルト値に戻る場合は、マップから削除する
    }

    // レンダリングステート設定
    m_pD3DDevice->??????????????(inState, inValue);
}
```

(５)レンダリングステートをデフォルト値に戻すClearRenderState関数を作成します。次のプログラムを適切な場所に追加しましょう(プロトタイプも忘れずに追加しましょう)。

```
/*****************************************************************************************************/
/*                                      レンダリングステート消去                                       */
/*****************************************************************************************************/
void CDXGraphics9::ClearRenderState(const D3DRENDERSTATETYPE inState)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_ClearRenderState)\n");
        return;
    }
#endif

    // ステートがマップに登録されているか調べる
    std::map<D3DRENDERSTATETYPE, DXGSAVESTATE>::iterator   it;
    it = m_RenderState.find(inState);
    if(it != m_RenderState.end()) {
        // 登録されている場合
        m_pD3DDevice->SetRenderState(it->first, it->second.DefValue);   // デフォルト値に戻す
        m_RenderState.erase(it);                                        // マップから削除する
    }
}
```

(６)すべてのレンダリングステートをデフォルト値に戻すClearAllRenderStates関数を作成します。次のプログラムを適切な場所に追加しましょう(プロトタイプも忘れずに追加しましょう)。

```
/*****************************************************************************************************/
/*                                      全レンダリングステート消去                                     */
/*****************************************************************************************************/
void CDXGraphics9::ClearAllRenderStates()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_SetRenderState)\n");
        return;
    }
#endif
```

```
    // レンダリングステートをデフォルト値に戻す
    std::map<D3DRENDERSTATETYPE, DXGSAVESTATE>::iterator   it;
    for(it = m_RenderState.begin(); it != m_RenderState.end(); it++)
        m_pD3DDevice->SetRenderState(it->first, it->second.DefValue);

    // マップの消去
    m_RenderState.clear();
}
```

(7)CDXGraphics9クラスの解放時に、すべてのレンダリングステートが消去されるようにします。以下のプログラムを適切な場所に追加しましょう。

```
    m_RenderState.clear();
```

(8)リセットの後に、保存しておいたステート値をもとに、レンダリングステートの設定値を復元します。以下のプログラムを適切な場所に追加しましょう。

```
    // レンダリングステート復元
    std::map<D3DRENDERSTATETYPE, DXGSAVESTATE>::iterator   render_it;
    for(render_it = m_RenderState.begin(); render_it != m_RenderState.end(); render_it++)
        m_pD3DDevice->SetRenderState(render_it->first, render_it->second.Value);
```

(9)レンダリングステートを変更してみます。CTestSceneクラスを第7回の課題(14)の状態(シンプルシェイプを表示し、マテリアルとライトの設定がされている状態)にしましょう。

(10)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CTestScene::CTestScene関数に追加し、どのようになるか確認しましょう。

```
    DXGraphics().SetRenderState(D3DRS_FILLMODE, D3DFILL_POINT);
```

(11)(10)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。

```
    DXGraphics().SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
```

(12)(11)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。

```
    DXGraphics().SetRenderState(D3DRS_FILLMODE, D3DFILL_SOLID);
```

(13)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。以下のプログラムをCTestScene::CTestScene関数に追加し、どのようになるか確認しましょう。

```
    DXGraphics().SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
```

(14)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。(13)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。

```
    DXGraphics().SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
```

(15)背面カリング(裏面のポリゴンを描画しない)を行わないようにしてみます。以下のプログラムをCTestScene::CTestScene関数に追加し、どのようになるか確認しましょう。

```
    DXGraphics().SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
```

(16)背面カリングを行うようにしてみます。(15)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。

```
    DXGraphics().SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
```

(17)スペキュラー処理を有効にしてみます。以下のプログラムをCTestScene::CTestScene関数に追加しましょう。

```
DXGraphics().SetRenderState(D3DRS_SPECULARENABLE, TRUE);
```

(18)メッシュにスペキュラーを設定してみます。「マテリアル設定」を以下のように修正し、どのような結果になるかを確認しましょう。

```
// マテリアル設定
D3DMATERIAL9   material;
::ZeroMemory(&material, sizeof(material));
material.Specular = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
m_pMesh->SetMaterial(0, material);
```

(19)メッシュにディフューズ反射、アンビエント反射、スペキュラー反射を設定してみます。(12)のプログラムを以下のように修正し、どのような結果になるかを確認しましょう。

```
// マテリアル設定
D3DMATERIAL9   material;
::ZeroMemory(&material, sizeof(material));
material.Diffuse  = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f);
material.Ambient  = D3DXCOLOR(0.0f, 1.0f, 0.0f, 1.0f);
material.Specular = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
m_pMesh->SetMaterial(0, material);
```