# オブジェクト指向とゲームプログラミング

## ▶ DirectX Graphics 3D編 － 第６回　マテリアル ◀

### マテリアル

マテリアルは、表面属性ともいわれるもので、オブジェクト表面の質感を表現する機能です。マテリアルは、光の反射と放射という属性を持ちます。光の反射には「ディフューズ反射」「アンビエント反射」「スペキュラー反射」があります。オブジェクトの色は、これらの反射によって決定されます。光の放射には「エミッシブ」というオブジェクト自体を発光させる属性があります。

**ディフューズ反射**
ディフューズ反射は、光を一定の方向に反射し、オブジェクトに陰影を与えます。ライトからの角度で反射が決まり、ライトに向いているところほど明るくなります。オブジェクトの陰の色は、この反射の影響を大きく受けます。

**アンビエント反射**
アンビエント反射は、光の反射方向がない反射で、オブジェクト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で反射します。

**スペキュラー反射**
スペキュラー反射は、オブジェクトの表面上での乱反射を再現し、オブジェクトに質感を与えます。物体の光沢を表す反射で、ライトの向きに影響されます。乱反射が小さいほど光沢が増し、大きいほど光沢が減ります。
スペキュラー反射では、光沢を出す光の「反射率」と「鮮明度」を設定します。反射率と鮮明度が大きいほど、オブジェクトの輝きが増します。
もっとも明るい部分(ハイライト)の色は、この反射の影響を大きく受けます。

**エミッシブ**
エミッシブは、オブジェクト自体から光を放射する機能です。エミッシブはライトの影響を受けないので、ライトが無くても物体自体が蛍光塗料を塗られたように光るような効果が得られます。ただし、別のオブジェクトを照らすことはできないので、光源の代わりにはなりません。

オブジェクトにマテリアルを適用するには、D3DMATERIAL9構造体にマテリアルの属性を設定し、IDirect3DDevice9::SetMaterialメソッドに渡すことで行います。以後、すべてのオブジェクトのレンダリング時に、設定したマテリアルが適用されます。

```
// マテリアル設定
D3DMATERIAL9   material;
ZeroMemory(&material, sizeof(material));
material.Diffuse  = D3DXCOLOR(0.5f, 0.0f, 0.0f, 1.0f);      // ディフューズ反射
material.Ambient  = D3DXCOLOR(0.0f, 1.0f, 0.0f, 1.0f);      // アンビエント反射
material.Specular = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);      // スペキュラー反射
material.Emissive = D3DXCOLOR(0.2f, 0.2f, 0.2f, 1.0f);      // エミッシブ
material.Power    = 5.0f;                                   // スペキュラー反射の鮮明度

// マテリアルの適用(pD3DDeviceは初期化済みのDirect3DDevice9オブジェクト)
pD3DDevice->SetMaterial(&material);
```

マテリアルにスペキュラーを設定しても、デフォルトでは無効になっているため効果が現れません。スペキュラー演算を有効にするには、IDirect3DDevice9::SetRenderStateメソッドの１つ目の引数に「D3DRS_SPECULARENABLE」、２つ目の引数に「TRUE」を指定し、レンダリングステートの変更を行います。

```
// スペキュラー演算を有効にする
pD3DDevice->SetRenderState(D3DRS_SPECULARENABLE, TRUE);
```

# 課　　題

頂点バッファクラスとメッシュクラスに、マテリアルを設定する機能を追加しましょう。

(１)頂点バッファクラスに、マテリアルを設定する機能を追加します。マテリアルを設定するSetMaterial関数のプロトタイプをIVertexBufferクラス定義の適切な場所に追加しましょう。

```
virtual void SetMaterial(const D3DMATERIAL9& inMaterial) = 0;
```

(２)CVertexBufferクラスに、マテリアルを保持するメンバ変数を追加します。

マテリアルは、3Dオブジェクトの表面の反射を定義するものなので、3Dオブジェクトが個別に持つ属性となります。以下のメンバ変数をCVertexBufferクラスに追加しましょう。

```
D3DMATERIAL9    m_Material;
```

(３)CVertexBufferクラスのインスタンス生成時に、(２)で追加したメンバが初期化されるようにします。以下のプログラムをコンストラクタの適切な場所に追加しましょう。

```
// マテリアル初期化
::ZeroMemory(&m_Material, sizeof(m_Material));
```

(４)CVertexBufferクラスにSetMaterial関数を追加します。以下のプログラムを適切な場所に追加しましょう。

```
virtual void SetMaterial(const D3DMATERIAL9& inMaterial) { m_Material = inMaterial; }
```

(５)CNullVertexBufferクラスに「何もしない」SetMaterial関数を追加しましょう。

(６)頂点バッファのレンダリング時に、マテリアルが適用されるようにします。CVertexBuffer::Render関数の適切な場所に、以下のプログラムを完成させて追加しましょう。

```
// マテリアル設定
if((m_FVF & D3DFVF_NORMAL) != 0)
    m_pD3DDevice->???????????(&m_Material);
```

(７)メッシュクラスに、マテリアルを設定する機能を追加します。マテリアルを設定するSetMaterial関数のプロトタイプをIMeshクラス定義の適切な場所に追加しましょう。

```
virtual void SetMaterial(const DWORD inSubset, const D3DMATERIAL9& inMaterial) = 0;
```

メッシュは、サブセットごとにマテリアルを持つので、サブセットのインデックス値が引数に加わります。

(８)CMeshクラスに、マテリアルを保持するメンバ変数を追加しましょう。

```
std::vector<D3DMATERIAL9>    m_Material;
```

メッシュは、サブセットごとにマテリアルを保持しますが、サブセットの数はメッシュごとに異なるので、固定配列では対応できません。そこで、配列の長さを自由に設定できるvectorを使用します。

(９)CMeshクラスのインスタンス生成時に、(９)で追加したメンバが初期化されるようにします。以下の初期化子をコンストラクタに追加しましょう。

```
m_Material(inSubsetCount)
```

サブセット数(inSubsetCount)をvectorのコンストラクタに渡すことにより、その数だけ配列の要素が確保されます。

(10)CMeshクラスのコンストラクタでマテリアル配列の要素を初期化します。以下のプログラムを適切な場所に追加しましょう。

```
// マテリアル初期化
for(DWORD i = 0; i < m_Material.size(); i++) {
    ::ZeroMemory(&m_Material[i], sizeof(m_Material[i]));
}
```

(11)CMeshクラスにSetMaterial関数を追加します。以下のプログラムを適切な場所に追加しましょう。

```
virtual void SetMaterial(const DWORD inSubset, const D3DMATERIAL9& inMaterial)
{ ※この関数の内容は、各自考えましょう }
```

(12)CNullMeshクラスに「何もしない」SetMaterial関数を追加しましょう。

(13)メッシュのレンダリング時に、マテリアルが適用されるようにします。CMesh::Render関数の適切な場所に、以下のプログラムを完成させて追加しましょう。

```
// マテリアル設定
m_pD3DDevice->???????????(&m_Material);
```

(14)保持しているマテリアルを返す機能(GetMaterial関数)が必要な場合は、それぞれのクラスに実装しましょう。