オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics 3D編 ― 第７回　ライト ◀

ライト

オブジェクトにマテリアルを設定しただけでは色が付きません。オブジェクトは、実世界と同じように、光で照らされることにより反射して色が付き、見えるようになります。
DirectX Graphicsでは、オブジェクトを照らす光を「ライト」という機能で表現します。DirectX Graphicsのライトは「ポイントライト」、「スポットライト」、「ディレクショナルライト」の３種類用意されています。デバイスの上限まで、同時にいくつでも使用することができます。
さまざまな光を再現できるように、それぞれのライトには位置や方向、減衰率といった属性と色があります。色は、マテリアルと同じように、ディフューズ、アンビエント、スペキュラーがありますが、マテリアルと異なり、これらは互いの色に影響を与えることはありません。オブジェクトの最終的な色は、ライトとマテリアルの同じ属性どうしが演算されたものになります。

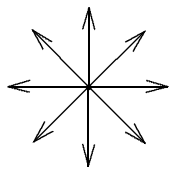
ライトの属性は、D3DLIGHT9構造体に設定します。この構造体のメンバは以下のようになっています。

```
D3DLIGHTTYPE    Type;         // ライトの種類
D3DCOLORVALUE   Diffuse;      // 光のディフューズ色
D3DCOLORVALUE   Specular;     // 光のスペキュラー色
D3DCOLORVALUE   Ambient;      // 光のアンビエント色
D3DVECTOR       Position;     // ライトの位置(ワールド座標)
D3DVECTOR       Direction;    // 光の方向ベクトル
float           Range;        // ライトの有効範囲(0.0～FLT_MAXの平方根)
float           Falloff;      // スポットライトのフォールオフ
float           Attenuation0; // 定常減衰係数
float           Attenuation1; // 線形減衰係数
float           Attenuation2; // 平方減衰係数
float           Theta;        // スポットライト内側の円錐の角度(０～πラジアン)
float           Phi;          // スポットライト外側の円錐の角度(０～πラジアン)
```

D3DLIGHT9構造体は、３つのライトに必要なメンバが定義されています。ライトの種類によって使用するメンバが異なります。

ポイントライト

ポイントライトは、ライトの位置から全方向に一様に照らす電球のようなライトです。

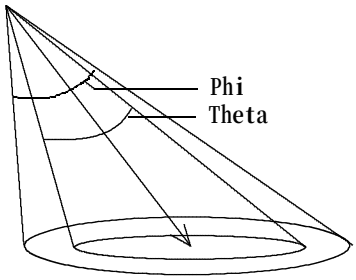


ポイントライトは「光の色」「光源の位置」「光の減衰」「光の有効範囲」の属性を持ちます。D3DLIGHT9構造体は、Type, Diffuse, Specular, Ambient, Position, Range, Attenuation0～2メンバに値を設定します。

```
D3DLIGHT9   light;
ZeroMemory(&light, sizeof(light));

light.Type         = D3DLIGHT_POINT;                    // ポイントライト
light.Diffuse      = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f); // ディフューズ色
light.Specular     = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f); // スペキュラー色
light.Ambient      = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f); // アンビエント色
light.Position     = D3DXVECTOR3(10.0f, 15.0f, 0.0f);   // ライトの位置(左からxyz)
light.Range        = 20.0f;                             // ライトの有効範囲
light.Attenuation0 =  1.0f;                             // 減衰定数
```

スポットライト
　スポットライトはその名のとおり、ライトの位置から光が一定方向に照射され、その照射範囲は距離が遠くなるにつれ広がります。

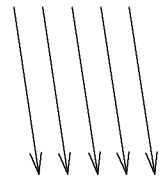


　光の照射範囲はライトの位置を頂点とする円錐状になり、この円錐はコーンとも呼ばれます。また、スポットライトには「フォールオフ」と呼ばれる減衰があります。フォールオフとは、円錐の内側から外側に向かって起こる光の減衰のことです。スポットライトは光の減衰によって「内側の明るい部分」と「外側の暗い部分」に分かれます。

　スポットライトは「光の色」「光の方向」「光源の位置」「光の減衰」の属性を持ちます。D3DLIGHT9構造体はすべてのメンバを使用します。ライトの位置を設定するPositionメンバとともに、ライトの照射方向を決めるDirectionメンバの設定が必要になります。また、ライトの強度が最も高いコーンの角度を指定するThetaメンバ、コーン全体の広がりを決定するPhiメンバなどの設定も重要です。Falloffメンバは、ThetaからPhiまでのライト強度の減衰量を決定します。

```
light.Type         = D3DLIGHT_SPOT;                          // スポットライト
light.Diffuse      = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);      // ディフューズ色
light.Specular     = D3DXCOLOR(0.5f, 0.5f, 0.5f, 1.0f);      // スペキュラー色
light.Ambient      = D3DXCOLOR(0.2f, 0.2f, 0.2f, 1.0f);      // アンビエント色
light.Position     = D3DXVECTOR3(1.0f, 15.0f, 2.0f);         // ライトの位置
light.Direction    = D3DXVECTOR3(0.0f, -1.0f, 0.0f);         // ライトの方向ベクトル
light.Range        = 12.0f;                                  // ライトの有効範囲
light.Falloff      =  1.0f;                                  // フォールオフ
light.Attenuation0 =  1.0f;                                  // 定常減衰係数
light.Theta        = D3DXToRadian(10.0f);                    // 内側のコーンの角度
light.Phi          = D3DXToRadian(30.0f);                    // 外側のコーンの角度
```

ディレクショナルライト
　ディレクショナルライトは、地球を照らす太陽光のように、無限大の距離からすべてのオブジェクトに対し、一方向に平行な光を照射します。



　ディレクショナルライトは「光の色」と「光の方向」の属性を持ちます。D3DLIGHT9構造体は、Type, Diffuse, Specular, Ambient, Directionメンバを設定します。

```
light.Type      = D3DLIGHT_DIRECTIONAL;                  // ディレクショナルライト
light.Diffuse   = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);     // ディフューズ色
light.Specular  = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);     // スペキュラー色
light.Ambient   = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);     // アンビエント色
light.Direction = D3DXVECTOR3(0.0f, -1.0f, 0.0f);        // ライトの方向ベクトル
```

ライトの設定

　D3DLIGHT9構造体に設定したライトは、IDirect3DDevice9::SetLightメソッドを呼び出すことにより、デバイスに割り当てられます。

IDirect3DDevice9::SetLightメソッド

－説明－

SetLightメソッドは、デバイスにライトを割り当てます。

－書式－

```
HRESULT SetLight(DWORD Index, CONST D3DLIGHT9* pLight);
```

－パラメータ－

１つ目の引数(Index)は、ライトのインデックス値(ライトの番号)です。すでにセットされているインデックス値を指定すると、新しいライトで上書きされます。

２つ目の引数(pLight)は、ライトの情報を格納したD3DLIGHT9構造体のアドレスを指定します。

－戻り値－

関数が成功するとD3D_OK、それ以外はエラーコードを返します。

SetLightメソッドはライトを割り当てるだけなので、このままでは使用されません。IDirect3DDevice9::LightEnableメソッドでライトを有効にすることで、光を出すようになります。

IDirect3DDevice9::LightEnableメソッド

－説明－

LightEnableメソッドは、指定されたライトを有効または無効にします。同時に有効にできるライトの最大数は、IDirect3DDevice9::GetDeviceCapsメソッドで取得することができます。

－書式－

```
HRESULT LightEnable(DWORD LightIndex, BOOL bEnable);
```

－パラメータ－

１つ目の引数(LightIndex)は、操作の対象となるライトのインデックス値です。

２つ目の引数(bEnable)は、ライトを有効にする場合はTRUE、無効にする場合はFALSEを指定します。

－戻り値－

関数が成功するとD3D_OK、それ以外はエラーコードを返します。

```
pD3DDevice->SetLight(0, &light);    // ライトをインデックス値０に割り当てる
pD3DDevice->LightEnable(0, TRUE);   // インデックス値０のライトを有効にする
```

照明演算はデフォルトで有効になっています。独自の照明演算を行う場合は、明示的に無効にする必要があります。照明演算の無効化は、IDirect3DDevice9::SetRenderStateメソッドの１つ目の引数に「D3DRS_LIGHTING」、２つ目の引数に「FALSE」を指定することにより行います。

```
// 照明演算を無効に設定
pD3DDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
```

照明演算を無効にすると、LightEnableメソッドで設定した状態にかかわらず、すべてのライトが使用されなくなります(以下のアンビエントライトは除く)。

アンビエントライト(環境光)

DirectX Graphicsには、３つのライト以外に、アンビエントライトという特殊なライトがあります。このライトは、すべてのオブジェクトに対し、方向に関係なく均一に当たります。すべてのオブジェクトのマテリアルのアンビエント反射に影響し、一定の明るさを提供します。

アンビエントライトは、IDirect3DDevice9::SetRenderStateメソッドで設定します。このメソッドの１つ目の引数に「D3DRS_AMBIENT」、２つ目の引数に色を指定します。

```
// アンビエントライトの設定
pD3DDevice->SetRenderState(D3DRS_AMBIENT, D3DXCOLOR(0.5f, 0.5f, 0.5f, 1.0f));
```

課　　題

CDXGraphics9クラスに、ライトを設定する機能を追加しましょう。

(１)CDXGraphics9クラスに、ライトを割り当てるSetLight関数を追加します。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(プロトタイプも忘れずに追加しましょう)。

```
/*****************************************************************************************************/
/*                                          ライト設定                                               */
/*****************************************************************************************************/
void CDXGraphics9::SetLight(const DWORD inIndex, const D3DLIGHT9& inLight, const BOOL inEnable)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_SetLight)\n");
        return;
    }
#endif
    m_pD3DDevice->????????   (inIndex, &inLight);   // 指定されたインデックスに、ライトを割り当てる
    m_pD3DDevice->??????????(inIndex, inEnable);   // ライトを有効または無効にする
}
```

SetLight関数の使用は、以下のとおりです。

SetLight　設定

指定されたインデックスにライトを割り当て、そのライトを有効または無効にします。

試 void SetLight(const DWORD inIndex, const D3DLIGHT9& inLight, const BOOL inEnable);

inIndex	ライトを割り当てるインデックス
inLight	割り当てるライトの情報を格納したD3DLIGHT9構造体
inEnable	ライトの有効または無効にするかを示す値　　有効：TRUE　無効：FALSE

(２)ライトを割り当てるLightEnable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(プロトタイプも忘れずに追加しましょう)。

```
/*****************************************************************************************************/
/*                                          ライト設定                                               */
/*****************************************************************************************************/
void CDXGraphics9::LightEnable(const DWORD inIndex, const BOOL inEnable)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_LightEnable)\n");
        return;
    }
#endif
    m_pD3DDevice->??????????(inIndex, inEnable);   // ライトを有効または無効にする
}
```

(３)ライトをデバイスのリセットに対応させます。

IDirect3DDevice9::Resetメソッドを呼び出すと、デバイスに割り当てられているすべてのライトが消去されてしまいます。復元する機能は用意されていないので、リセット前に割り当てられているライトの情報を保存し、リセット後に戻します(マテリアルや各変換行列も消去されますが、これらはレンダリング前にその都度設定するため、復元しなくても問題ありません)。

割り当てられているライトのインデックスを保存する以下のメンバを適切な場所に追加しましょう。

```
std::set<DWORD>   m_LightIndex;      // 割り当てられているライトのインデックスを格納するセット
```

setとは、STLが提供するコンテナの１つです。vectorのように要素を格納しますが、全要素は常にソートされており、格納する要素自身をキーとして検索することができます。キー(要素)を重複して格納することはできません。順次検索が必要なvectorやlistよりも平均して速く検索することができます。

このsetには、ライトが割り当てられているインデックスを保存しておきます。vectorやlistでも良さそうですが、インデックスを重複して保存する必要はないので、setを用います。vectorやlistだと、インデックスが登録されているかどうかを、ライトを割り当てるたびに調べなければなりません。

リセット行う前に、setに登録されているインデックスのライトの情報を保存し、リセット後に復元するというわけです。

(４)set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)
```

(５)ライトを割り当てたときに、setにインデックスを登録します。次のプログラムをCDXGraphics9::SetLight関数の適切な場所に追加しましょう。

```
m_LightIndex.insert(inIndex);  // setにインデックスを登録
```

(６)割り当てられているすべてのライトを消去するClearAllLights関数を作成します。以下のプログラムを完成させ、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)。

```
/*****************************************************************************************************/
/*                                          全ライトクリア                                           */
/*****************************************************************************************************/
void CDXGraphics9::ClearAllLights()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL)
        return;
#endif

    D3DLIGHT9   light;
    ::ZeroMemory(&light, sizeof(light));

    std::set<DWORD>::iterator   it;
    for(it = m_LightIndex.begin(); it != m_LightIndex.end(); it++) {
        m_pD3DDevice->LightEnable(*it, FALSE);
        m_pD3DDevice->SetLight   (*it, &light);
    }

    m_LightIndex.clear();
}
```

(７)CDXGraphics9クラスの解放時に、すべてのライ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_LightIndex.clear();
```

(８)ライトの情報を保存するための構造体を定義します。以下のプログラムを適切な場所に追加しましょう。

```
// ライト構造体
struct DXGLIGHT {
    DWORD       Index;
    D3DLIGHT9   Light;
    BOOL        Enable;
};
```

(９)リセットの前に、デバイスに割り当てられているすべてのライトを保存します。以下のプログラムを適切な場所に追加しましょう。

```
// ライト保存
DWORD   idx = 0;
std::vector<DXGLIGHT>       LightData(m_LightIndex.size());
std::set<DWORD>::iterator   light_it;
for(light_it = m_LightIndex.begin(); light_it != m_LightIndex.end(); light_it++) {
    LightData[idx].Index = *light_it;
    m_pD3DDevice->GetLight      (*light_it, &LightData[idx].Light);
    m_pD3DDevice->GetLightEnable(*light_it, &LightData[idx].Enable);
    idx++;
}
```

(10)リセットの後に、保存しておいたライトを復元します。以下のプログラムを適切な場所に追加しましょう。

```
// ライト復元
for(idx = 0; idx < LightData.size(); idx++) {
    m_pD3DDevice->????????   (LightData[idx].Index, &LightData[idx].Light);
    m_pD3DDevice->???????????(LightData[idx].Index,  LightData[idx].Enable);
}
```

(11)ライトを設定してみます。次のプログラムをCTestScene::CTestScene関数の適切な場所に追加しましょう。

```
// ライト設定
D3DLIGHT9   light;
light.Type      = D3DLIGHT_DIRECTIONAL;                 // ディレクショナルライト
light.Diffuse   = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // ディフューズ色
light.Ambient   = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // アンビエント色
light.Specular  = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // スペキュラ色
light.Direction = D3DXVECTOR3(0.0f, -1.0f, 0.0f);       // ライトの向き(下)
DXGraphics().SetLight(0, light, TRUE);                  // ライトを設定
```

(12)メッシュにディフューズ反射を設定してみます。以下のプログラムをCTestScene::CTestScene関数の適切な場所に追加し、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);
m_pMesh->SetMaterial(0, material);
```

(13)メッシュにアンビエント反射を設定してみます。(11)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.Ambient = D3DXCOLOR(0.0f, 1.0f, 0.0f, 1.0f);
m_pMesh->SetMaterial(0, material);
```

(14)メッシュにディフューズ反射とアンビエント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);
m_pMesh->SetMaterial(0, material);
```

※スペキュラー反射は、レンダリングステートで無効に設定されているため、設定しても効果は現れません。