Chapter18-19 ポイントライト　[応用]

DirectX１０以降、固定機能ライティングがなくなってから一般的なライティングを実現するためにもシェーダーの記述が必要になった。
３D グラフィックスの代表的なライトとして利用されるものは平行光源・ポイントライト・スポットライトの３種類が存在するが、ここでは通常のライティング効果として扱った以外のポイントライトとスポットライトを扱う。

ポイントライト（点光源）はその名前の通り、ポイント（点）を中心としてライトの効果を適用するライトで電球のようなライトの効果を実現できる。
ポイントライトの情報に必要なものは光源の座標と色になる。また光源から離れればライトの効果が弱くなるようにするため、距離による減衰値をパラメーターとして用意している。

ポイントライトのライティング処理は描画するピクセルのワールド座標と光源のワールド座標から光源へのベクトルとその距離を求め、光源へのベクトルと法線ベクトルでランバート拡散照明によるライティング計算をおこなっている。距離は減衰値を利用して光源の影響力を求め最終のライトの強さに積算をしている。

Program18-19 Sample.hlsl

```
//! コンスタントバッファ
cbuffer cbSceneParam : register( b0 )
{
	float4		vecViewPos		: packoffset( c0 );
	matrix		mtxView			: packoffset( c1 );
	matrix		mtxProj			: packoffset( c5 );
};

cbuffer cbMeshParam : register( b1 )
{
	matrix		mtxWorld		: packoffset(c0);
	float4		colRevise		: packoffset(c4);
	float4		CoordsRevise		: packoffset(c5);
};

cbuffer cbMaterialParam : register( b2 )
{
	float4		matDiffuse		: packoffset( c0 );
	float4		matAmbient		: packoffset( c1 );
	float4		matSpeculer		: packoffset( c2 );
	float4		matEmissive		: packoffset( c3 );
	float		matPower		: packoffset( c4 );
};

cbuffer cbLightParam : register( b3 )
{
	float3		litDirection		: packoffset( c0 );
	float4		litDiffuse		: packoffset( c1 );
	float4		litAmbient		: packoffset( c2 );
	float4		litSpeculer		: packoffset( c3 );
};

Texture2D txDiffuse : register( t0 );
SamplerState samLinear : register( s0 );

//! 頂点属性
struct InputVS
{
	float4		pos			: POSITION;
	float3		normal			: NORMAL;
	float2		Tex			: TEXCOORD;
	float4		color			: COLOR0;
};
```

```
struct OutputVS
{
	float4	pos			: SV_POSITION;
	float3	normal			: NORMAL;
	float3		wpos			: COLOR0;
};

//! 新規のコンスタントバッファ作成
cbuffer cbGameParam : register( b4 )
{
	float4		cblgPosition		: packoffset(c0);
	float4		cblgDiffuse			: packoffset(c1);
	float4		cblgAttenuation		: packoffset(c2);
};

//! 頂点シェーダ
OutputVS RenderVS( InputVS inVert )
{
	OutputVS	outVert;

	matrix		mtxVP = mul( mtxView, mtxProj );
	float4 Pos = float4(inVert.pos.xyz, 1.0);
	Pos = mul(Pos, mtxWorld );
	outVert.pos = mul( Pos , mtxVP );
	outVert.normal = normalize( mul( inVert.normal, (float3x3)mtxWorld ) );
	outVert.wpos = Pos;

	return outVert;
}

//! ピクセルシェーダ
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
	//法線
	float3 N = normalize(inPixel.normal);
	//ハーフランバートライティング
	float l = dot(N, -litDirection);
	l = l * 0.5f + 0.5f;
	l *= l;
	float3 amb = litAmbient.xyz * matAmbient.xyz;
	float3 diff = l * litDiffuse.xyz * matDiffuse.xyz;
	//ポイントライティング
	float3 lv = cblgPosition.xyz - inPixel.wpos.xyz;
	float d = length(lv);
	lv = normalize(lv);
	float pl = dot(N, lv);
	float Att = 1.0f / (cblgAttenuation.x + cblgAttenuation.y * d + cblgAttenuation.z * d * d);
	pl *= Att;
	diff += clamp(pl, 0, 1) * cblgDiffuse.xyz * matDiffuse.xyz;
	float4 LP = float4(saturate(matEmissive.xyz + amb + diff), matDiffuse.w);
	return LP;
}

technique11 TShader
{
	pass P0
	{
		SetVertexShader( CompileShader( vs_4_0, RenderVS() ) );
		SetGeometryShader( NULL );
		SetHullShader( NULL );
		SetDomainShader( NULL );
		SetPixelShader(  CompileShader( ps_4_0, RenderPS() ) );
		SetComputeShader( NULL );
	}
}
```

ゲーム側ではシェーダーと同じ構造体を作成し、描画処理でライトの情報を設定して描画を実行する。

Program18-19 GameApp.cpp

```
//カメラ
CCamera                             gCamera;
//メッシュ
LPGeometry                          pGeometry;
//シェーダー
CShader                             gShader;
CShaderBind_3DPrimitiveBase         gShaderBind;

//シェーダー側のコンスタントバッファと同様の構造体
struct cbGameParam
{
    Vector4    cblgPosition;
    Vector4    cblgDiffuse;
    Vector4    cblgAttenuation;
};

MofBool CGameApp::Initialize(void){
    //リソース配置ディレクトリの設定
    CUtilities::SetCurrentDirectory("Resource");

    //カメラ初期化
    gCamera.SetViewPort();
    gCamera.LookAt(Vector3(-2.0f,2.0f,-2.0f),Vector3(0,0,0),Vector3(0,1,0));
    gCamera.PerspectiveFov(MOF_ToRadian(60.0f),1024.0f / 768.0f,0.01f,1000.0f);
    gCamera.Update();
    CGraphicsUtilities::SetCamera(&gCamera);

    //メッシュの読み込み
    pGeometry = CGraphicsUtilities::CreatePlaneGeometry(10, 10, 1, 1, TRUE, Vector3(0, 0, 0));

    //シェーダーの読み込み
    gShader.Load("Shader.hlsl");
    gShaderBind.Create(&gShader);

    //シェーダーとのバッファの連携
    gShaderBind.CreateShaderBuffer("cbGameParam", sizeof(cbGameParam));
    return TRUE;
}
```

Program18-19 GameApp.cpp

```
MofBool CGameApp::Render(void){
    //描画処理
    g_pGraphics->RenderStart();
    //画面のクリア
    g_pGraphics->ClearTarget(0.0f,0.0f,1.0f,0.0f,1.0f,0);

    //深度バッファ有効化
    g_pGraphics->SetDepthEnable(TRUE);

    //カメラを設定
    gShaderBind.SetCamera(&gCamera);

    //シェーダーに送るバッファを作成
    cbGameParam sb;
    static float st = 0.0f;
    st += 0.01f;
    sb.cblgPosition = Vector3(0, 5 + sin(st) * 5, 0);
    sb.cblgDiffuse = Vector4(1, 1, 1, 1);
    sb.cblgAttenuation = Vector4(0.0f, 0.1f, 0.05f, 0);
```

```
	//シェーダーにバッファを送る
	gShaderBind.GetShaderBuffer(0)->SetBuffer(&sb);

	//メッシュの描画
	CMatrix44 matWorld;
	pGeometry->Render(matWorld, &gShader, &gShaderBind);
	//描画の終了
	g_pGraphics->RenderEnd();
	return TRUE;
}
```

Chapter18-20 スポットライト　[応用]

スポットライトはポイントライトのように光源の座標からのライティングを実現するが、その光に指向性（方向）を持たせ一定の方向にのみライトの影響を適用する。

スポットライトはポイントライトの情報に加え方向と影響を与える角度とその減衰値を持たせる。

Program18-20 Sample.hlsl

```
//! コンスタントバッファ
cbuffer cbSceneParam : register( b0 )
{
	float4		vecViewPos		: packoffset( c0 );
	matrix		mtxView			: packoffset( c1 );
	matrix		mtxProj			: packoffset( c5 );
};

cbuffer cbMeshParam : register( b1 )
{
	matrix		mtxWorld		: packoffset(c0);
	float4		colRevise		: packoffset(c4);
	float4		CoordsRevise		: packoffset(c5);
};

cbuffer cbMaterialParam : register( b2 )
{
	float4		matDiffuse		: packoffset( c0 );
	float4		matAmbient		: packoffset( c1 );
	float4		matSpeculer		: packoffset( c2 );
	float4		matEmissive		: packoffset( c3 );
	float		matPower		: packoffset( c4 );
};

cbuffer cbLightParam : register( b3 )
{
	float3		litDirection		: packoffset( c0 );
	float4		litDiffuse		: packoffset( c1 );
	float4		litAmbient		: packoffset( c2 );
	float4		litSpeculer		: packoffset( c3 );
};

Texture2D txDiffuse : register( t0 );
SamplerState samLinear : register( s0 );

//! 頂点属性
struct InputVS
{
	float4		pos			: POSITION;
	float3		normal			: NORMAL;
	float2		Tex			: TEXCOORD;
	float4		color			: COLOR0;
};
struct OutputVS
{
	float4		pos			: SV_POSITION;
	float3		normal			: NORMAL;
	float3		wpos			: COLOR0;
};

//! 新規のコンスタントバッファ作成
cbuffer cbGameParam : register( b4 )
{
	float4		cblgPosition		: packoffset(c0);
```

```
    float4      cblgDirection       : packoffset(c1);
    float4      cblgDiffuse             : packoffset(c2);
    float4      cblgAttenuation     : packoffset(c3);
    float   cblgCosTheta        : packoffset(c4.x);
    float   cblgCosPhi              : packoffset(c4.y);
    float   cblgFallOff             : packoffset(c4.z);
};

//! 頂点シェーダ
OutputVS RenderVS( InputVS inVert )
{
    OutputVS   outVert;

    matrix      mtxVP = mul( mtxView, mtxProj );
    float4 Pos = float4(inVert.pos.xyz, 1.0);
    Pos = mul(Pos, mtxWorld );
    outVert.pos = mul( Pos , mtxVP );
    outVert.normal = normalize( mul( inVert.normal, (float3x3)mtxWorld ) );
    outVert.wpos = Pos;

    return outVert;
}

//! ピクセルシェーダ
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
    //法線
    float3 N = normalize(inPixel.normal);
    //ハーフランバートライティング
    float l = dot(N, -litDirection);
    l = l * 0.5f + 0.5f;
    l *= l;
    float3 amb = litAmbient.xyz * matAmbient.xyz;
    float3 diff = l * litDiffuse.xyz * matDiffuse.xyz;
    //スポットライト
    float3 lv = cblgPosition.xyz - inPixel.wpos.xyz;
    float d = dot(-cblgDirection, lv);
    lv = normalize(lv);
    float pl = dot(N, lv);

    float Att = 1.0f / (cblgAttenuation.x + cblgAttenuation.y * d + cblgAttenuation.z * d * d);
    float lc = dot(-cblgDirection, lv);
    if (lc > cblgCosPhi)
    {
        Att *= saturate(pow(abs(max(lc - cblgCosPhi, 0.0f) *
            (1 / (cblgCosTheta - cblgCosPhi))), cblgFallOff));
    }
    else
    {
        Att = 0.0;
    }

    pl *= Att;
    diff += clamp(pl, 0, 1) * cblgDiffuse.xyz * matDiffuse.xyz;

    float4 LP = float4(saturate(matEmissive.xyz + amb + diff), matDiffuse.w);
    return LP;}

technique11 TShader
{
    pass P0
    {
        SetVertexShader( CompileShader( vs_4_0, RenderVS() ) );
        SetGeometryShader( NULL );
        SetHullShader( NULL );
```

```
		SetDomainShader( NULL );
		SetPixelShader(  CompileShader( ps_4_0, RenderPS() ) );
		SetComputeShader( NULL );
	}
}
```

ゲーム側ではシェーダーと同じ構造体を作成し、描画処理でライトの情報を設定して描画を実行する。

Program18-19 GameApp.cpp

```
//カメラ
CCamera                             gCamera;
//メッシュ
LPGeometry                          pGeometry;
//シェーダー
CShader                             gShader;
CShaderBind_3DPrimitiveBase         gShaderBind;

//シェーダー側のコンスタントバッファと同様の構造体
struct cbGameParam
{
	Vector4	cblgPosition;
	Vector4	cblgDirection;
	Vector4	cblgDiffuse;
	Vector4	cblgAttenuation;
	float	cblgCosTheta;
	float	cblgCosPhi;
	float	cblgFallOff;
	float	dummy;};

MofBool CGameApp::Initialize(void){
	//リソース配置ディレクトリの設定
	CUtilities::SetCurrentDirectory("Resource");

	//カメラ初期化
	gCamera.SetViewPort();
	gCamera.LookAt(Vector3(-2.0f,2.0f,-2.0f),Vector3(0,0,0),Vector3(0,1,0));
	gCamera.PerspectiveFov(MOF_ToRadian(60.0f),1024.0f / 768.0f,0.01f,1000.0f);
	gCamera.Update();
	CGraphicsUtilities::SetCamera(&gCamera);

	//メッシュの読み込み
	pGeometry = CGraphicsUtilities::CreatePlaneGeometry(10, 10, 1, 1, TRUE, Vector3(0, 0, 0));

	//シェーダーの読み込み
	gShader.Load("Shader.hlsl");
	gShaderBind.Create(&gShader);

	//シェーダーとのバッファの連携
	gShaderBind.CreateShaderBuffer("cbGameParam", sizeof(cbGameParam));
	return TRUE;
}
```

Program18-19 GameApp.cpp

```
MofBool CGameApp::Render(void){
	//描画処理
	g_pGraphics->RenderStart();
	//画面のクリア
	g_pGraphics->ClearTarget(0.0f,0.0f,1.0f,0.0f,1.0f,0);

	//深度バッファ有効化
	g_pGraphics->SetDepthEnable(TRUE);
```

```
	//カメラを設定
	gShaderBind.SetCamera(&gCamera);

	//シェーダーに送るバッファを作成
	cbGameParam sb;
	sb.cblgPosition = Vector3(0, 2.5f, 0);
	sb.cblgDirection = Vector3(0, -1, 0.6f);
	CVector4Utilities::Normal(sb.cblgDirection, sb.cblgDirection);
	sb.cblgDiffuse = Vector4(1, 1, 1, 1);
	sb.cblgAttenuation = Vector4(0.0f, 0.1f, 0.05f, 0);
	sb.cbLgCosTheta = cos(MOF_ToRadian(30));
	sb.cbLgCosPhi = cos(MOF_ToRadian(50));
	sb.cbLgFallOff = 3.2f;//シェーダーにバッファを送る
	gShaderBind.GetShaderBuffer(0)->SetBuffer(&sb);

	//メッシュの描画
	CMatrix44 matWorld;
	pGeometry->Render(matWorld, &gShader, &gShaderBind);
	//描画の終了
	g_pGraphics->RenderEnd();
	return TRUE;
}
```