# 7.13 光照演示程序

在我们的光照演示程序中，我们实现了3种个不同的光源：1个平行光、1个点光、1个聚光灯。平行光的位置是固定的，点光绕着地形转圈，而聚光灯跟随相机移动，并指向相机的观察方向。光照演示程序只是对上一章的水波演示程序做了一些修改。effect文件的内容在下一节中，其中使用的结构体和函数都已在7.10节中讨论过了。

****

**图7.21：光照演示程序的屏幕截图。**

## 7.13.1 effect文件

//=============================================================================

// Lighting.fx by Frank Luna (C) 2011 All Rights Reserved.

//

// Transforms and lights geometry.

//=============================================================================

#include "LightHelper.fx"

cbuffer cbPerFrame

{

 DirectionalLight gDirLight;

 PointLight gPointLight;

 SpotLight gSpotLight;

 float3 gEyePosW;

};

cbuffer cbPerObject

{

 float4x4 gWorld;

 float4x4 gWorldInvTranspose;

 float4x4 gWorldViewProj;

 Material gMaterial;

};

struct VertexIn

{

 float3 PosL : POSITION;

 float3 NormalL : NORMAL;

};

struct VertexOut

{

 float4 PosH : SV\_POSITION;

 float3 PosW : POSITION;

 float3 NormalW : NORMAL;

};

VertexOut VS(VertexIn vin)

{

 VertexOut vout;

 // 转换到世界空间

 vout.PosW = mul(float4(vin.PosL, 1.0f), gWorld).xyz;

 vout.NormalW = mul(vin.NormalL, (float3x3)gWorldInvTranspose);

 // 转换到齐次剪裁空间

 vout.PosH = mul(float4(vin.PosL, 1.0f), gWorldViewProj);

 return vout;

}

float4 PS(VertexOut pin) : SV\_Target

{

 // 插值后的法线需要重新规范化

 pin.NormalW = normalize(pin.NormalW);

 float3 toEyeW = normalize(gEyePosW - pin.PosW);

 // 初始化光照变量

 float4 ambient = float4(0.0f, 0.0f, 0.0f, 0.0f);

 float4 diffuse = float4(0.0f, 0.0f, 0.0f, 0.0f);

 float4 spec = float4(0.0f, 0.0f, 0.0f, 0.0f);

 // 每个光源的贡献

 float4 A, D, S;

 ComputeDirectionalLight(gMaterial, gDirLight, pin.NormalW, toEyeW, A, D, S);

 ambient += A;

 diffuse += D;

 spec += S;

 ComputePointLight(gMaterial, gPointLight, pin.PosW, pin.NormalW, toEyeW, A, D, S);

 ambient += A;

 diffuse += D;

 spec += S;

 ComputeSpotLight(gMaterial, gSpotLight, pin.PosW, pin.NormalW, toEyeW, A, D, S);

 ambient += A;

 diffuse += D;

 spec += S;

 float4 litColor = ambient + diffuse + spec;

 // 通常从漫反射材质中提取alpha

 litColor.a = gMaterial.Diffuse.a;

 return litColor;

}

technique11 LightTech

{

 pass P0

 {

 SetVertexShader( CompileShader( vs\_5\_0, VS() ) );

 SetGeometryShader( NULL );

 SetPixelShader( CompileShader( ps\_5\_0, PS() ) );

 }

}

## 7.13.2 C++程序代码

光照计算需要表面法线的信息。我们是在顶点层次定义法线的，然后对法线进行插值用于逐像素光照计算。此外，现在也不需要指定顶点颜色了，表面的颜色可以代入光照方程逐像素地求得。输入布局描述如下所示：

D3D11\_INPUT\_ELEMENT\_DESC vertexDesc[] =

{

 {"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

 {"NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 12,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0}

};

在应用程序中，我们定义了三个光源和两个材质。

DirectionalLight mDirLight;

PointLight mPointLight;

SpotLight mSpotLight;

Material mLandMat;

Material mWavesMat;

它们在构造函数中进行初始化：

LightingApp::LightingApp(HINSTANCE hInstance)

: D3DApp(hInstance), mLandVB(0), mLandIB(0), mWavesVB(0), mWavesIB(0),

 mFX(0), mTech(0), mfxWorld(0), mfxWorldInvTranspose(0), mfxEyePosW(0),

 mfxDirLight(0), mfxPointLight(0), mfxSpotLight(0), mfxMaterial(0),

 mfxWorldViewProj(0),

 mInputLayout(0), mEyePosW(0.0f, 0.0f, 0.0f), mTheta(1.5f\*MathHelper::Pi), mPhi(0.1f\*MathHelper::Pi), mRadius(80.0f)

{

 …

 // Directional light.

 mDirLight.Ambient = XMFLOAT4(0.2f, 0.2f, 0.2f, 1.0f);

 mDirLight.Diffuse = XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f);

 mDirLight.Specular = XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f);

 mDirLight.Direction = XMFLOAT3(0.57735f, -0.57735f, 0.57735f);

 // Point light--position is changed every frame to animate in UpdateScene function.

 mPointLight.Ambient = XMFLOAT4(0.3f, 0.3f, 0.3f, 1.0f);

 mPointLight.Diffuse = XMFLOAT4(0.7f, 0.7f, 0.7f, 1.0f);

 mPointLight.Specular = XMFLOAT4(0.7f, 0.7f, 0.7f, 1.0f);

 mPointLight.Att = XMFLOAT3(0.0f, 0.1f, 0.0f);

 mPointLight.Range = 25.0f;

 // Spot light--position and direction changed every frame to animate in UpdateScene function.

 mSpotLight.Ambient = XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f);

 mSpotLight.Diffuse = XMFLOAT4(1.0f, 1.0f, 0.0f, 1.0f);

 mSpotLight.Specular = XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f);

 mSpotLight.Att = XMFLOAT3(1.0f, 0.0f, 0.0f);

 mSpotLight.Spot = 96.0f;

 mSpotLight.Range = 10000.0f;

 mLandMat.Ambient = XMFLOAT4(0.48f, 0.77f, 0.46f, 1.0f);

 mLandMat.Diffuse = XMFLOAT4(0.48f, 0.77f, 0.46f, 1.0f);

 mLandMat.Specular = XMFLOAT4(0.2f, 0.2f, 0.2f, 16.0f);

 mWavesMat.Ambient = XMFLOAT4(0.137f, 0.42f, 0.556f, 1.0f);

 mWavesMat.Diffuse = XMFLOAT4(0.137f, 0.42f, 0.556f, 1.0f);

 mWavesMat.Specular = XMFLOAT4(0.8f, 0.8f, 0.8f, 96.0f);

}

当使用多光源时，必须要小心别让颜色超出范围。所以你需要测试环境光、漫反射光和镜面光的强度，对光照范围和衰减的测试也是必须的。有时很容易忘记使用带颜色的灯光，但是它们可以被用来产生不同的效果。例如，你在一个太阳模型上使用平行光，太阳本身被设置为橘黄色，你可以将光的颜色也设置为橘黄色，这样，场景中的物体都会带上橘黄色的色调。异形飞船爆炸时可以加点淡蓝色，红光通常表示紧急情况。

如前所述，点光源和聚光灯是活动的；这一工作在**UpdateScene**方法中实现：

void LightingApp::UpdateScene(float dt)

{

 …

 //

 // 光源动画

 //

 // 让点光源在地面之上转圈

 mPointLight.Position.x = 70.0f\*cosf( 0.2f\*mTimer.TotalTime() );

 mPointLight.Position.z = 70.0f\*sinf( 0.2f\*mTimer.TotalTime() );

 mPointLight.Position.y = MathHelper::Max(GetHillHeight(mPointLight.Position.x,

 mPointLight.Position.z), -3.0f) + 10.0f;

 // 聚光灯的位置与观察点的位置相同，光照方向与观察方向相同；

 // 这样产生的效果就像是观察者拿着一个手电筒在场景中照来照去一样。

 mSpotLight.Position = mEyePosW;

 XMStoreFloat3(&mSpotLight.Direction, XMVector3Normalize(target - pos));

}

基本上，点光源会沿着*xz*平面上的一个圆形轨迹来运动，只是它总保持在地面和水面之上。聚光灯的位置与观察点的位置相同，光照方向与观察方向相同；这样产生的效果就像是观察者拿着一个手电筒在场景中照来照去一样。

最后，我们要在渲染之前将灯光和材质指定给effect：

void LightingApp::DrawScene()

{

 …

 // Set per frame constants.

 mfxDirLight->SetRawValue(&mDirLight, 0, sizeof(mDirLight));

 mfxPointLight->SetRawValue(&mPointLight, 0, sizeof(mPointLight));

 mfxSpotLight->SetRawValue(&mSpotLight, 0, sizeof(mSpotLight));

 mfxEyePosW->SetRawValue(&mEyePosW, 0, sizeof(mEyePosW));

 …

 D3DX11\_TECHNIQUE\_DESC techDesc;

 mTech->GetDesc( &techDesc );

 for(UINT p = 0; p < techDesc.Passes; ++p)

 {

 …

 mfxMaterial->SetRawValue(&mLandMat, 0, sizeof(mLandMat));

 /\* Draw the hills...\*/

 …

 mfxMaterial->SetRawValue(&mWavesMat, 0, sizeof(mWavesMat));

 /\* Draw waves...\*/

 }

 HR(mSwapChain->Present(0, 0));

}

## 7.13.3 法线计算

因为我们的地形表面是通过一个函数*y* = *f*(*x*,*z*)生成的，所以我们可以直接使用微积分计算法线向量，而不是7.2.1节描述的法线平均值技术。要为表面上的每个点计算法线向量，我们必须通过偏导数生成+*x*和+*z*方向上的两个正切向量：





这两个向量位于表面点的正切平面上。我们可以通过计算叉积得到正切平面的法线向量：



我们用于生成地面网格的函数为：



偏导数为：



表面点(*x*,*f*(*x*,*z*),*z*)的表面法线为：



注意，表面法线不是规范化向量，在执行照计算之前必须对它进行规范化处理。

我们只在每个顶点上执行上述法线计算，获得顶点法线：

XMFLOAT3 LightingApp::GetHillNormal(float x, float z)const

{

 // n = (-df/dx, 1, -df/dz)

 XMFLOAT3 n(

 -0.03f\*z\*cosf(0.1f\*x) - 0.3f\*cosf(0.1f\*z),

 1.0f,

 -0.3f\*sinf(0.1f\*x) + 0.03f\*x\*sinf(0.1f\*z));

 XMVECTOR unitNormal = XMVector3Normalize(XMLoadFloat3(&n));

 XMStoreFloat3(&n, unitNormal);

 return n;

}

对于水体表面来说，法线向量的计算过程基本相同，只是我们没有用于生成水体的公式。不过，读者可以使用有限差异图（finite difference scheme）估算每个顶点上的正切向量（详情请参见[Lengyel02]或任何一本关于数值分析的书籍）。

**注意**：如果你的微积分已经荒废了，那也不用担心；因为它在本书中的作用不是非常重要。我们在这里使用微积分，只是为了以数学方式生成一个曲面几何体，使我们在演示程序中能够渲染一些比较有趣的物体。在本书最后，我们将为读者讲解如何载入和使用那些由3D建模软件导出的3D网格文件。