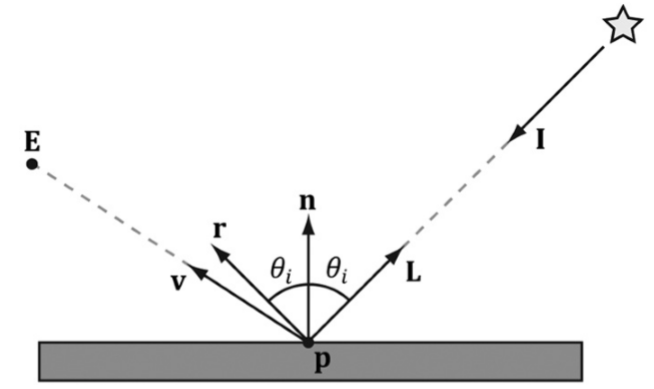
# 灯光

# 光照模型



我们使用冯氏光照模型（phong lighting），在pixel shader阶段，逐像素计算光照。

## 灯光与材质的相互作用

材质可以被认为是决定灯光如何与物体表面相互作用的属性。例如，表面反射的灯光颜色、 吸收的灯光颜色、反射率、透明度和光泽度都是构成表面材质的参数。

## 法线向量

使用法线向量来描述多边形的方向

当对点进行变换时，应使用该变换的逆转置矩阵来变换法线向量。若变换矩阵为正交矩阵，则无需额外计算法线向量的变换矩阵。注意变换后需重新normalize。

## 兰博特余弦定理

垂直照向平面的线比从侧面照向平面的线更加强烈，故计算光照强度时，需要乘以cosθ

## 漫反射光(Diffuse Lighting)

当光线照射在粗糙表面上时，会在不同的随机方向上散开，我们将这种反射称为漫反射（diffuse reflection）。故在计算时不需要考虑观察点的位置。

我们指定一个入射光颜色和一个漫反射材质颜色，使用分量颜色乘法来计算反射颜色。

例如：入射光颜色**l**d = (0.8, 0.8, 0.8)，漫反射材质颜色**m**d = (0.5, 1.0, 0.75)；反射光的总量为: **D** =**l**d ⨂ **m**d = (0.8, 0.8, 0.8)⨂(0.5, 1.0, 0.75) = (0.4, 0.8, 0.6)

之后再使用兰伯特余弦定理计算光照强度。

**c**d = kd·**l**d ⨂ **m**d = kd**D**

## 环境光(Ambient Lighting)

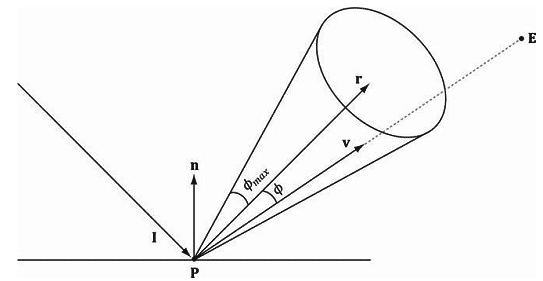
为了模拟间接光，我们在光照方程中引入了一个环境光项，通过环境材质颜色来计算。

**A**＝**l**a⨂**m**a

## 高光(Specular Lighting)

当灯照射在光滑表面上时，光线会在一个由反射系数描述的圆锥体区域内形成锐利的反射；我们将这种反射称为镜面高光反射（specular reflection，或直译为镜面反射）。

与漫反射不同，高光可能不会传入眼睛，因为它只在一个特定的方向上反射。高光的计算过程与观察点的位置相关。



当v在圆锥体内时，我们可以看到高光；反之，看不到高光。v与r的夹角越小，我们看到的高光就越强。

我们引入参数p来控制表面的平滑程度

**c**s = *k*s·**l**s ⨂ **m**s = ks**S**

其中

_scroll_external/attachments/image2020-11-23_18-12-58-425e09b897bdac53f14a4e37c0028b92f9cdb2ccb3b37c4f01fc30ff438639a6.png

注意cos*ϕ* = **v**·**r**

总光照方程：

 LitColor = **l**a⨂**m**a + **k**d·**l**d ⨂ **m**d + ks·**l**s ⨂ **m**s = **A** + kd**D** + ks**S**

## 摘要

在我们的模型中，光源可以发射3种不同类型的线：

1．环境光（ambient light）：模拟间接光照。

2．漫反射光（diffuse light）：模拟对粗糙表面的直接照。

3．高光（specular light）：模拟对光滑表面的直接光照。

同样，物体表面有以下材质属性与其对应：

1．环境材质：平面反射和吸收的环境光的总量。

2．漫反射材质：平面反射和吸收的漫反射光的总量。

3．高光材质：平面反射和吸收的高光的总量。

4．高光指数：它是在高光计算中使用的一个指数，它通过一个由反射系数描述的圆锥体区域来控制表面的光滑程度。圆锥体越小，表面越平滑/光亮。

Example:

|  |
| --- |
| **struct** Material  {  Material() { ZeroMemory(**this**, **sizeof**(**this**)); }    XMFLOAT4 Ambient;  XMFLOAT4 Diffuse;  XMFLOAT4 Specular; // w = SpecPower  XMFLOAT4 Reflect;  }; |

# 光源

## 平行光（parallel light ）

平行光由一个描述光线传播方向的向量来表示。因为它产生的光线是平行的，所以所有光线都使用相同的方向向量。光照向量与平行光的传播方向相反。

## 点光（point light）

在现实生活中，最常见的点光源是灯泡；它可以向各个方向发射光线。

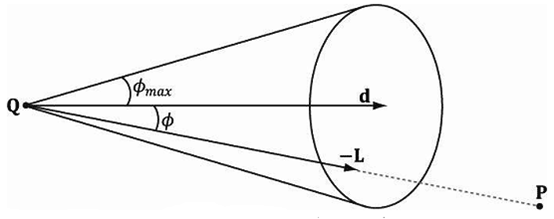
按物理规律来说，光照强度会随着距离的增加而衰减，用于调节灯光强度的典型公式为：

_scroll_external/attachments/image2020-11-23_18-22-43-33f7de2a30b542ff3323a460e93b156d9c7d7f0cc8b5e3aafc88b127284469f6.png

我们也可以引入一个附加的范围参数。当一个点与点光源之间的距离大于指定的范围时，使它不接收该光源的照射。

## 聚光灯（spotlight）

在现实生活中，最常见的聚光灯是手电筒。本质上，聚光灯由一个位置**Q**、一个方向向量**d**和一个圆锥体光照区域来描述



与高光类似，我们在圆锥体内计算光照，然后通过夹角计算强度。

Example:

|  |
| --- |
| **struct** DirectionalLight  {  DirectionalLight() { ZeroMemory(**this**, **sizeof**(**this**)); }    XMFLOAT4 Ambient;  XMFLOAT4 Diffuse;  XMFLOAT4 Specular;  XMFLOAT3 Direction;  float Pad; // Pad the last float so we can set an array of lights if we wanted.  };    **struct** PointLight  {  PointLight() { ZeroMemory(**this**, **sizeof**(**this**)); }    XMFLOAT4 Ambient;  XMFLOAT4 Diffuse;  XMFLOAT4 Specular;    // Packed into 4D vector: (Position, Range)  XMFLOAT3 Position;  float Range;    // Packed into 4D vector: (A0, A1, A2, Pad)  XMFLOAT3 Att;  float Pad; // Pad the last float so we can set an array of lights if we wanted.  };    **struct** SpotLight  {  SpotLight() { ZeroMemory(**this**, **sizeof**(**this**)); }    XMFLOAT4 Ambient;  XMFLOAT4 Diffuse;  XMFLOAT4 Specular;    // Packed into 4D vector: (Position, Range)  XMFLOAT3 Position;  float Range;    // Packed into 4D vector: (Direction, Spot)  XMFLOAT3 Direction;  float Spot;    // Packed into 4D vector: (Att, Pad)  XMFLOAT3 Att;  float Pad; // Pad the last float so we can set an array of lights if we wanted.  }; |

# Shader节选

PixelShader

|  |
| --- |
| // Start with a sum of zero.  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);    // Sum the light contribution from each light source.  [unroll]  **for**(int i = 0; i < gLightCount; ++i)  {  float4 A, D, S;  ComputeDirectionalLight(gMaterial, gDirLights[i], pin.NormalW, toEye,  A, D, S);    ambient += A;  diffuse += D;  spec += S;  }    float4 litColor = ambient + diffuse + spec;    // Common to take alpha from diffuse material.  litColor.a = gMaterial.Diffuse.a; |

ComputeDirectionalLight

|  |
| --- |
| //---------------------------------------------------------------------------------------  // Computes the ambient, diffuse, and specular terms in the lighting equation  // from a directional light. We need to output the terms separately because  // later we will modify the individual terms.  //---------------------------------------------------------------------------------------  **void** ComputeDirectionalLight(Material mat, DirectionalLight L,  float3 normal, float3 toEye,  out float4 ambient,  out float4 diffuse,  out float4 spec)  {  // Initialize outputs.  ambient = float4(0.0f, 0.0f, 0.0f, 0.0f);  diffuse = float4(0.0f, 0.0f, 0.0f, 0.0f);  spec = float4(0.0f, 0.0f, 0.0f, 0.0f);    // The light vector aims opposite the direction the light rays travel.  float3 lightVec = -L.Direction;    // Add ambient term.  ambient = mat.Ambient \* L.Ambient;    // Add diffuse and specular term, provided the surface is in  // the line of site of the light.    float diffuseFactor = dot(lightVec, normal);    // Flatten to avoid dynamic branching.  [flatten]  **if**( diffuseFactor > 0.0f )  {  float3 v = reflect(-lightVec, normal);  float specFactor = pow(max(dot(v, toEye), 0.0f), mat.Specular.w);    diffuse = diffuseFactor \* mat.Diffuse \* L.Diffuse;  spec = specFactor \* mat.Specular \* L.Specular;  }  } |