备忘录_光照

笔记本:	DirectX 12		
创建时间:	2022/8/15 10:47	**更新时间:**	2022/8/16 18:53
作者:	handsome小赞		

- **法向量**:

> **phong光照模型/逐像素光照模型**：对每个像素逐一进行法线插值并执行光照计算的 方法。
> **逐顶点光照模型**：对每个顶点逐一进行光照计算的方法。
> **变换法向量**： 对点或向量（非法线）进行一个非等比缩放变换A，那么就需要对法向量进行变换。

$\boldsymbol{u}\cdot\boldsymbol{n}=0$	切向量正交于法向量
$\boldsymbol{u}\boldsymbol{n}^{\mathrm{T}}=0$	将点积改写为矩阵乘法
$\boldsymbol{u}(\boldsymbol{A}\boldsymbol{A}^{-1})\boldsymbol{n}^{\mathrm{T}}=0$	插入单位矩阵 $\boldsymbol{I}=\boldsymbol{A}\boldsymbol{A}^{-1}$
$(\boldsymbol{u}\boldsymbol{A})(\boldsymbol{A}^{-1}\boldsymbol{n}^{\mathrm{T}})=0$	根据矩阵乘法运算的结合律
$(\boldsymbol{u}\boldsymbol{A})((\boldsymbol{A}^{-1}\boldsymbol{n}^{\mathrm{T}})^{\mathrm{T}})^{\mathrm{T}}=0$	根据转置矩阵的性质$(\boldsymbol{A}^{\mathrm{T}})^{\mathrm{T}}=\boldsymbol{A}$
$(\boldsymbol{u}\boldsymbol{A})(\boldsymbol{n}(\boldsymbol{A}^{-1})^{\mathrm{T}})^{\mathrm{T}}=0$	根据转置矩阵的性质$(\boldsymbol{A}\boldsymbol{B})^{\mathrm{T}}=\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}$
$\boldsymbol{u}\boldsymbol{A}\cdot\boldsymbol{n}(\boldsymbol{A}^{-1})^{\mathrm{T}}=0$	将矩阵乘法改写为点积的形式
$\boldsymbol{u}\boldsymbol{A}\cdot\boldsymbol{n}\boldsymbol{B}=0$	变换后的切向量正交于变换后的法向量

> 在通过逆转置矩阵（B）对向量进行变换时，我们可以将向量变换矩阵中与平移有关的 项清零，而只允许点类才有平移变换。虽然在齐次坐标下，通过将向量的第四个分量 设为 w=0 就可以防止向量因平移而受影响，但这里逆转置矩阵可能与另一个不含非 等比缩放的矩阵相连接，如观察矩阵 BV，B 中经转置后的第4列将导致错误。变换法 线所采用的正确公式实则为 $((\boldsymbol{A}\boldsymbol{V})^{-1})^{\mathrm{T}}$

- **光照计算**:

> - 关键向量：法向量 n、入射光向量 I、光向量 L、观察向量（E-p）、反射向量 r
> - 反射向量：r = I - 2(n * I)n 在着色器内是利用内置函数reflect来计算r

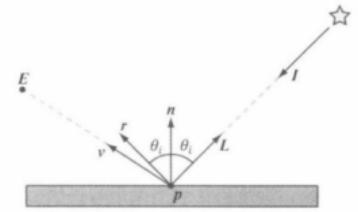

图 8.8 光照计算中所需要的各种重要向量

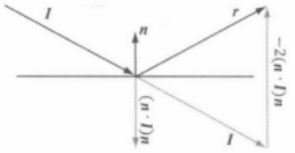

图 8.9 光线的反射示意图

- **朗伯余弦定律**:

> 辐射通量: 光源每秒发出的能量（ P ）
> 辐射照度: 单位面积上的辐射通量密度（ E = P/A ）(A为面积)
> cosθ = A1/A2 或 = n * l
> E2 = E1 * cosθ
> 最后，用max函数来限制缩放因子的范围（背光）
> max(cosθ, 0)

- **漫反射光照**

> 入射光量 B、漫反射反照率 m、光向量 L、表面法线 n
> 漫反射反照率 m: 根据表面的漫反射率而被反射的入射光量
>
> $$c = \max(L \cdot n , 0) \cdot B \otimes m$$
>
> 反射光量:

- **环境光照**

> $$c = A \otimes m$$
>
> 环境光:
> 颜色A 指定了表面收到的间接光量。
> 漫反射反照率 m

- **镜面反射**

> - 菲涅尔效应: 反射光量取决于材质（RF(0°)）以及法线与光向量的夹角 RF 反射光量、（1-RF）折射光量。RF是一个RGB向量。因为菲涅尔方程的复杂性，所以一般采用石里克近似法来代替，近似的计算出反射光的百分比:
>
> $$\boldsymbol{R}_F(\theta_i) = \boldsymbol{R}_F(0°) + (1 - \boldsymbol{R}_F(0°))(1 - \cos\theta_i)^5$$
>
> - **表面粗糙度**
> 微平面：h为法线、光线从L反射至v
> 较流行的一种可控函数:
>
> $$\rho(\theta_h) = \cos^m(\theta_h)$$
> $$= \cos^m(\boldsymbol{n} \cdot \boldsymbol{h})$$
>
> 将它与某种归一化因子组合，获得基于粗糙度来模拟镜像反射光量的函数:
>
> $$S(\theta_h) = \frac{m+8}{8}\cos^m(\theta_h)$$
> $$= \frac{m+8}{8}(\boldsymbol{n} \cdot \boldsymbol{h})^m$$
>
> 根据粗糙度与菲涅尔效应，这段镜面反射到观察者眼中的实际光量:
>
> $$\boldsymbol{c}_s = \max(\boldsymbol{L} \cdot \boldsymbol{n}, 0) \cdot \boldsymbol{B}_L \otimes \boldsymbol{R}_F(\alpha_h)\frac{m+8}{8}(\boldsymbol{n} \cdot \boldsymbol{h})^m$$

- **光照模型**

- **光照方程**（较流行的一种）：
  环境光Ca：模拟经表面反射的间接光量
  漫反射光Cd：对进入介质内部，又经表面下吸收而最终散射出表面的光进行模拟。由于对表面下的散射光建模比较困难，我们便假设在表面下与介质相互作用后的光进入表面处返回，并向各个方向均匀散射。
  镜面光Cs：模拟经菲涅尔效应与表面粗糙度共同作用的表面反射光

$$\begin{aligned}\text{LitColor} &= \boldsymbol{c}_{\text{a}} + \boldsymbol{c}_{\text{d}} + \boldsymbol{c}_{\text{s}} \\ &= \boldsymbol{A}_{\text{L}} \otimes \boldsymbol{m}_{\text{d}} + \max(\boldsymbol{L}\cdot\boldsymbol{n}, 0)\cdot \boldsymbol{B}_{\text{L}} \otimes \left(\boldsymbol{m}_{\text{d}} + \boldsymbol{R}_{F}(\alpha_{h}) \frac{m+8}{8} (\boldsymbol{n}\cdot\boldsymbol{h})^{m} \right)\end{aligned}$$

设式（8.4）中的所有向量均为单位长度。

1. $\boldsymbol{L}$：指向光源的光向量。
2. $\boldsymbol{n}$：表面法线。
3. $\boldsymbol{h}$：列于光向量与观察向量（由表面点指向观察点的单位向量）之间的中间向量。
4. $\boldsymbol{A}_{L}$：表示入射的环境光量。
5. $\boldsymbol{B}_{L}$：表示入射的直射光量。
6. $\boldsymbol{m}_{d}$：指示根据表面漫反射率而反射的入射光量。
7. $\boldsymbol{L}\cdot\boldsymbol{n}$：朗伯余弦定律。
8. α_{h}：中间向量 $\boldsymbol{h}$ 与光向量 $\boldsymbol{L}$ 之间的夹角。
9. $\boldsymbol{R}_{F}(\alpha_{h})$：根据菲涅耳效应，关于中间向量 $\boldsymbol{h}$ 所反射到观察者眼中的光量。
10. m：控制表面的粗糙度。
11. $(\boldsymbol{n}\cdot\boldsymbol{h})^{m}$：指定法线 $\boldsymbol{h}$ 与宏观表面法线 $\boldsymbol{n}$ 之间夹角为 θ_{h} 的所有微平面片段。
12. $\frac{m+8}{8}$：在镜面反射过程中，为模拟能量守恒所采用的归一化因子。

- **材质的实现**

利用常量缓冲区

```
struct Material
{
  // 便于查找材质的唯一对应名称
  std::string Name;

  // 本材质的常量缓冲区索引
  int MatCBIndex = -1;

  // 漫反射纹理在 SRV 堆中的索引。在第 9 章纹理贴图时会用到
  int DiffuseSrvHeapIndex = -1;

  // 已更新标志（dirty flag，也作脏标志）表示本材质已有变动，而我们也就需要更新常量缓冲区了。
  // 由于每个帧资源 FrameResource 都有一个材质常量缓冲区，所以必须对每个 FrameResource 都进
  // 行更新。因此，当修改某个材质时，应当设置 NumFramesDirty = gNumFrameResources，以使每
  // 个帧资源都能得到更新
  int NumFramesDirty = gNumFrameResources;

  // 用于着色的材质常量缓冲区数据
  DirectX::XMFLOAT4 DiffuseAlbedo = { 1.0f, 1.0f, 1.0f, 1.0f };
  DirectX::XMFLOAT3 FresnelR0 = { 0.01f, 0.01f, 0.01f };
  float Roughness = 0.25f;
  DirectX::XMFLOAT4X4 MatTransform = MathHelper::Identity4x4();
};
```

材质在不同表面而发生变化，更普遍的方法是采用纹理贴图来实现。
使用顶点也可以，它将在三角形的光栅化处理期间进行插值，但是很粗糙，而且绘制每个顶点颜色还需要向顶点结构体中添加额外的数据，还需要添加额外的属性。

> 在项目示例中，允许在绘制调用时对材质进行频繁地更改。只要更改就标脏，并对upload缓冲区进行复制。每个渲染项都有指向Material结构体地指针。
>
> 1. 创建材质的映射表(unordered_map)->
> 2. 存放于系统内存并将关键数据复制到常量缓冲区(帧资源的)中->
> 3. 当材质数据有了变化则标脏，进行数据同步->

- **光源**

> - **方向光源**
> - **点光源**
>
>   衰减
>
> $$\text{att}(d) = \text{saturate}\left(\frac{\text{falloffEnd} - d}{\text{falloffEnd} - \text{falloffStart}}\right)$$
>
>   线性衰减函数：
>
>   与得到的衰减因子 att(d) 还须和光源的直射光量BL相乘
>
>   为了优化，在着色器程序中，如果一个点超过了光照的有效范围，那么就可以采用动态分支，跳过此处的光照计算并提前返回。
> - **聚光灯光源**
>
>   光照方向 d
>
> $$k_{spot}(\phi) = \max(\cos\phi, 0)^s = \max(-\boldsymbol{L}\cdot\boldsymbol{d}, 0)^s$$
>
>   损耗 平行光源 < 点光源 < 聚光灯光源
> - **光照的具体实现**
>
> ```
> struct Light
> {
>   DirectX::XMFLOAT3 Strength = {0.5f, 0.5f, 0.5f};    // 光源的颜色
>   float FalloffStart = 1.0f;                          // 仅供点光源/聚光灯光源使用
>   DirectX::XMFLOAT3 Direction = {0.0f, -1.0f, 0.0f}; // 仅供方向光源/聚光灯光源使用
>   float FalloffEnd = 10.0f;                           // 仅供点光源/聚光灯光源使用
>   DirectX::XMFLOAT3 Position = { 0.0f, 0.0f, 0.0f }; // 仅供点光源/聚光灯光源使用
>   float SpotPower = 64.0f;                            // 仅供聚光灯光源使用
> };
> ```
>
> 文件 LightingUtil.hlsl 中则定义了与之对应的结构体：
>
> ```
> struct Light
> {
>   float3 Strength;
>   float FalloffStart; // 仅供点光源/聚光灯光源使用
>   float3 Direction;   // 仅供方向光源/聚光灯光源使用
>   float FalloffEnd;   // 仅供点光源/聚光灯光源使用
>   float3 Position;    // 仅供点光源/聚光灯光源使用
>   float SpotPower;    // 仅供聚光灯光源使用
> };
> ```
>
> **注意：结构体的数据成员排列顺序需要遵从HLSL的结构体封装规则。**
>
> **HLSL所用皆是内联函数，所以函数或传递参数不会有过多的性能开销**

- **实现方向光源**

```
float3 ComputeDirectionalLight(Light L, Material mat, float3 normal, float3 toEye)
{
    // The light vector aims opposite the direction the light rays travel.
    // 光向量与光线传播方向刚好相反
    float3 lightVec = -L.Direction;

    // Scale light down by Lambert's cosine law.
    // 通过朗伯余弦定律按比例降低光强
    float ndotl = max(dot(lightVec, normal), 0.0f);
    float3 lightStrength = L.Strength * ndotl;

    return BlinnPhong(lightStrength, lightVec, normal, toEye, mat);
}
```

- **实现点光源**

```
float3 ComputePointLight(Light L, Material mat, float3 pos, float3 normal, float3 toEye)
{
    // The vector from the surface to the light.
    // 自表面指向光源的向量
    float3 lightVec = L.Position - pos;

    // The distance from surface to light.
    // 由表面到光源的距离
    float d = length(lightVec);

    // Range test.
    // 范围检测
    if(d > L.FalloffEnd)
        return 0.0f;

    // Normalize the light vector.
    // 对光向量进行规范化处理
    lightVec /= d;

    // Scale light down by Lambert's cosine law.
    // 通过朗伯余弦定律按比例降低光强
    float ndotl = max(dot(lightVec, normal), 0.0f);
    float3 lightStrength = L.Strength * ndotl;

    // Attenuate light by distance.
    // 根据距离计算光的衰减
    float att = CalcAttenuation(d, L.FalloffStart, L.FalloffEnd);
    lightStrength *= att;

    return BlinnPhong(lightStrength, lightVec, normal, toEye, mat);
}
```

- **实现聚光灯光源**

```
float3 ComputeSpotLight(Light L, Material mat, float3 pos, float3 normal, float3 toEye)
{
    // The vector from the surface to the light.
    // 从表面射向光源的向量
    float3 lightVec = L.Position - pos;

    // The distance from surface to light.
    // 由表面到光源的距离
    float d = length(lightVec);

    // Range test.
    // 范围检测
    if(d > L.FalloffEnd)
        return 0.0f;

    // Normalize the light vector.
    // 对光向量进行规范化处理
    lightVec /= d;

    // Scale light down by Lambert's cosine law.
    // 通过朗伯余弦定律按比例缩小光的强度
    float ndotl = max(dot(lightVec, normal), 0.0f);
    float3 lightStrength = L.Strength * ndotl;

    // Attenuate light by distance.
    // 通过距离计算光的衰减
    float att = CalcAttenuation(d, L.FalloffStart, L.FalloffEnd);
    lightStrength *= att;

    // Scale by spotlight
    // 根据聚光灯光照模型对光强进行缩放处理
    float spotFactor = pow(max(dot(-lightVec, L.Direction), 0.0f), L.SpotPower);
    lightStrength *= spotFactor;

    return BlinnPhong(lightStrength, lightVec, normal, toEye, mat);
}
```