

# 5 PBR

## 5-0 变量定义

```
// 变量定义
float3 T = normalize(i.tangent); // 计算切线向量，并将其标准化，确保长度为1
float3 N = normalize(i.normal); // 计算法线向量，并将其标准化，确保长度为1
float3 B = normalize(cross(N, T)); // 计算副切线向量，通过叉乘法与法线向量和切线向量正交，并将其标准化，确保长度为1

// float3 B = normalize(i.bitangent); // 注释掉上面的计算方式，使用预计算的副切线向量
float3 L =
normalize(UnityWorldSpaceLightDir(i.worldPosition.xyz)); // 计算世界空间中的光照方向向量，并将其标准化，确保长度为1
float3 V =
normalize(UnityWorldSpaceViewDir(i.worldPosition.xyz)); // 计算世界空间中的视线方向向量，并将其标准化，确保长度为1
float3 H = normalize(V + L); // 计算半程向量（视线向量与光照方向向量之和），并将其标准化，确保长度为1，用于计算镜面反射
float2 uv = i.uv; // 获取顶点的纹理坐标，用于采样纹理进行着色

//================================================================ Normal Map =====
// 
// 从法线贴图中解包得到法线向量
float3 NormalMap = UnpackNormal(tex2D(_NormalTex, uv));
// UnpackNormal 函数用于从法线贴图中解包法线向量，_NormalTex 是法线贴图的采样器，uv 是当前顶点的纹理坐标

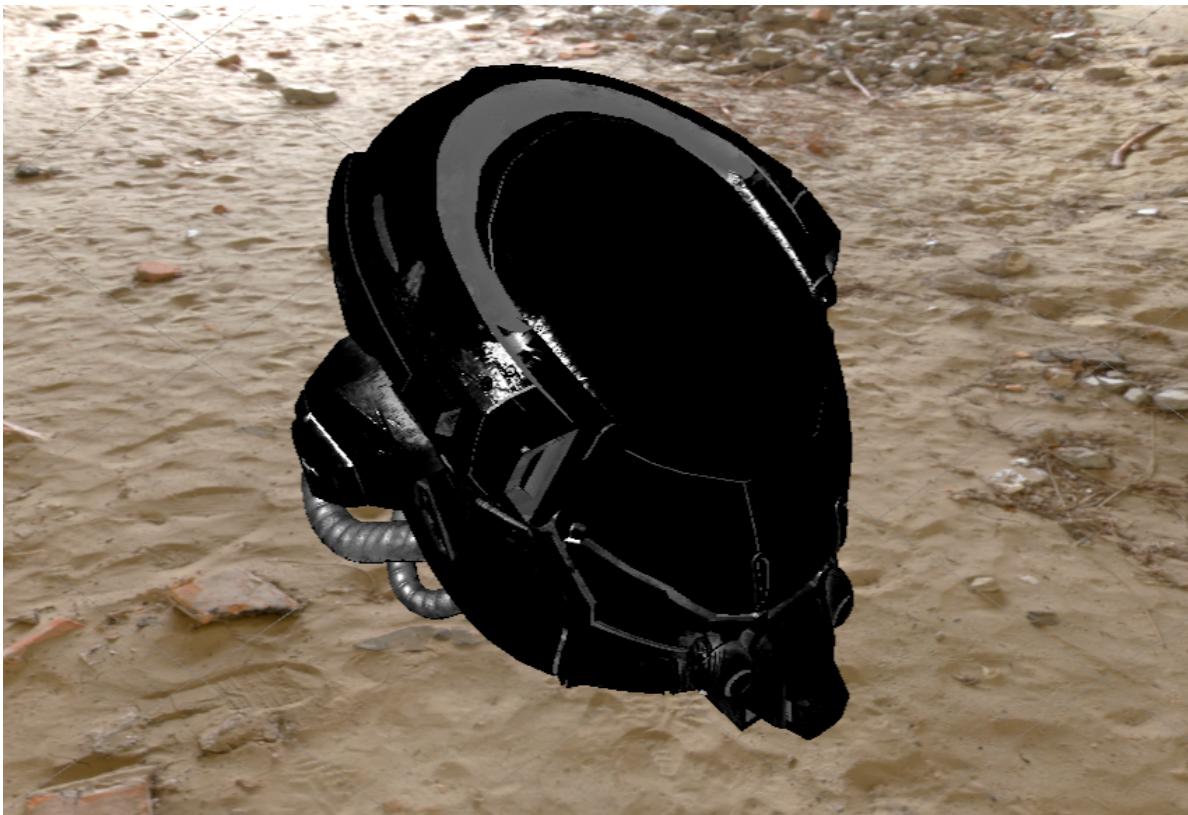
// 构建 TBN 矩阵
float3x3 TBN = float3x3(T, B, N);
// T、B、N 分别为切线、副切线和法线向量，构成了 TBN 矩阵
// TBN 矩阵用于将法线贴图中的法线向量从切线空间转换为世界空间或视图空间

// 将法线向量从切线空间转换为世界空间或视图空间
N = normalize(mul(NormalMap, TBN));
// 通过将法线向量与 TBN 矩阵相乘，将法线从切线空间转换为世界空间或视图空间
// 最后再对结果进行标准化，确保法线向量的长度为1

// 注：这里的 TBN 矩阵是一个正交矩阵，所以它的逆矩阵等于其转置矩阵
// 在法线贴图中存储的法线向量通常是在切线空间中，这里使用 TBN 矩阵将其转换到世界空间或视图空间
// 这样在片段着色器中就可以基于顶点的法线信息进行光照计算和表面细节处理
```

## 5-1 直接光的高光

在 PBR (Physically Based Rendering) 中，实现直接光的高光效果通常采用 Cook-Torrance BRDF (双向反射分布函数) 模型。该模型利用视线方向向量  $V$ 、光照方向向量  $L$  和法线  $N$  的夹角余弦值，以及材质的粗糙度和 Fresnel 反射率，计算出镜面高光的颜色。这样的实现能更准确地表达材质的反射特性，使 PBR 渲染效果更加真实。



```
// Specular
// 使用 Cook-Torrance BRDF 计算镜面高光
// HV 表示视线方向向量 V 和光照方向向量 L 的半角向量与法线 N 的夹角余弦值
float HV = max(dot(H, V), 0);
// NV 表示视线方向向量 V 和法线 N 的夹角余弦值
float NV = max(dot(N, V), 0);
// NL 表示法线 N 和光照方向向量 L 的夹角余弦值
float NL = max(dot(N, L), 0);

// 使用 Cook-Torrance BRDF 计算镜面高光的颜色值
// Specular_Cook_Torrance 函数用于根据光照方向向量 L、视线方向向量 V、法线 N、粗糙度 Roughness 和 Fresnel 反射率 F0 计算镜面高光颜色
float3 Specular = Specular_Cook_Torrance(L, V, N, Roughness, F0);
// return Specular.xyz;
```

## 5-2 直接光的漫反射

通过 Schlick's approximation 近似计算镜面反射率  $K_S$ ，然后根据基础颜色  $BaseColor$  和金属度  $Metallic$  计算漫反射的反射率  $K_D$ 。接着，将漫反射颜色  $Diffuse$  计算为  $K_D$  乘以  $BaseColor$ ，而没有除以  $\pi$  ( $\pi$ )。最后，计算直接光照的颜色  $DirectLight$ ，包括漫反射和之前实现的镜面高光部分  $Specular$ 。通过将  $Diffuse$  和  $Specular$  乘以光照方向向量  $L$  和法线  $N$  的夹角余弦值  $NL$ ，并乘以光源颜色  $Radiance$ ，得到最终的直接光照颜色。



```
// Diffuse
// 计算漫反射颜色
// 使用 Schlick's approximation 近似计算镜面反射率
// KS 表示视角和法线的 Schlick's approximation 镜面反射率
float3 KS = F_FresnelSchlick(HV, F0);
// KD 表示漫反射的反射率，由基础颜色 BaseColor 和金属度 Metallic 决定
float3 KD = (1 - KS) * (1 - Metallic);

// 计算漫反射颜色 Diffuse
// Diffuse = KD * BaseColor, 没有除以 PI
float3 diffuse = KD * BaseColor;

// 计算直接光照的颜色，包括漫反射和镜面高光部分
// DirectLight = (diffuse + Specular) * NL * Radiance, 其中
Radiance 表示光源颜色
//return diffuse.xyz;
float3 directLight = (diffuse + Specular) * NL * Radiance;
return DirectLight.xyz;
```

## 5-3 间接光高光

首先，根据入射向量  $V$  和表面法线  $N$  计算反射向量  $R$ ，表示光线从表面反射出去的方向。然后，使用 FresnelSchlickRoughness 函数计算间接光的菲涅尔反射项  $F_{IndirectLight}$ ，影响表面在不同视角下的镜面反射强度。接着，计算材质球谐光照环境贴图的LOD（级别） $mip$ ，用于后续环境贴图的采样。根据反射向量  $R$  和 LOD 值  $mip$ ，采样预过滤环境贴图，解码得到间接光的镜面反射采样结果  $EnvSpecularPrefiltered$ 。从 BRDF Lookup Table (LUT) 中采样环境反射的值  $env\_brdf$ ，影响间接光的镜面反射强度。最后，通过预过滤环境贴图与菲涅尔反射项  $F_{IndirectLight}$  和环境反射值  $env\_brdf$  的乘积得到间接光的镜面反射分量  $Specular_{Indirect}$ ，实现了间接光的高光效果。



```
float3 IndirectLight = 0;

    //Specular
    // 计算反射向量 R
    // 使用 reflect 函数，根据入射向量 v 和表面法线 N 计算出反射向量 R
    float3 R = reflect(-v, N);
    // R 表示表面的反射向量，即光线从表面反射出去的方向

    // 使用 FresnelSchlickRoughness 函数计算间接光的菲涅尔反射项
F_IndirectLight
    // FresnelSchlickRoughness 函数用于计算基于 schlick 近似和粗糙度的菲涅尔反射项
    // F_IndirectLight 表示间接光的菲涅尔反射项，影响表面在不同视角下的镜面反射强度
    float3 F_IndirectLight = FresnelSchlickRoughness(NV, F0,
Roughness);

    // 计算材质球谐光照环境贴图的LOD（级别）
    // Roughness 控制 LOD 的计算，使其在粗糙度较小时增加，较大时减小
    // mip 表示计算得到的 LOD 值，用于后续环境贴图的采样
    float mip = Roughness * (1.7 - 0.7 * Roughness) *
UNITY_SPECCUBE_LOD_STEPS;
    // UNITY_SPECCUBE_LOD_STEPS 为预定义常量，表示环境贴图的LOD级别数量

    // 根据反射向量 R 和 LOD 值 mip，采样预过滤环境贴图，并将结果存储到
rgb_mip
    // 这里使用 UNITY_SAMPLE_TEXCUBE_LOD 函数进行贴图采样
    float4 rgb_mip = UNITY_SAMPLE_TEXCUBE_LOD(unity_SpecCube0, R,
mip);

    // 解码采样得到的 HDR 贴图值，得到间接光的镜面反射采样结果
EnvSpecularPrefiltered
    // 这里使用 DecodeHDR 函数解码采样结果，得到实际的 HDR 线性空间颜色值
    //
```

```

float3 EnvSpecularPrefiltered = DecodeHDR(rgb_mip,
unity_SpecCube0_HDR);
//return EnvSpecularPrefiltered.xyzz;

// 从 BRDF Lookup Table (LUT) 中采样环境反射的值 env_brdf
// 这里使用 EnvBRDFApprox 函数进行数值近似采样
// env_brdf 表示环境反射的值，影响间接光的镜面反射强度
float2 env_brdf = EnvBRDFApprox(Roughness, NV);

// 计算间接光的镜面反射分量 Specular_Indirect
// Specular_Indirect 为预过滤环境贴图与菲涅尔反射项 F_IndirectLight 和
环境反射值 env_brdf 的乘积
// 最终得到间接光的镜面反射效果
float3 Specular_Indirect = EnvSpecularPrefiltered *
(F_IndirectLight * env_brdf.r + env_brdf.g);

return Specular_Indirect.xyzz;

```

## 5-4 间接光漫反射

首先，通过将直接光中的镜面反射项  $F_{\text{IndirectLight}}$  从 1 中减去，得到漫反射项的权重  $KD_{\text{IndirectLight}}$ ，表示光线被表面吸收的部分。然后，将  $KD_{\text{IndirectLight}}$  乘以  $(1 - Metallic)$  的值，控制金属度对漫反射的影响。当  $Metallic$  较小时， $KD_{\text{IndirectLight}}$  保留较大的权重，表现为较强的漫反射效果；当  $Metallic$  较大时， $KD_{\text{IndirectLight}}$  减小，表现为更多的光线被表面吸收，更强的镜面反射效果。接着，使用  $\text{ShadeSH9}$  函数计算光照的球谐函数值  $\text{irradianceSH}$ ，这些系数可以用来近似计算表面在各个方向上的漫反射光照。最后，计算间接光的漫反射分量  $\text{Diffuse_Indirect}$ ，为球谐函数值与基础颜色  $\text{BaseColor}$  的乘积，并乘以  $KD_{\text{IndirectLight}}$ ，得到表面在各个方向上的漫反射光照，实现了间接光的漫反射效果。



```
// 计算间接光的漫反射项
```

```
// 通过将直接光中的镜面反射项 F_IndirectLight 从 1 中减去，得到漫反射项的权重
float3 KD_IndirectLight = float3(1, 1, 1) - F_IndirectLight;
// KD_IndirectLight 表示间接光中漫反射项的权重，即光线被表面吸收的部分

// 将 KD_IndirectLight 乘以 (1 - Metallic) 的值，控制金属度对漫反射的影响
// 当 Metallic 较小时，KD_IndirectLight 将保留较大的权重，表现为较强的漫反射效果
// 当 Metallic 较大时，KD_IndirectLight 将减小，表现为更多的光线被表面吸收，更强的镜面反射效果
KD_IndirectLight *= 1 - Metallic;

// 使用 ShadeSH9 函数计算光照的球谐函数值
// ShadeSH9 函数基于光照环境计算出一个 9 个值的球谐函数系数
// 这些系数可以用来近似计算表面在各个方向上的漫反射光照
float3 irradiancesH = ShadeSH9(float4(N, 1));
// irradiancesH 表示光照环境的球谐函数系数

// 计算间接光的漫反射分量 Diffuse_Indirect
// Diffuse_Indirect 为球谐函数值与基础颜色 BaseColor 的乘积，并乘以 KD_IndirectLight
// 这里未除以 PI，因为在之前的计算中已经包含了 PI 的计算
// Diffuse_Indirect 表示间接光的漫反射分量，表面在各个方向上的漫反射光照
float3 diffuse_Indirect = irradiancesH * BaseColor *
KD_IndirectLight;
return Diffuse_Indirect.xyz;
```