UE4材质（UMaterial）属性：

1. UPhysicalMaterial\* PhysMaterial 物理材质
2. FColorMaterialInput DiffuseColor\_DEPRECATED 漫反射颜色（弃用）
3. FColorMaterialInput SpecularColor\_DEPRECATED 镜面反射（高光）颜色（弃用）
4. FColorMaterialInput BaseColor 底色
5. FScalarMaterialInput Metallic 金属色（标量，float）
6. FScalarMaterialInput Specular 高光（标量，float）
7. FScalarMaterialInput Roughness 粗糙度（标量，float）
8. FVectorMaterialInput Normal 法向量（向量）
9. FColorMaterialInput EmissiveColor 自发光颜色
10. FScalarMaterialInput Opacity 不透明度（标量，float）
11. FScalarMaterialInput OpacityMask 不透明度蒙版（标量，float）
12. TEnumAsByte<enum EMaterialDomain> MaterialDomain

材质域，决定了材质的使用场合，如表面、光照函数、后处理材质

1. TEnumAsByte<enum EBlendMode> BlendMode 混合模式
2. TEnumAsByte<enum EDecalBlendMode> DecalBlendMode

贴花混合模式，决定了贴花材质通道（Pass）如何操作GBuffer

1. TEnumAsByte<enum EMaterialDecalResponse> MaterialDecalResponse

材质贴花响应，决定了材质如何响应使用DBuffer的贴花

1. TEnumAsByte<enum EMaterialShadingModel> ShadingModel

着色模型

1. float OpacityMaskClipValue 不透明度蒙版裁减阈值

UE4默认光照模型 – 像素着色器关键计算逻辑

【输入】

half3 BaseColor 底色

half Metallic 金属色

half Specular 高光

float MaterialAO 材质环境遮挡（Ambient Occlusion）

float Roughness 粗糙度

half Opacity 不透明度

half3 Emissive 自发光颜色

【输出】

float4 OutColor

【计算】

漫反射颜色

half3 DiffuseColor = BaseColor – BaseColor \* Metallic

DiffuseColor \*= 1 – 0.3333 \* Roughness

镜面反射颜色

half3 SpecularColor = lerp((0.08 \* Specular.xxx), BaseColor, Metallic.xxx)

注：lerp(x, y, s)为线性插值函数，返回x至y之间的一个颜色，s越大，该颜色越接近于y

有效漫反射颜色

float3 EffectiveDiffuseColor = DiffuseColor

漫射非方向光

float3 DiffuseIndirectLighting

颜色

half3 Color += (DiffuseIndirectLighting \* EffectiveDiffuseColor) \* MaterialAO

如果开启了简单动态光照（Simple Dynamic Lighting）

{

顶点法线向量与方向光向量的点积

half Lambert =

saturate(dot(MaterialParameters.WorldNormal, View.DirectionalLightDirection))

注：saturate(x)为饱和度函数，将输入x裁减为一个[0, 1]之间的值

Color += DiffuseColor \* Lambert \* View.DirectionalLightColor.rgb

获取天空球的光传输

Color += GetMaterialHemisphereLightTransferFull(DiffuseColor,

MaterialParameters,

View.UpperSkyColor.rgb,

View.LowerSkyColor,rgb)

}

如果使用基础通道雾化（Base Pass Fogging）

{

float4 VertexFog = BasePassInterpolants.VertexFog

}

否则

{

float4 VertexFog = float4(0, 0, 0, 1)

}

应用不照明模式蒙版

Color = lerp(Color, DiffuseColor + SpecularColor, View.UnlitViewmodeMask)

Color += Emissive

如果混合模式为半透明

{

OutColor = half4(Color \* VertexFog.a + VertexFog.rgb, Opacity)

OutColor = OutColor 这一步作用不明

}

否则，如果混合模式为叠加

{

OutColor = half4(Color \* VertexFog.a \* Opacity, 0.0f)

OutColor = OutColor

}

否则，如果混合模式为乘法

{

half3 FoggedColor = lerp(float3(1, 1, 1), Color, VertexFog.aaa \* VertexFog.aaa)

OutColor = half4(FoggedColor, Opacity)

}

否则

{

FLightAccumulator LightAccumulator = 0

LightAccumulator\_Add(LightAccumulator, Color, 0, 1.0f)

OutColor = LightAccumulator\_GetResult(LightAccumulator)

注：1. FLightAccumulator在LightAccumulator.usf中定义，它是一个结构体，其定义如下

struct FLightAccumulator

{

float3 TotalSum;

float ViewIndependentLightingLuminanceSum;

float3 ViewIndependentLightingSum;

float EstimatedCost;

}

2. LightAccumulator\_Add函数定义如下

void LightAccumulator\_Add(inout FLightAccumulator In,

float3 NonSpecularContribution,

float3 SpecularContribution,

float3 CommonMultiplier)

{

In.TotalSum +=

(NonSpecularContribution + SpecularContribution) \* CommonMultiplier;

if (SUBSURFACE\_CHANNEL\_MODE == 1)

{

In.ViewIndependentLightingLuminanceSum +=

Luminance(NonSpecularContribution \* CommonMultiplier);

}

else if (SUBSURFACE\_CHANNEL\_MODE ==2)

{

In.ViewIndependentLightingSum +=

NonSpecularContribution \* CommonMultiplier;

}

}

3. LightAccumulator\_GetResult函数定义如下

float4 LightAccumulator\_GetResult(FLightAccumulator In)

{

float4 Ret;

if (VISUALIZE\_LIGHT\_CULLING == 1)

{

Ret = 0.1f \* float4(1.0f, 0.25f, 0.075f, 0) \* In.EstimatedCost;

}

else

{

Ret = float4(In.TotalSum, 0);

if (SUBSURFACE\_CHANNEL\_MODE == 1)

{

Ret.a = In.ViewIndependentLightingLuminanceSum;

}

else if (SUBSURFACE\_CHANNEL\_MODE == 2)

{

Ret.a = Luminance(In.ViewIndependentLightingSum);

}

}

return Ret;

}

}