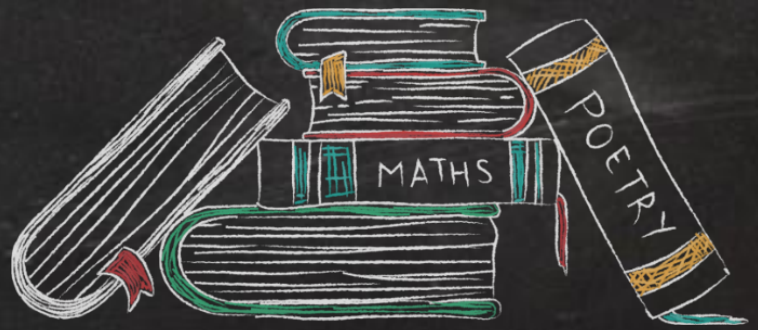
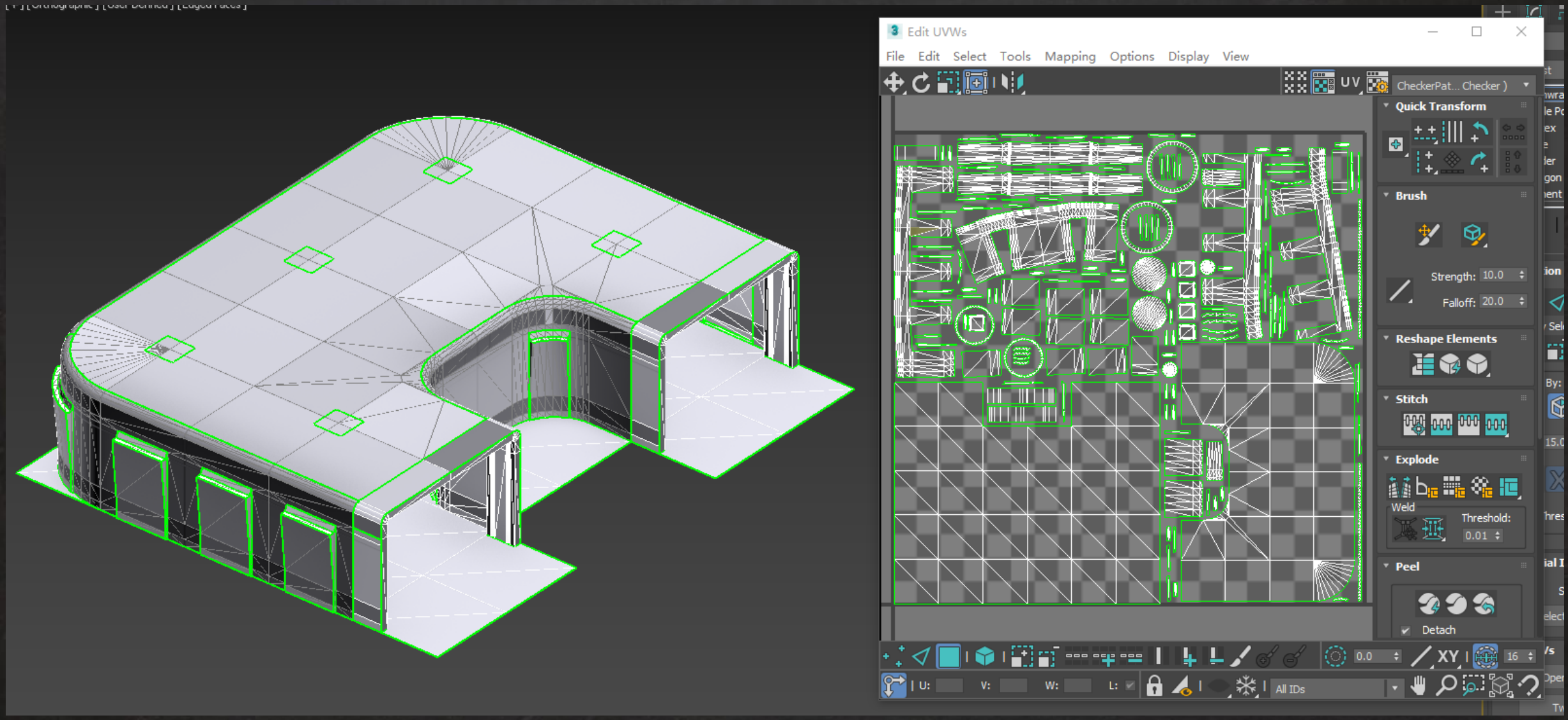
今日内容

- 忍具·筑基·Lightmap-外部
- 蒔文·化神·SeparatedLM

1 忍具·筑基·Lightmap-外部



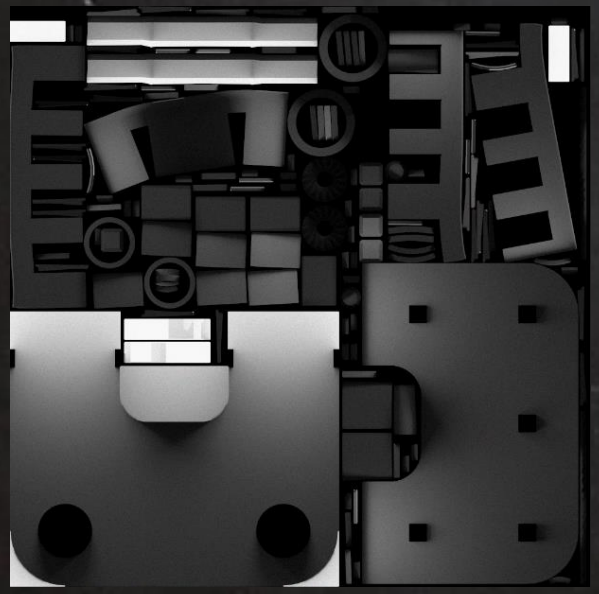
01 制作UV1(2U)

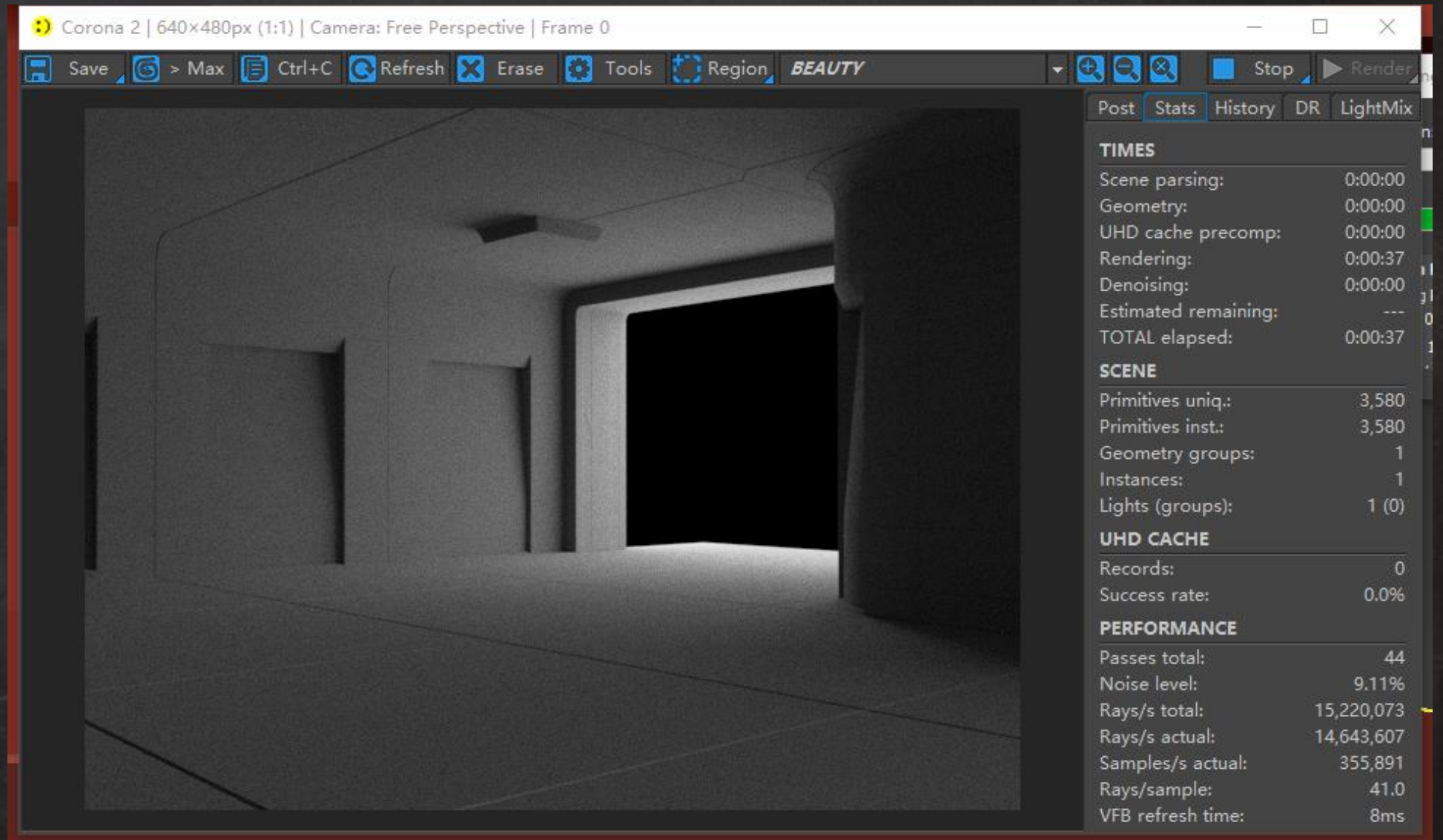


02 烘焙天光：SkyLightMask

自发光材质，将AO结点连入自发光强度，开启GI，烘焙到纹理：

- 调整为想要的照明区域；
- 明暗分布偏线性；

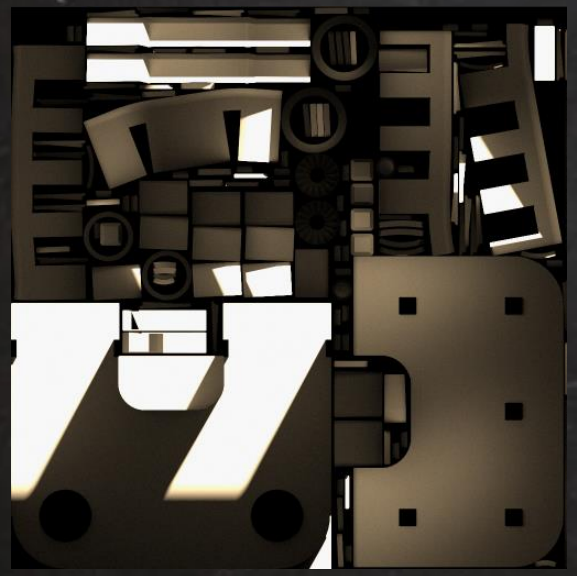


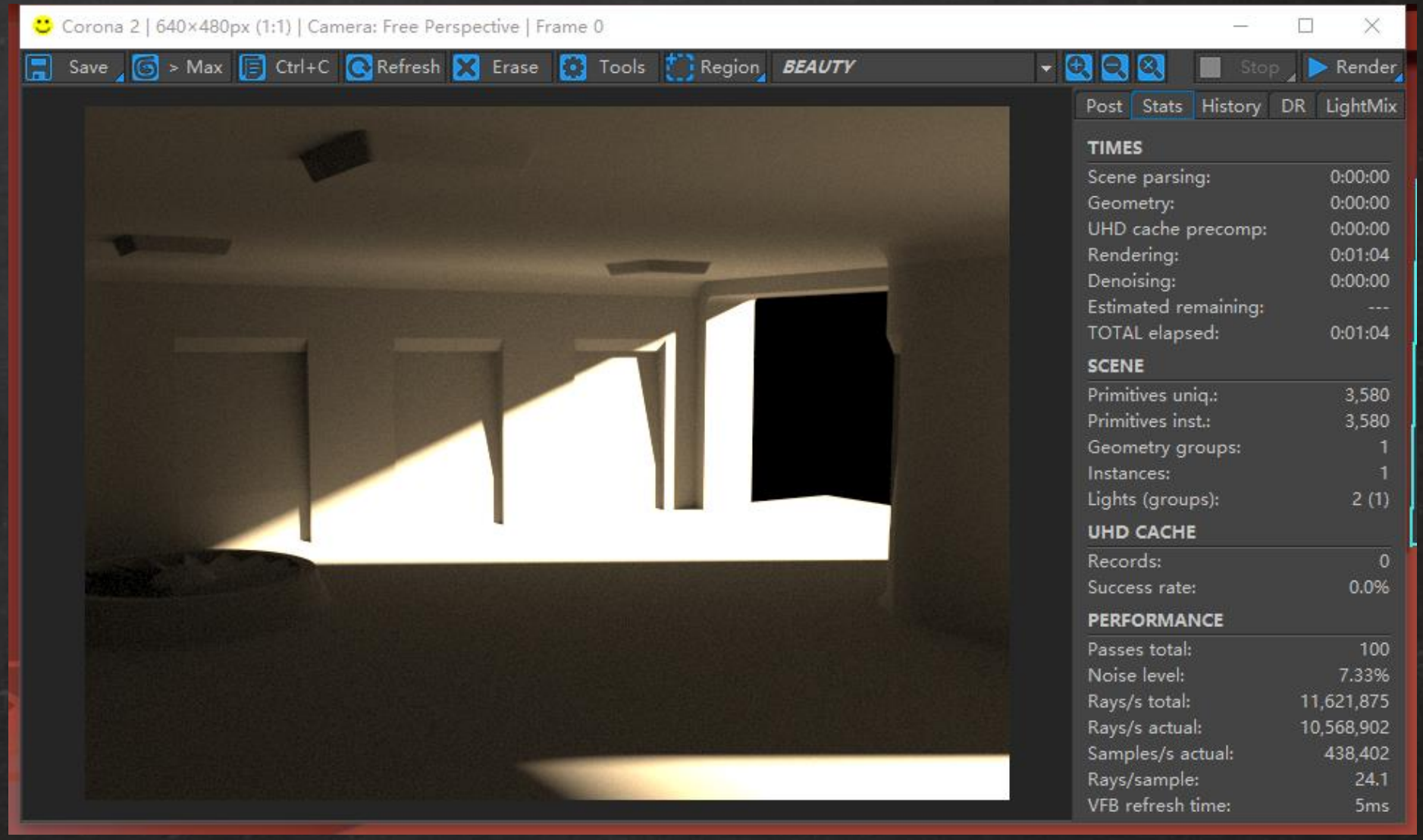


03 烘焙主光：MainLightMask-1

打方向光，开启GI，烘焙到纹理：

- 直接光照部分打爆；
- 反弹光部分照明充分，明暗调子偏线性；
- 明暗交界线半影留半影；

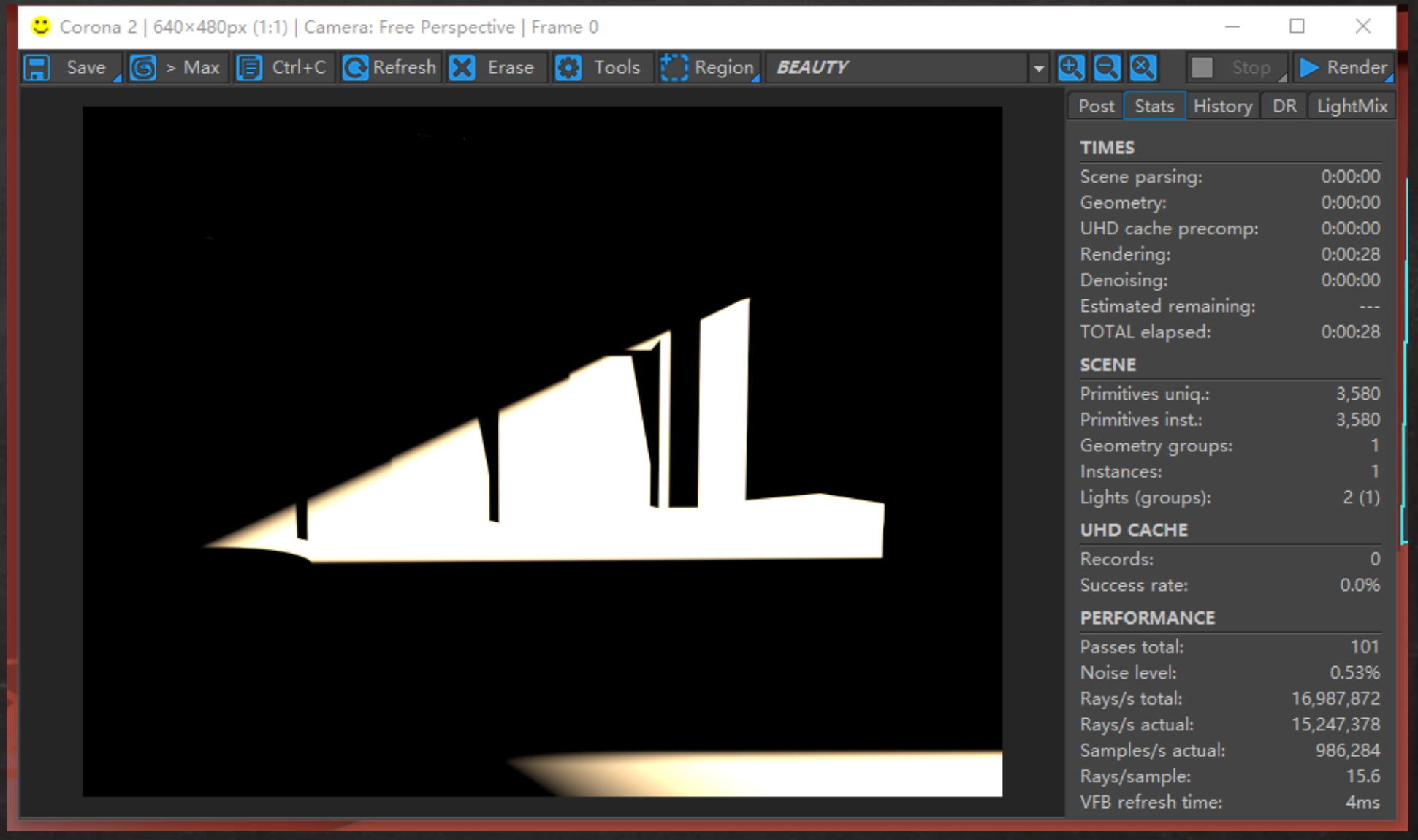




04 烘焙主光：MainLightMask-2

打方向光，关闭GI，烘焙到纹理：

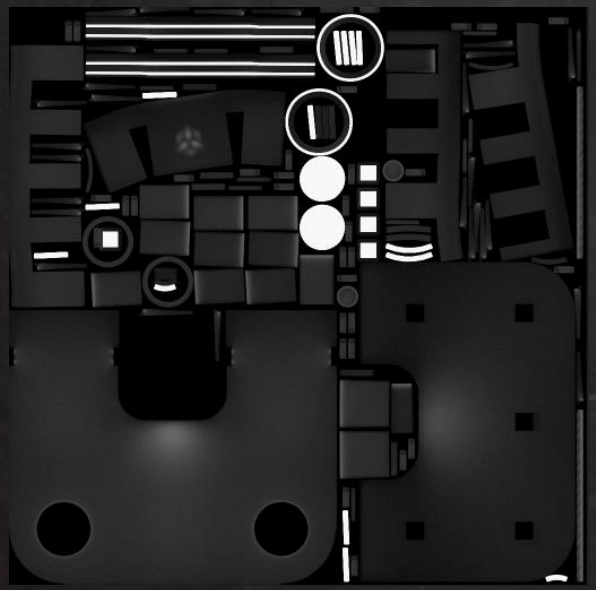
- 与上部分同样光照设置；
- 再次确认明暗交界线半影；

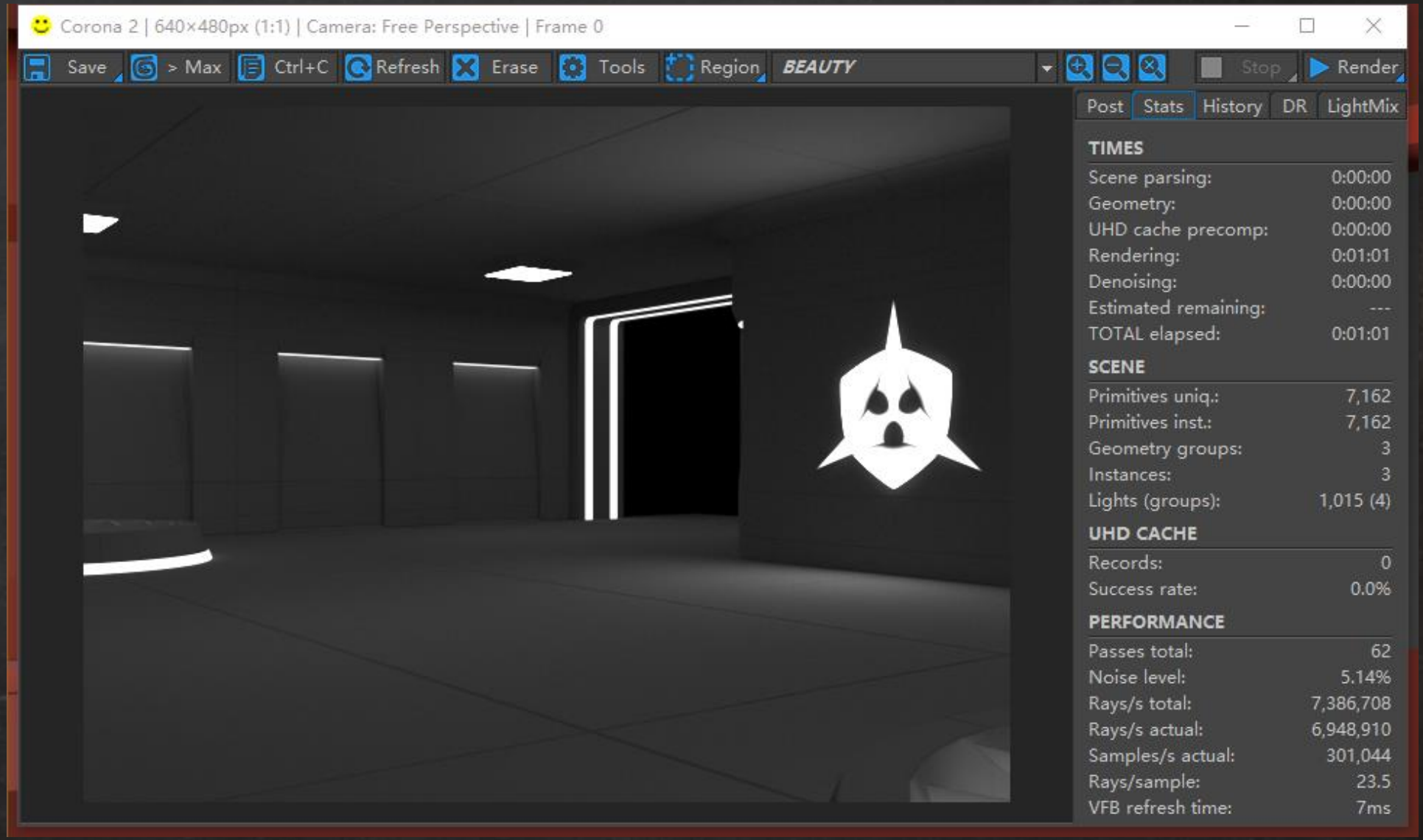


05 烘焙自发光光照：EmitLightMask

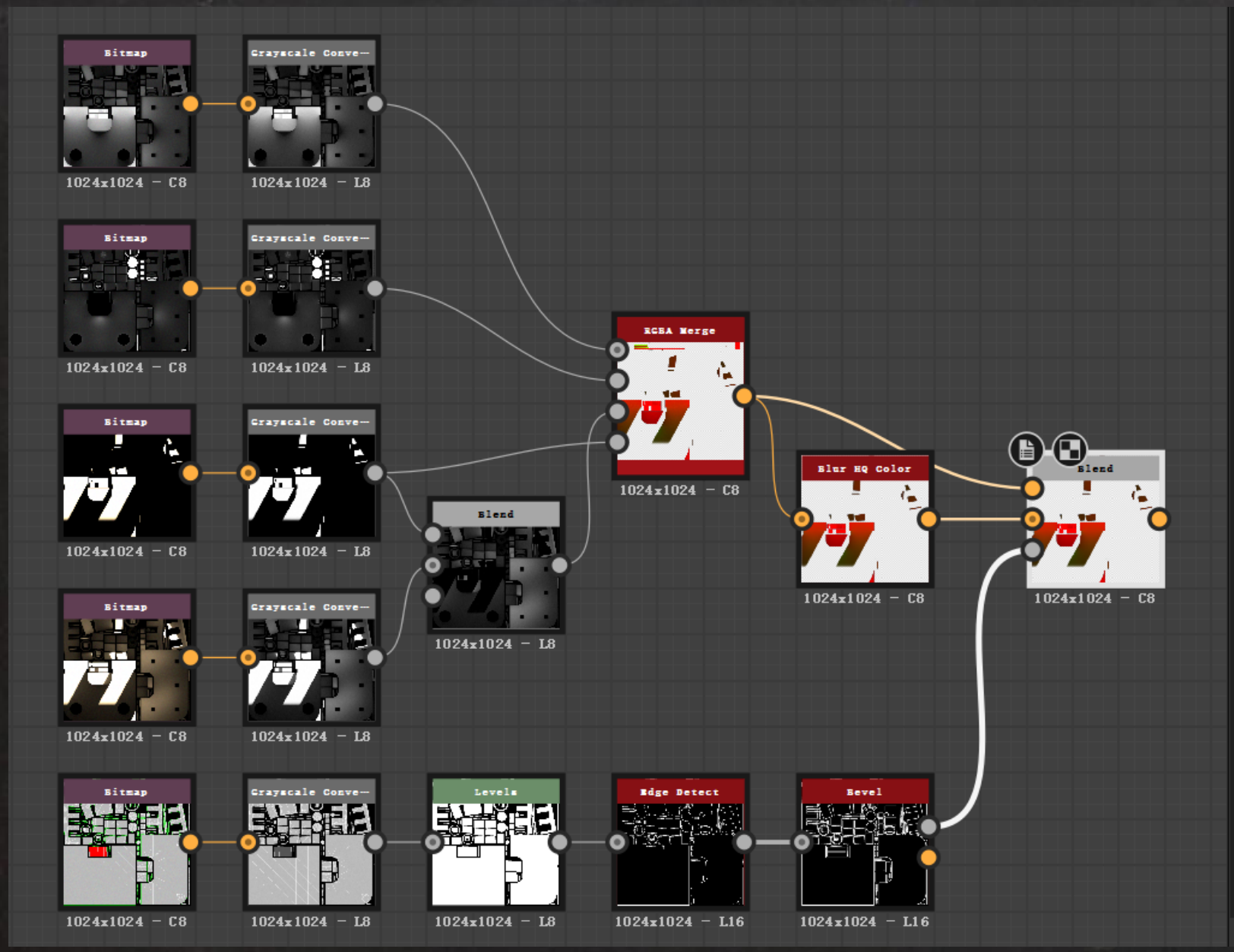
自发光材质，开启GI，烘焙到纹理：

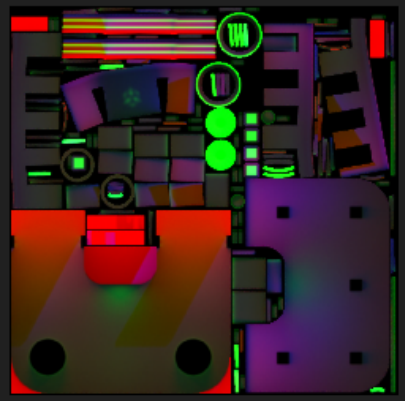
- 白色光即可；
- 调整为想要的照明区域；
- 可按需要调整光照方向性；





06 自定义Lightmap贴图





1024 x 1024 (RGBA, 8bpc)

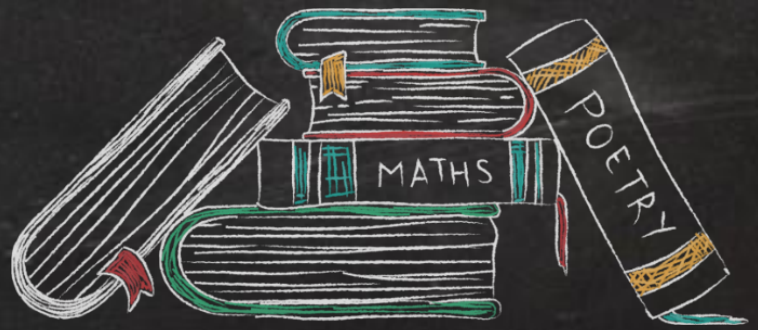
R: SkyLightOcc

G: EmitLightInt

B: MainLightGIInt

A: MainLightShadow

2 符文·化神·SeparatedLM

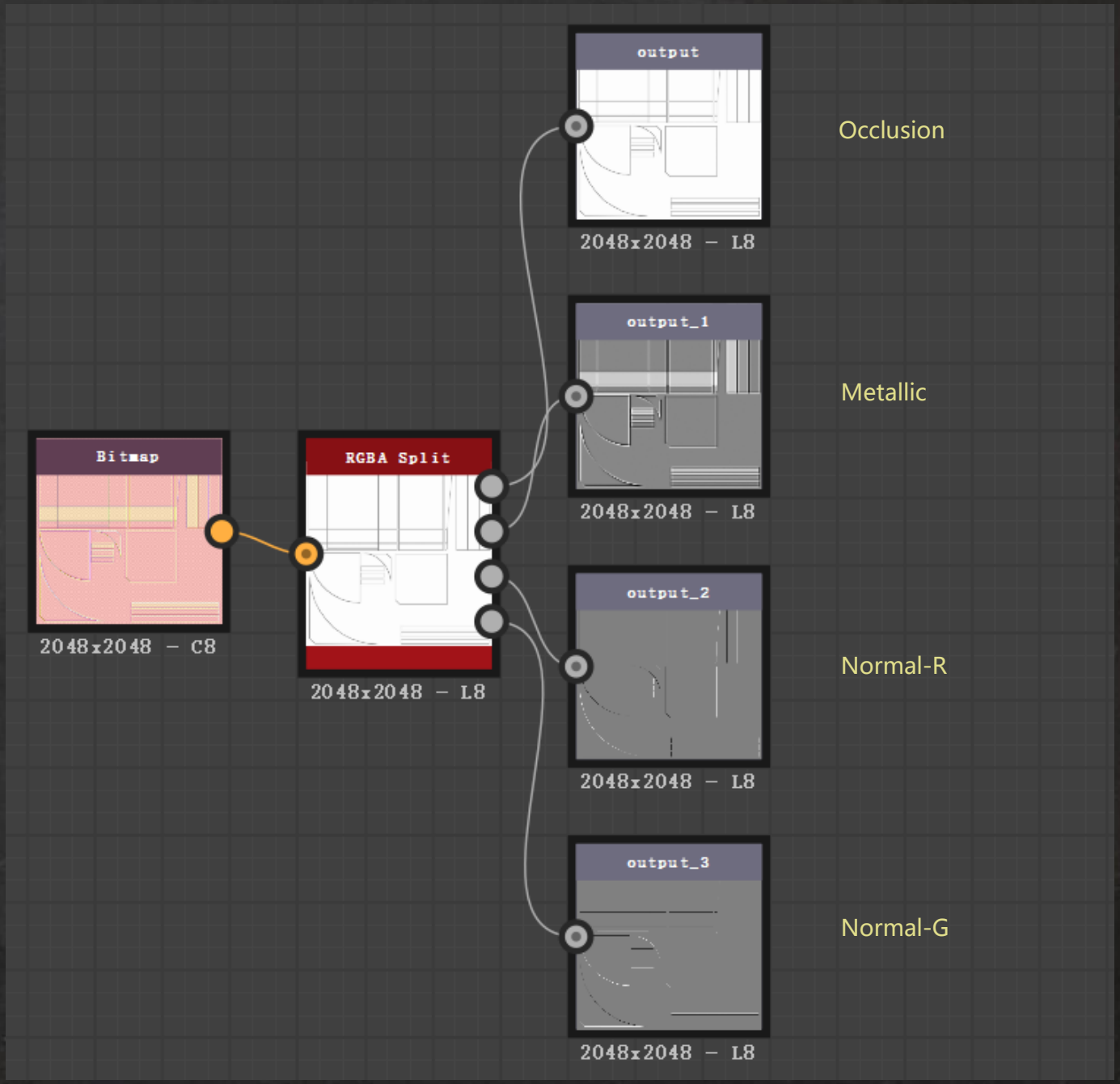


01 SeparatedLM光照构成

02 Mask纹理通道设计

为介绍Shader中的纹理采样，将Mask纹理如右图合并通道：

- 此套免费资源纹理未按PBR规范制作，不要作为纹理绘制参考；
- 法线只取RG两个通道，在Shader解码为完整信息；



03 SeparatedLM代码实现

1. 尝试脱离模板，有需要时从不同模板处CtrlCV；
2. 修改路径名；
3. 规划材质参与材质面板
   - 参数过多不一一说明，绝大多数就是属性名字面意思；
   - _SpecParams：
     - X：非金属高光次幂；
     - Y：非金属高光强度；
     - Z：金属高光次幂；
     - W：金属高光强度；
   - _EnvReflectParams：
     - X：非金属Cube采样Mip值；
     - Y：非金属反射强度；
     - Z：金属Cube采样Mip值；
     - W：金属反射强度；

```
Shader "AP01/L22/Building" {
    Properties {
        [Header(Texture)]
                    _MainTex        ("  颜色纹理", 2D)          = "white" {}
                    _MaskTex        ("  遮罩纹理", 2D)          = "gray" {}
                    _Lightmap       ("  光照纹理", 2D)          = "white" {}
                    _MetalDarken    ("  金属压暗", Float)       = 0.0
        [Header(MainLight)]
        [Toggle]    _MainLightOn    ("  主光开关", Float)       = 0.0
        [HDR]       _MainLightCol   ("  主光颜色", Color)       = (1.0, 1.0, 1.0, 1.0)
        [HDR]       _HalfShadowCol  ("  半影颜色", Color)       = (1.0, 1.0, 1.0, 1.0)
                    _SpecParams     ("  高光参数", Vector)      = (10.0, 1.0, 30.0, 1.0)
        [Header(MainLightGI)]
        [Toggle]    _MainLightGIOn  ("  主光GI开关", Float)     = 0.0
                    _GIInt          ("  GI强度", Float)         = 0.5
        [Header(SkyLight)]
        [Toggle]    _SkyLightOn     ("  天光开关", Float)       = 0.0
                    _SkyCube        ("  天空球", Cube)          = "_Skybox" {}
                    _SkyLightInt    ("  天光强度", Float)       = 1.0
        [Header(EnvReflect)]
        [Toggle]    _EnvReflectOn   ("  环境反射开关", Float)   = 0.0
                    _EnvCube        ("  环境球", Cube)          = "_Skybox" {}
                    _EnvReflectParams ("  反射参数", Vector)    = (7.0, 1.0, 1.0, 0.0)
                    _FresnelPow     ("  菲涅尔次幂", Float)     = 5.0
        [Header(EmitLight)]
        [Toggle]    _EmitLightOn    ("  自发光光照开关", Float) = 0.0
        [HDR]       _EmissionCol    ("  自发光颜色", Color)     = (1.0, 1.0, 1.0, 1.0)
    }
```

04 SeparatedLM代码实现

4. SubShader Tags；
5. Pass Name，Tags；
6. 对应声明输入参数；

```
SubShader {
    Tags {
        "RenderType"="Opaque"
    }
    Pass {
        Name "FORWARD"
        Tags {
            "LightMode"="ForwardBase"
        }
        CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        #include "UnityCG.cginc"
        // Texture
        uniform sampler2D   _MainTex;
        uniform sampler2D   _MaskTex;
        uniform sampler2D   _Lightmap;
        uniform float       _MetalDarken;
        // MainLight
        uniform float       _MainLightOn;
        uniform float3      _MainLightCol;
        uniform float4      _HalfShadowCol;
        uniform float4      _SpecParams;
        // MainLightGI
        uniform float       _MainLightGIOn;
        uniform float       _GIInt;
        // SkyLight
        uniform float       _SkyLightOn;
        uniform samplerCUBE _SkyCube;
        uniform float       _SkyLightInt;
        // EnvReflect
        uniform float       _EnvReflectOn;
        uniform samplerCUBE _EnvCube;
        uniform float4      _EnvReflectParams;
        uniform float       _FresnelPow;
        // EmitLight
        uniform float       _EmitLightOn;
        uniform float3      _EmissionCol;
```

05 SeparatedLM代码实现

7. 输入结构：
   - 用到法线贴图，依赖normal tangent；
   - 用到UV0采样一般纹理，UV1采样Lightmap；
8. 输出结构：
   - 合并UV0，UV1为uvs输出；
   - 计算镜面反射高光，需要输出posWS；
   - 用到法线贴图，需要输出tDirWS，bDirWS；
9. 顶点Shader：
   - 参照往期课程编写；
10. 法线解码方法：
   - 纹理中法线信息值域为(0~1)，映射到(-1~1)；
   - 根据法线信息length为1的特性，求出nDirTS；

```hlsl
// 输入结构
struct VertexInput {
    float4 vertex   : POSITION;   // 顶点信息 Get✔
    float2 uv0      : TEXCOORD0;  // UV信息 Get✔
    float2 uv1      : TEXCOORD1;  // UV信息 Get✔
    float4 normal   : NORMAL;     // 法线信息 Get✔
    float4 tangent  : TANGENT;    // 切线信息 Get✔
};
// 输出结构
struct VertexOutput {
    float4 pos     : SV_POSITION; // 屏幕顶点位置
    float4 uvs      : TEXCOORD0;  // UV0
    float4 posWS    : TEXCOORD1;  // 世界空间顶点位置
    float3 nDirWS   : TEXCOORD2;  // 世界空间法线方向
    float3 tDirWS   : TEXCOORD3;  // 世界空间切线方向
    float3 bDirWS   : TEXCOORD4;  // 世界空间副切线方向
};
// 输入结构>>>顶点Shader>>>输出结构
VertexOutput vert (VertexInput v) {
    VertexOutput o = (VertexOutput)0;                          // 新建输出结构
        o.pos = UnityObjectToClipPos( v.vertex );              // 顶点位置 OS>CS
        o.uvs = float4(v.uv0, v.uv1);                          // 传递UV
        o.posWS = mul(unity_ObjectToWorld, v.vertex);          // 顶点位置 OS>WS
        o.nDirWS = UnityObjectToWorldNormal(v.normal);         // 法线方向 OS>WS
        o.tDirWS = normalize(mul(unity_ObjectToWorld, float4(v.tangent.xyz, 0.0)).xyz); // 切线方向 OS>WS
        o.bDirWS = normalize(cross(o.nDirWS, o.tDirWS) * v.tangent.w);  // 副切线方向
    return o;                                                  // 返回输出结构
}
// 法线信息解码方法
float3 DecodeNormal(float2 maskXY) {
    float2 nDirTSxy = maskXY * 2.0 - 1.0;
    float nDirTSz = sqrt(1.0 - nDirTSxy.x * nDirTSxy.x + nDirTSxy.y * nDirTSxy.y);
    return float3(nDirTSxy, nDirTSz);
}
```

06 SeparatedLM代码实现

11.采样纹理；

12.向量准备；

13.中间量准备；

14.提取表面详细；

15.提取光照信息；

16.采样Cubemap；

```
// 输出结构>>>像素
float4 frag(VertexOutput i) : COLOR {
    // 采样纹理2D
    float3 var_MainTex = tex2D(_MainTex, i.uvs.xy).rgb;
    float4 var_MaskTex = tex2D(_MaskTex, i.uvs.xy);
    float4 var_LightMap = tex2D(_Lightmap, i.uvs.zw);
    // 向量准备
    float3 nDirTS = DecodeNormal(var_MaskTex.zw);
    float3x3 TBN = float3x3(i.tDirWS, i.bDirWS, i.nDirWS);
    float3 nDirWS = normalize(mul(nDirTS, TBN));
    float3 vDirWS = normalize(_WorldSpaceCameraPos.xyz - i.posWS);
    float3 vrDirWS = reflect(-vDirWS, nDirWS);
    float3 lDirWS = _WorldSpaceLightPos0.xyz;
    float3 lrDirWS = reflect(-lDirWS, nDirWS);
    // 中间量准备
    float ndotl = dot(nDirWS, lDirWS);
    float ndotv = dot(nDirWS, vDirWS);
    float vdotr = dot(vDirWS, lrDirWS);
    // 提取表面信息
    float occlusion = var_MaskTex.r;
    float matMask = var_MaskTex.g;
    float3 diffCol = var_MainTex.rgb * lerp(1.0, _MetalDarken, pow(matMask, 5.0));
    float specPow = max(1.0, lerp(_SpecParams.x, _SpecParams.z, matMask));
    float specInt = max(0.0, lerp(_SpecParams.y, _SpecParams.w, matMask));
    float reflectMip = clamp(lerp(_EnvReflectParams.x, _EnvReflectParams.z, matMask), 0.0, 7.0);
    float reflectInt = max(0.0, lerp(_EnvReflectParams.y, _EnvReflectParams.w, matMask));
    float fresnel = lerp(pow(1.0 - max(0.0, ndotv), _FresnelPow), 1.0, matMask);
    // 提取光照信息
    float skyLightOcc = var_LightMap.r;
    float emitLightingInt = var_LightMap.g;
    float mainLightGIInt = var_LightMap.b;
    float mainLightShadow = var_LightMap.a;
    // 采样纹理Cube
    float3 var_SkyCube = texCUBElod(_SkyCube, float4(vrDirWS, 7.0)).rgb;
    float3 var_EnvCube = texCUBElod(_EnvCube, float4(vrDirWS, reflectMip)).rgb;
```

07 最终效果

17.计算主方向光部分

- 漫反射部分；
- 镜面反射部分；
- GI部分；
- 混合；

18.计算复杂光源部分：

- 天光漫反射部分；
- 自发光漫反射部分
- 环境镜面反射部分
- 混合

19.返回值。

```
                // 光照模型
                // MainLight
                    // Diff
                    float3 halfShadowCol = lerp(_HalfShadowCol.rgb, _MainLightCol, mainLightShadow);
                    float3 mainLightCol = lerp(_MainLightCol, halfShadowCol, _HalfShadowCol.a) * mainLightShadow;
                    float3 mainLightDiff = diffCol * mainLightCol * max(0.0, ndotl);
                    // Spec
                    float3 mainLightSpec = mainLightCol * pow(max(0.0, vdotr), specPow) * specInt;
                    // GI
                    float3 mainLightGI = _MainLightCol * occlusion * mainLightGIInt * _GIInt;
                    // 混合
                    float3 mainLight = (mainLightDiff + mainLightSpec + mainLightGI * _MainLightGIOn) * _MainLightOn;
                // OtherLight
                    // Diff
                    float3 skyLightDiff = diffCol * var_SkyCube * _SkyLightInt * skyLightOcc * occlusion;
                    float3 emitLightDiff = diffCol * _EmissionCol * emitLightingInt * occlusion;
                    // Spec
                    float3 envLightSpec = var_EnvCube * reflectInt * fresnel * occlusion;
                    // 混合
                    float3 OtherLight = skyLightDiff * _SkyLightOn + emitLightDiff * _EmitLightOn + envLightSpec * _EnvReflectOn;
                // 返回值
                float3 finalRGB = mainLight + OtherLight;
                return float4(finalRGB, 1.0);
            }
            ENDCG
        }
    }
}
```