

场景Shader

简介

场景使用BlinnPhong光照模型和IBL环境反射，包含视差、潮湿、ScreenDoor半透、自定义阴影颜色等功能。

效果展示



参数说明

1. 叠加色：作用于固有色贴图
2. 固有色贴图：描述物体反射光的参数

3. 剔除模式：1. 关闭 2. 剔除正面 3. 剔除背面
4. 颜色平衡开关【费性能，不需要别开】
 1. 饱和度
 2. 亮度
 3. 对比度
5. 法线贴图：
 1. 法线贴图的纹理设置要改成Default
 2. 不勾选sRGB
6. 开启高度图
 1. 高度图使用法线的b通道
 2. 高度系数：视差强度
7. 法线消隐
 1. 超出视线一定范围的使用顶点法线代替法线贴图，中间有过渡
 2. 法线消隐距离
8. 高光
 1. 高光开关
 2. 高光叠加颜色：高光颜色乘上该颜色
 3. 叠加固有色：高光颜色最后乘上固有色
 4. 光滑度
 5. 高光强度
 6. 高光方向会被EnvironmentShaderProperty脚本（挂在摄像机上）的specLight替换
9. 高光区域调整（已废弃,代删除）
10. 环境反射
 1. 开关
 2. 反射强度（0-4）
 3. 环境反射光泽度增强：针对环境反射加强光泽度，而不影响高光
 4. 环境盒旋转
11. 自发光：
 1. 自发光开关
 2. 流动自发光开关，用一层流动的噪声去扰动自发光。
 3. 流动方向：x: U方向流速 y: V方向流速
 4. 自发光噪声尺寸，值越小，尺寸越大
 5. 自发光遮罩：r: 自发光遮罩 g:自发光流动噪声
12. 高度雾，材质球上的高度雾计算，区别于全屏高度雾
 1. 开关
 2. 高度雾底部颜色
 3. 高度雾顶部颜色
 4. 高度雾起始位置：高度雾底部Y坐标
 5. 高度雾区域厚度：高度雾渐变的高度
 6. 高度雾衰减方式：1. 线性 2. 平方
13. ScreenDoor半透
 1. 开关
 2. 使用透明模板（否则使用8x8矩阵）
 3. 透明模板图（不压缩）
 4. 透明度。调整材质球透明度
14. 潮湿
 1. 开关
 2. 潮湿程度，越潮湿漫反射越弱，镜面反射越强

2. 高光归一化: [The Blinn-Phong Normalization Zoo](#) | [The Tenth Planet](#)

```
half normalizationTerm = (gloss + 2.0) / (2*UNITY_PI); // normalized
blinn-phong
```

3. 镜面反射 IBL(Unity实现): [镜面IBL - LearnOpenGL CN \(learnopengl-cn.github.io\)](#)

```
inline half3 EnvBRDFApprox2(in half3 SpecularColor,in half
perceptualRoughness,in half oneMinusReflectivity,in half NoV)
{
    #ifdef UNITY_COLORSPACE_GAMMA
        half surfaceReduction = 0.28;
    #else
        half surfaceReduction = (0.6-0.08*perceptualRoughness);
    #endif
    surfaceReduction = 1.0 -
perceptualRoughness*perceptualRoughness*perceptualRoughness*surfaceReduc
tion;
    half grazingTerm = saturate((1-perceptualRoughness) + (1-
oneMinusReflectivity));
    return surfaceReduction * FresnelLerp (SpecularColor,
grazingTerm, NoV);
}

// 镜面反射的积分被拆分为两部分，一部分是预滤波环境贴图EnvRef1Col1，一份是模
拟的EnvBRDFApprox[Unity方式]
inline half3 EnvRef1(half3 N,half3 V, half metal, half ao, half
oneMinusReflectivity, half perceptualRoughness,half3 specColor)
{
    half3 dotNV = abs(dot(N,V));
    half3 modelN = UnityWorldToObjectDir(N);
    half3 R = reflect(- V, N);
    half rad = radians(_Ref1Rotate);
    R.xz = Rotate2D(R.xz,rad);

    // half3 env =
EnvBRDFApprox(specColor,perceptualRoughness,dotNV);
    half3 env =
EnvBRDFApprox2(specColor,perceptualRoughness,oneMinusReflectivity,dotNV)
;

    perceptualRoughness = perceptualRoughness*(1.7 -
0.7*perceptualRoughness);
    half mip = perceptualRoughness * UNITY_SPECCUBE_LOD_STEPS;;

    half4 EnvRef1Data =
UNITY_SAMPLE_TEXCUBE_LOD(unity_SpecCube0, R,mip);
    half3 EnvRef1Col1 = DecodeHDR (EnvRef1Data,
unity_SpecCube0_HDR);
    return ao * _Ref1Power * EnvRef1Col1 * env;
}
```

4. 自定义高光方向: 使用另一盏灯控制高光方向, 需要通过EnviromentShaderProperty控制

5. 高光拉伸平移：已废弃

2. 视差计算：[视差贴图 \(Parallax Mapping\) 学习笔记 - 知乎\(zhihu.com\)](#)

```
// 开启凹凸，先取出视差图（高度图）。依据视差图修改UV。  
#if _HEIGHT_ON  
    half Heightmap = tex2D(_NormalTex, i.uv.zw).b;  
    half3 viewDir = mul(i.worldTBN, i.worldview);  
    half2 offset = ParallaxOffset (Heightmap, _Parallax, viewDir);  
    i.uv.xy += offset;  
    i.uv.zw += offset;  
#endif
```

3. 潮湿计算：[Observe rainy world](#)

1. 划分成潮湿、水坑、水坑边缘三部分

1. 潮湿地面，降低漫反射、增强高光。

2. 水坑，调整光泽度、高光系数并使法线固定朝上，使用菲涅尔效应的反射效果，使水坑反射天空景物。

3. 水坑边缘是前两者的过渡。通过控制光泽度、高光系数实现过渡。

```
// 地板水面法线计算，融合了涟漪  
// 先计算涟漪高度，再由高度差得出法线  
half3 waterNormal(float3 pos, half rz){  
    half EPSILON = 0.001 * rz;  
    half3 dx = half3( EPSILON, 0., 0. );  
    half3 dz = half3( 0., 0., EPSILON );  
  
    half3 normal = half3( 0., 1., 0. );  
    half bumpfactor = 0.2 * (1. - smoothstep( 0., 1000, rz )); //根据  
    // 距离所见Bump幅度  
    half2 nor_ofs = WaterDistort(pos);  
    // 计算扰动噪声  
    // 扰动和涟漪结合  
    normal.x = -bumpfactor * (WaterMap(pos + dx) - WaterMap(pos-dx))  
    / (2. * EPSILON) + nor_ofs.x;  
    normal.z = -bumpfactor * (WaterMap(pos + dz) - WaterMap(pos-dz))  
    / (2. * EPSILON) + nor_ofs.y;  
    return normalize( normal );  
}  
  
// 水面法线计算可能有点费(法线是先计算涟漪高度，再由高度计算高度差得到的)，可以考虑换成其他方案  
// 水坑计算被废弃掉了，美术不用  
inline void DOWetSetup(inout half3 diffuse, inout half gloss, inout  
half specular, in float2 uv, in float3 worldPos, inout half3  
worldNormal, half height){  
    // 水坑计算  
    fixed waterDepth = 0; //saturate(1-tex2D(_WaterMask, uv).a);  
  
    // 如果高度图和缝隙不太搭，在这里调参试试  
    half ScaleHeight = 1.0f;  
    half BiasHeight = 0.0f;  
    height = height * ScaleHeight + BiasHeight;
```

```

// 接近水的程度
half2 Accumulatedwaters;
// 缝隙间的接近水的程度
Accumulatedwaters.x = min(_FloodLevel1, 1.0 - height) ;
// 水坑里接近水的程度
Accumulatedwaters.y = saturate(_FloodLevel2 - waterDepth);
// 取当前像素点最接近水的系数
half Accumulatedwater = max(Accumulatedwaters.x,
Accumulatedwaters.y);
// 潮湿程度 = 潮湿系数+当前位置接近水的程度
// half NewWetLevel = saturate(_WetLevel + Accumulatedwater);

// 设置潮湿地面的潮湿程度。越潮湿，漫反射越弱，高光的光泽度越强
diffuse.rgb *= lerp(1.0, 0.3, _WetLevel);
gloss = min(gloss * lerp(1.0, 2.5, _WetLevel), 1.0);
// // 设置水坑或缝隙间水的光泽度和反射系数
// gloss = lerp(gloss, 1.0, Accumulatedwater);
// specular = lerp(specular, 0.02, Accumulatedwater);

// 涟漪法线
half3 RippleNormal = WaterNormal(worldPos, 1);
// 给涟漪添加自发光
diffuse.rgb += saturate(WaterMap(worldPos)) * _RainIntensity *
0.5/pow(_Tile,1.5);
// 水面和涟漪混合
half3 waterNormal = lerp(half3(0, 1,
0),RippleNormal,saturate(_RainIntensity) );
// 计算世界坐标系下的normal、lightDir、viewDir、H
half3 N = lerp(worldNormal,waterNormal,Accumulatedwater);
// 修改后续光照的法线
worldNormal = N;
}

```

2. 涟漪，通过数学计算或动态ripple纹理，计算涟漪高度，应用到法线上。[中级Shader教程25 两种涟漪实现方式/jepengTan's blog专栏-CSDN博客unity_涟漪](#)



```

// 计算涟漪
// 核心公式  $y = \sin(31.*t) * \text{smoothstep}(-0.6, -0.3, t) * \text{smoothstep}(0., -0.3, t)$ 
half _Ripple(half period, half spreadSpd, half waveGap, float2 uv, half
rnd){

    const fixed CROSS_NUM = 1.0;
    half radius = (half(CROSS_NUM));
    half2 p0 = floor(uv);
    half sum = 0.;

    //多个格子中的波动全部累计起来,避免涟漪被分割
    for (half j = -CROSS_NUM; j <= CROSS_NUM; ++j){
        for (half i = -CROSS_NUM; i <= CROSS_NUM; ++i){
            half2 pi = p0 + half2(i, j);
            half2 h22 = Hash23(half3(pi, rnd));
            half h12 = Hash13(half3(pi, rnd));
            // 波峰个数设为2~3个
            half WAVE_NUM = 2. + round(h12);
            half ww = -WAVE_NUM * .5 * waveGap;
            half hww = ww * 0.5;
            half freq = WAVE_NUM * PI2 / waveGap / (CROSS_NUM + 1.);
            half pd = period * (h12 * 1. + 1.); //让周期随机
            float time = _Time.y; //+pd*h12; //让时间偏移点 不会全部同时
出现

            float t = fmod(time, pd);
            half spd = spreadSpd * ((1.0 - h12) * 0.2 + 0.8); //让传播速度
随机

            half size = (h12) * 0.4 + 0.6;
            half maxt = min(pd * 0.6, radius * size / spd);
            half amp = clamp01(1. - t / maxt);
            float2 p = pi + Hash21(h12 + floor(time / pd)) * 0.4;
            half d = (length(p - uv) - spd * t) / radius * 0.5;
            sum -= amp * sin(freq * d) * smoothstep(ww * size, hww * size,
d) * smoothstep(0., hww * size, d); //让波动传播开来
        }
    }
    sum /= (CROSS_NUM * 2 + 1) * (CROSS_NUM * 2 + 1);
    return sum;
}

```

4. ScreenDoor半透 [Screen-Door Transparency \(digitalrune.github.io\)](https://digitalrune.github.io/)

```

// 这是简化版逻辑,按像素坐标从矩阵中取alpha阈值。裁掉小于阈值的。
// 实际使用时,使用了更大的矩阵,且不是矩阵格式,计算略复杂。
float4x4 thresholdMatrix =
{
    1.0 / 17.0, 9.0 / 17.0, 3.0 / 17.0, 11.0 / 17.0,
    13.0 / 17.0, 5.0 / 17.0, 15.0 / 17.0, 7.0 / 17.0,
    4.0 / 17.0, 12.0 / 17.0, 2.0 / 17.0, 10.0 / 17.0,
    16.0 / 17.0, 8.0 / 17.0, 14.0 / 17.0, 6.0 / 17.0
};

clip(alpha - thresholdMatrix[pos.x % 4][pos.y % 4]);

```

绘制流程

```
half4 frag_main (v2f_main i) : SV_Target
{
    // 开启凹凸，先取出视差图（高度图）。依据视差图修改uv。
    // 取出固有色
    // 高光开启，设置光泽度和高光反射系数
    // 设置高光反射颜色(依据金属度设置)
    // 归一化各个方向向量
    // 潮湿地面开启，调整漫反射、高光系数，并修改表面法线。
    // 打雷
    // screen_door
    // 光源衰减
    // lightmap
    // 依据光照类型计算光照 1. FULLY_BAKED_LIGHTING 2. SUBTRACTIVE_LIGHTING 3.
    SHADOWMASK_LIGHTING 4. BAKE_INDIRECT_LIGHTING 5. REALTIME_LIGHTING
    // 自发光
    // 阴影颜色
    // 叠加雾效
    // 色彩平衡
}
```