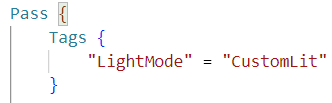
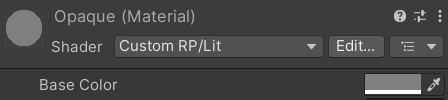
03 直接光照

首先，需要将原先的Unlit着色器改成接收光源的Lit着色器。

复制一份Unlit着色器，为Pass增加Tag属性：

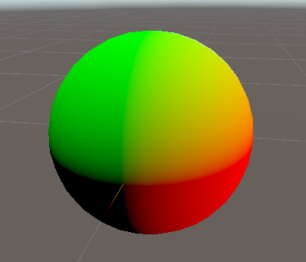


然后在管线中，绘制时为DrawSettings添加对应的ShaderTagId，就可以正常显示该着色器的绘制结果了。



然后把顶点法线变换到世界坐标系下，再传给片元着色器，可以将法线作为颜色输出检查。

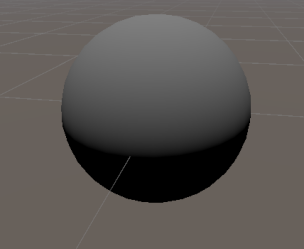
注：当模型矩阵（M）包含非均一变换时，法线的变换矩阵为M的逆的转置（推导思路：切向量的变换矩阵一直是M）。



因为在三角面上对法向量进行线性插值会导致法向量长度不完全等于1，所以在片元着色器中，还需要对法向量进行额外的均一化操作（normalization）。

接下来搭建框架，把法线和baseColor（Albedo）封装成Surface结构体，表示表面的属性，再构造一个函数GetLighting，输入表面属性，输出照明结果。

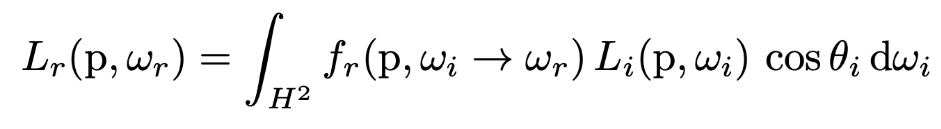
先用baseColor和法线的y分量相乘的结果作为照明结果输出：



下面处理光源，目前只考虑方向光。

同样，使用结构体Light封装光源属性，通过GetDirectionalLight返回光源信息。

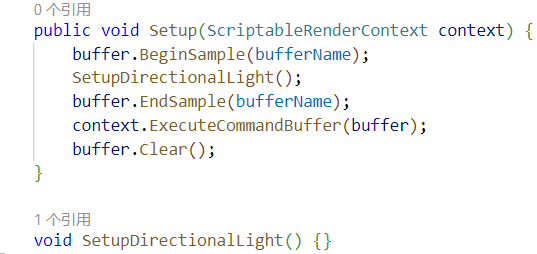
先贴一下渲染方程：



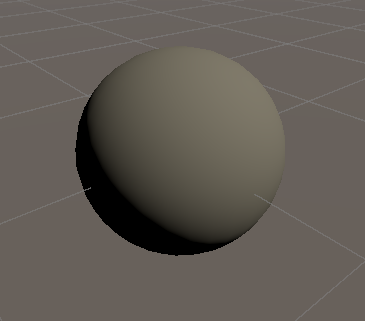
那么首先，用函数IncomingLight表示L乘cos项的结果。注意cos项为负时应该记为0。

为了简化，先直接用IncomingLight和baseColor点乘的结果作为光照结果。

接下来，尝试上传光源数据到GPU。



基本结构如上，和CameraRenderer类似，每帧向context里写入上传指令即可。此外，通过RenderSetting.sun获取太阳光源的信息。



如果要支持多光源，可以从cullingResult.visibleLights里获取，然后作为数组上传，同时限制数组的最大数量，并且要把float3改成float4。

光照影响通过求和模拟积分。

