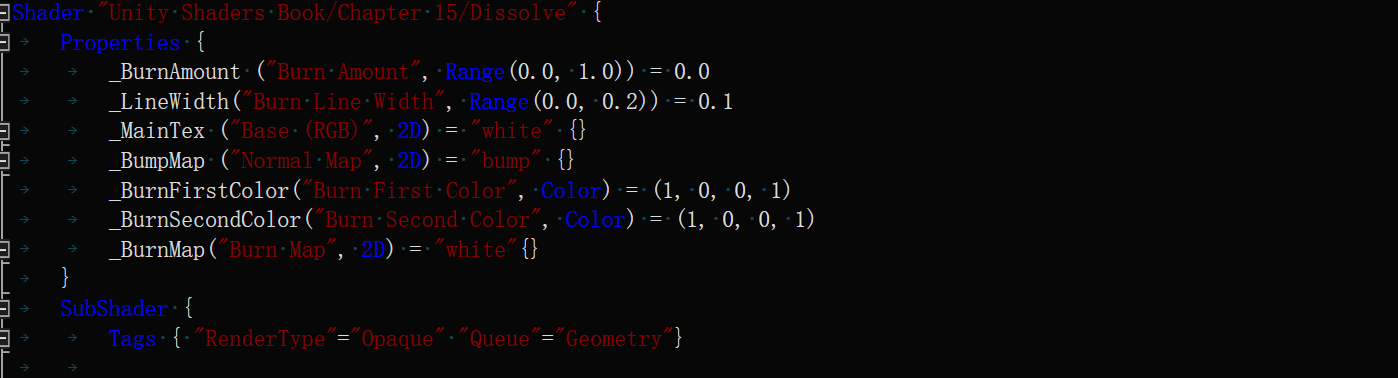
GPU很难生成随机数，为了能够产生随机数，可以使用一张随机数纹理来作为随机数的查找表。这样产生一个伪随机的效果。在随机数的加持下。

消融效果：

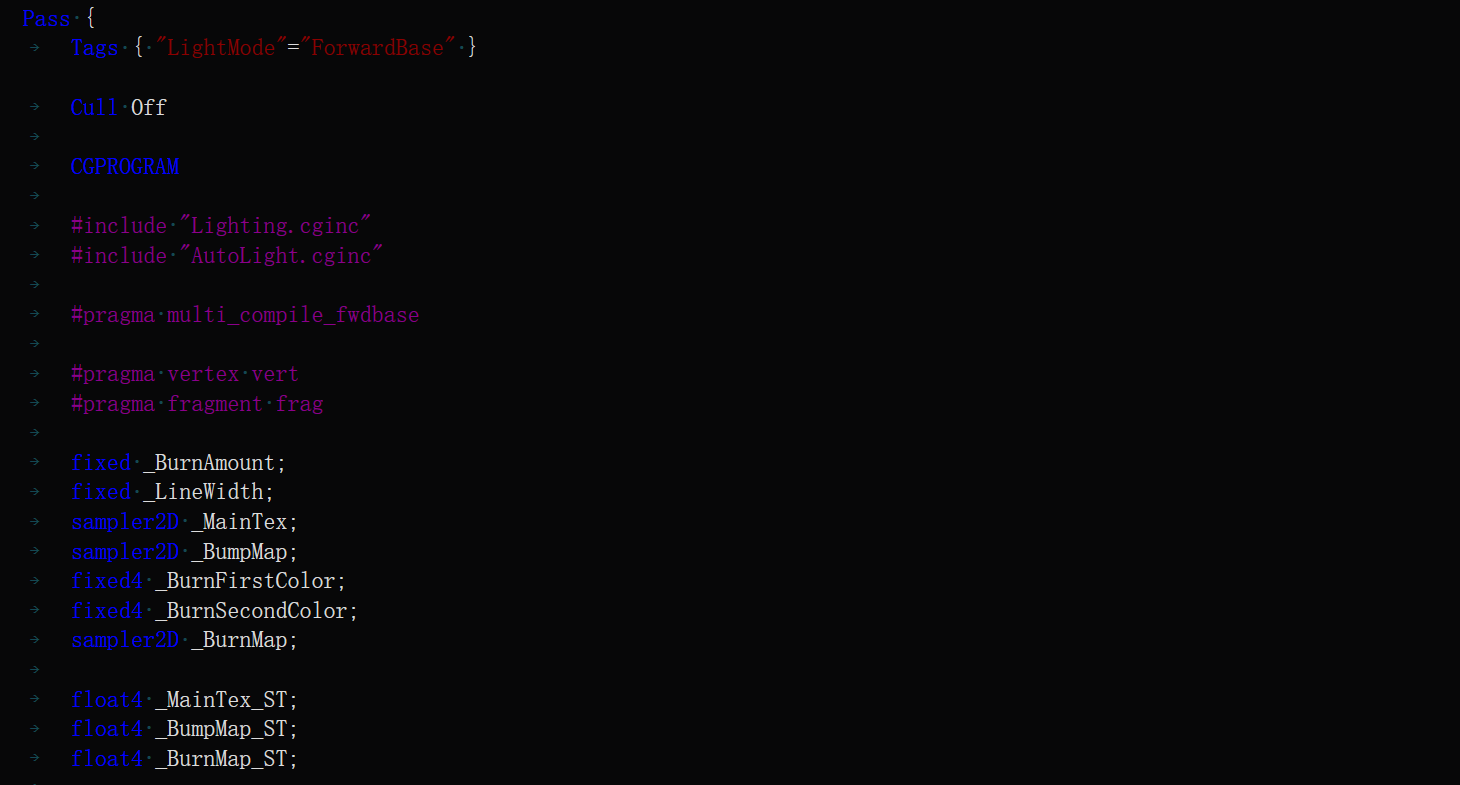
消融效果（dissolve），经常用于一些游戏中的怪物死亡，或者建筑烧毁然后消失的情景。消融从随机的地方开始，然后向四周蔓延，直到所有的地方都消失不见，然后被系统删除。

原理：噪声纹理+透明度测试。我们用对噪声纹理采样的结果和一个指定的阈值作对比，如果小于阈值，就把这个像素剔除掉，这些部分就对应了被烧毁的区域。镂空区域边缘的烧焦效果则是将两种颜色混合，再用pow函数处理后，与原纹理颜色混合后的结果。

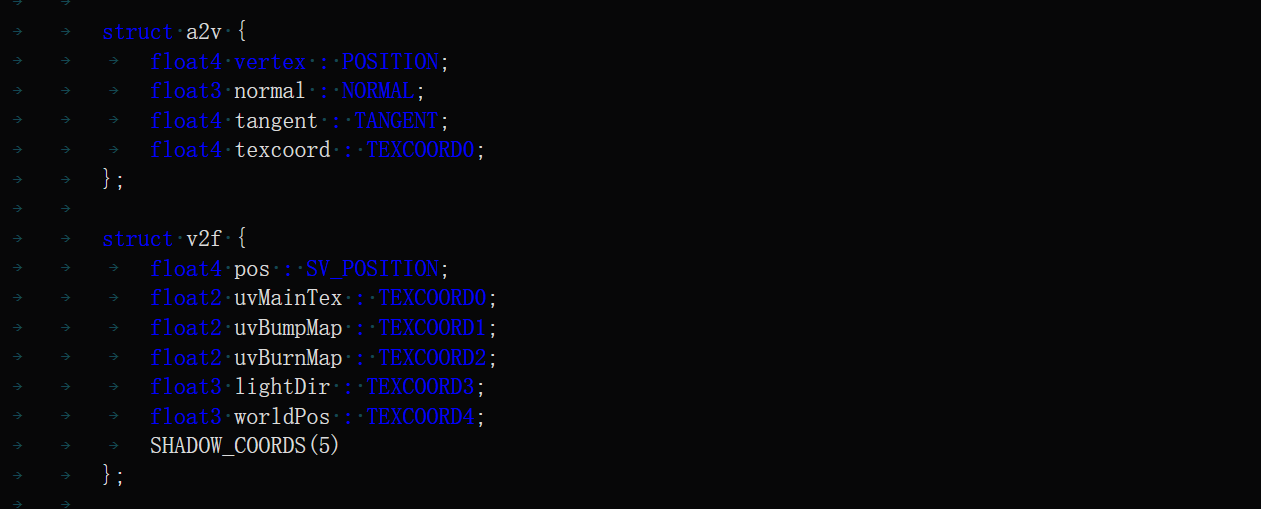


\_BurnAmount控制消融程度，0为正常1为完全消融。\_LineWidth空中模拟烧焦效果的线宽，它的值越大，火焰蔓延范围越广，\_BurnFirstColor和\_BurnSecondColor对应火焰边缘的两种颜色，\_BurnMap是噪声纹理

第一个Pass定义消融

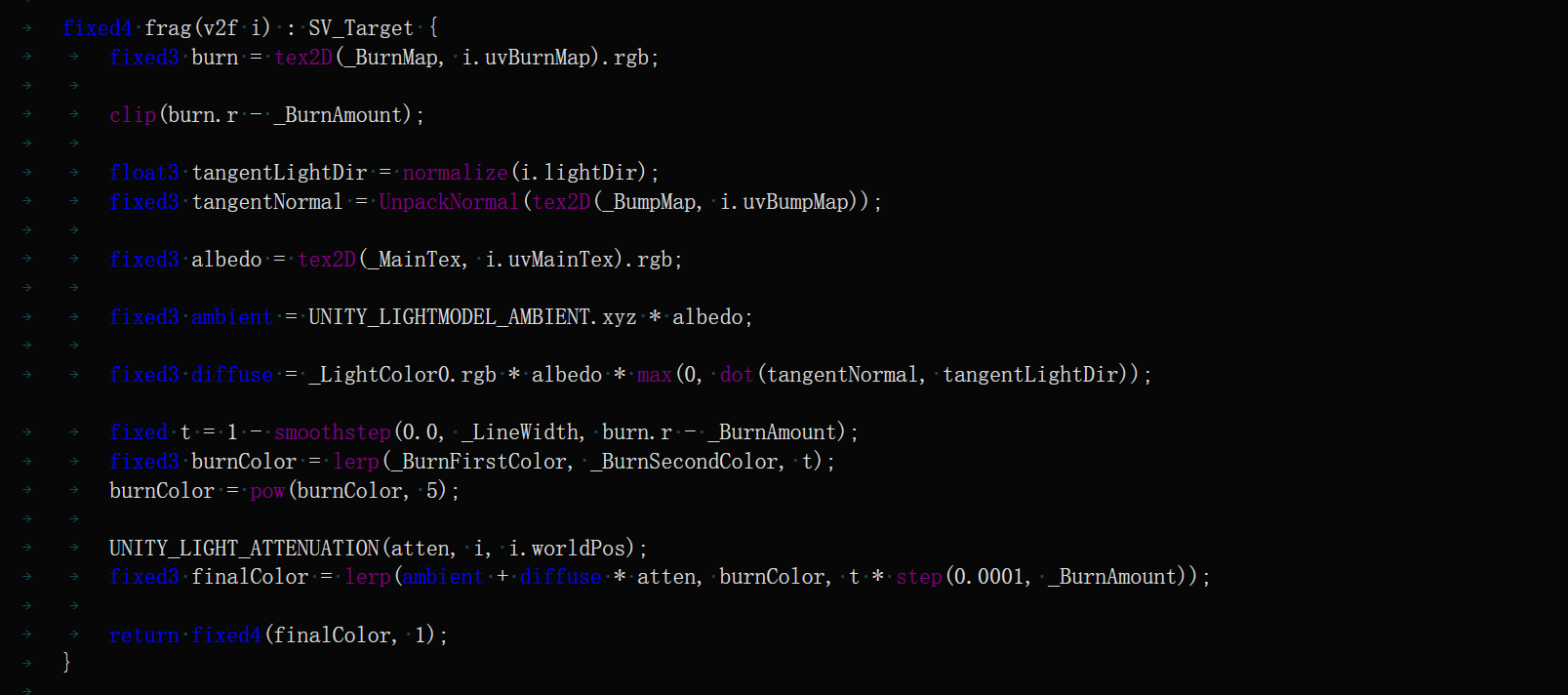


关闭面片剔除，因为烧毁之后可以看到模型的内部，因此需要渲染两面



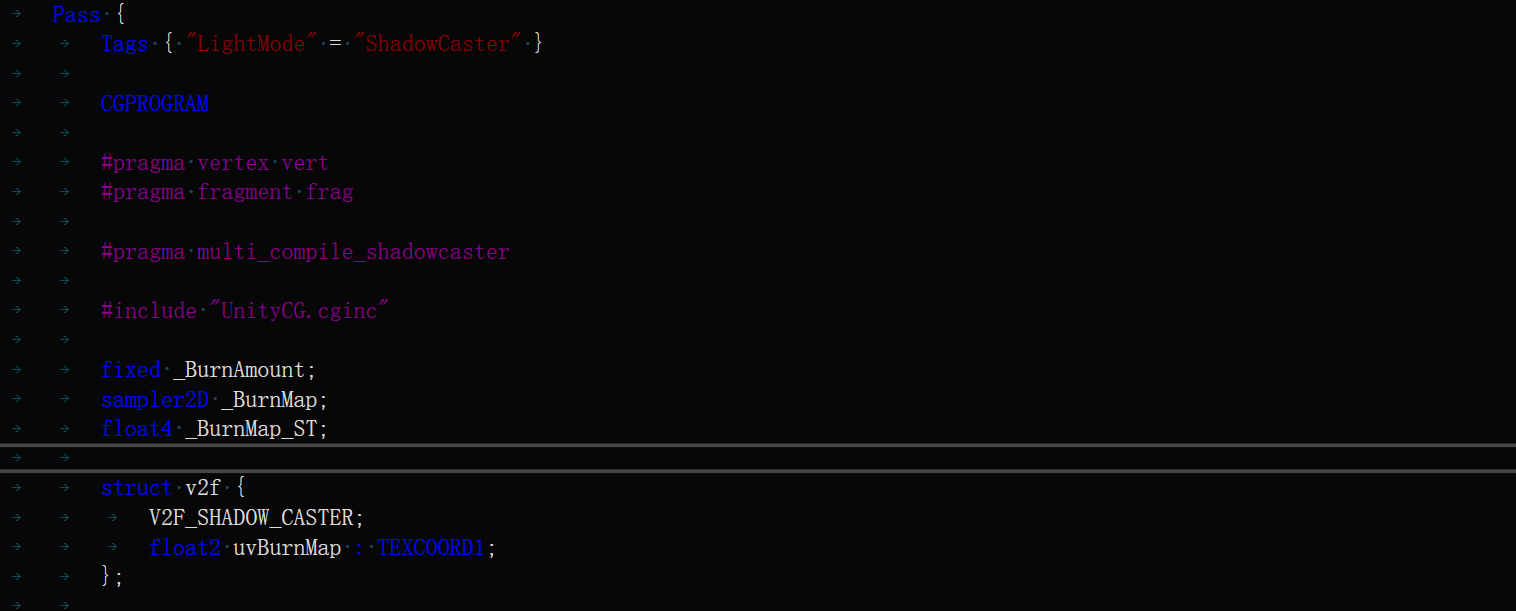
文本

描述已自动生成

顶点着色器中计算纹理坐标，然后把光源方向变换到切线空间，然后计算了阴影纹理的采样坐标

片元着色器中完成消融效果。对噪声纹理采样，在clip函数中与控制消融程度的属性相减判断该像素是否需要剔除，通过测试进行正常的光照计算。计算漫反射光照和环境光照后。之后计算烧焦颜色burnColor，在宽度为\_LineWidth的范围内模拟一个烧焦变化，第一步使用smoothstep函数计算混合系数t，t为1时像素处于消融的边界处，为0时表面像素为正常的模型颜色，中间的差值表示模拟烧焦的效果。使用两种火焰颜色和pow函数让效果更接近烧焦的痕迹（增强burnColor的对比度），再使用t混合正常光照颜色和烧焦颜色，最后返回混合后的颜色值。

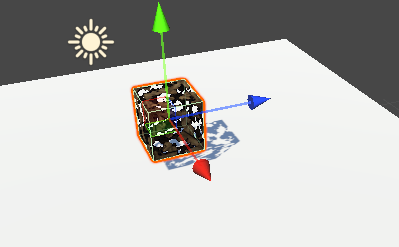
第二个pass完成投射阴影，不能使用普通的阴影pass因为这样被剔除的区域仍会向其他物体投射阴影



文本

描述已自动生成

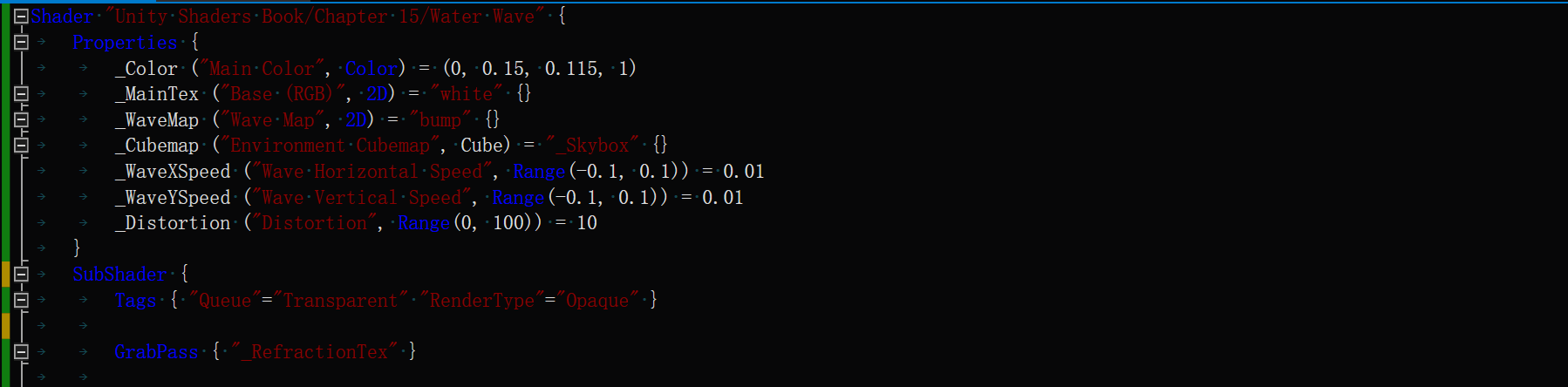
片元着色器中剔除了被烧毁部分的阴影



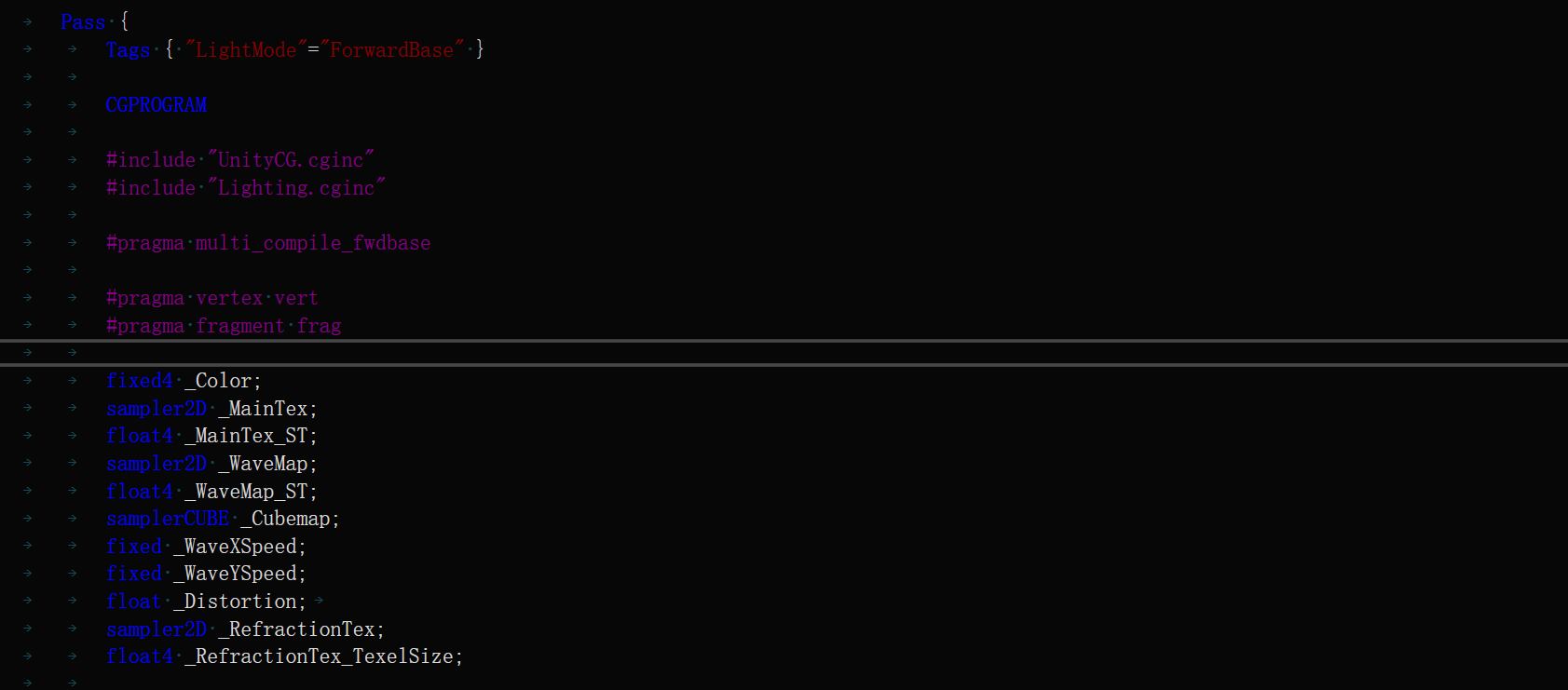
水波效果：

Ps：这里的水波效果只是视觉效果上的水波，准确来说，是有光照，但是没有顶点变化。（也就是在一张不会动的纸上，用纯粹的光照计算来表达出水波）当我们从水平面观察这个水面的时候就露馅了。

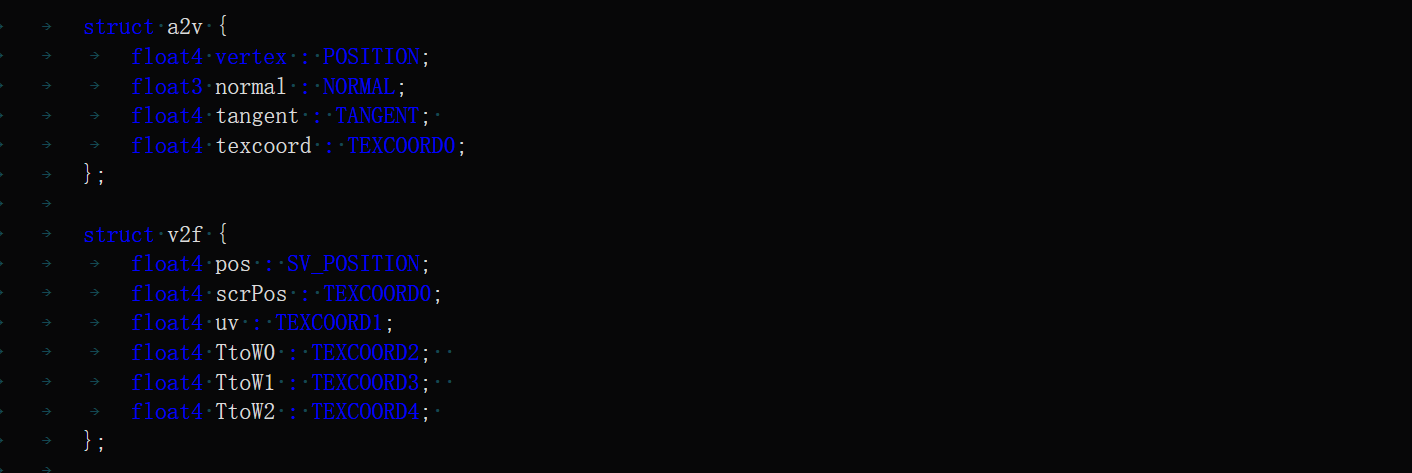
这里会使用噪声纹理来充当一个高度图，用它来表示水面的法线。我们会使用到时间变量和这个高度图，两者混合在一起。当法线随时间进行变化。得到法线之后，再根据法线来进行正常的光照计算。



\_WaveMap是由纹理噪声生成的法线纹理，\_Distortion用于控制模拟折射时图像的扭曲程度，\_WaveXSpeed和\_WaveYSpeed控制法线再X和Y轴上的平移速度。使用GrabPass获取屏幕图像

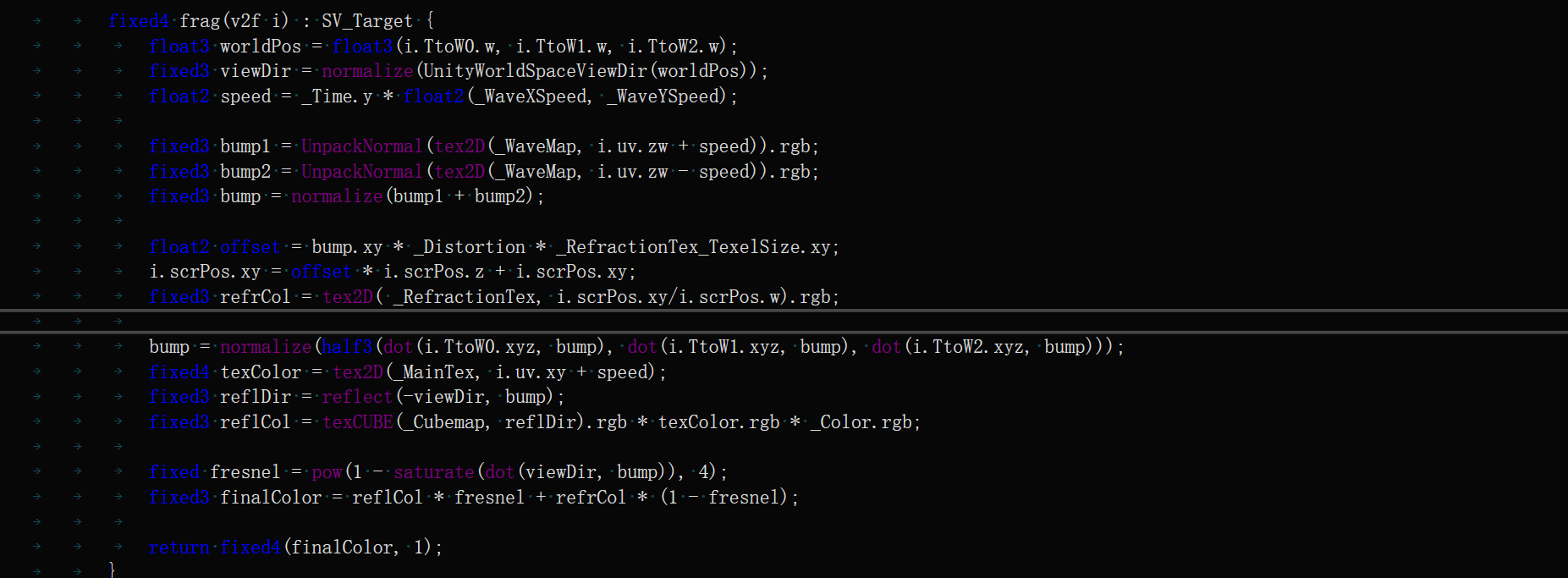


\_RefractionTex和\_RefractionTex\_TexelSize对应使用GrabPass时指定的纹理名称和纹素大小



文本

描述已自动生成



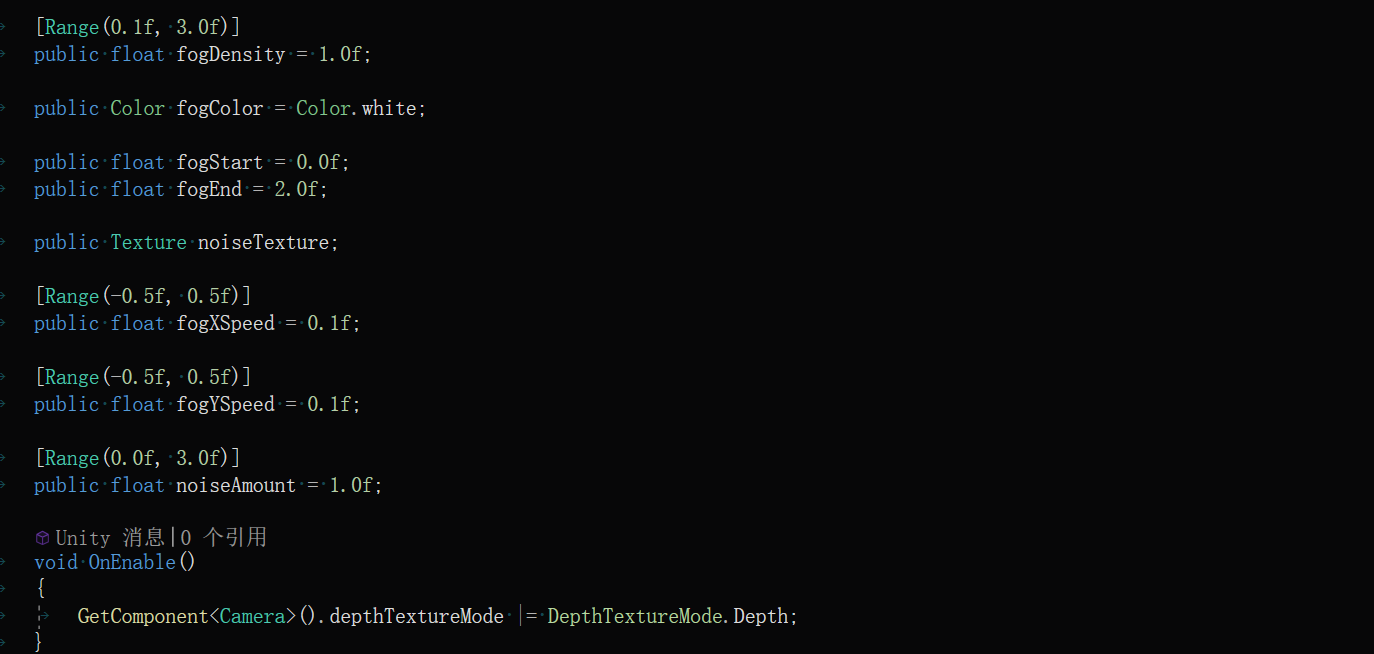
fixed3 bump1 = UnpackNormal(tex2D(\_WaveMap, i.uv.zw + speed)).rgb;：这行代码从波形图（\_WaveMap）中获取一个法线贴图，贴图的UV坐标被加上了一个速度值（speed），这样可以模拟波动的效果。UnpackNormal函数将压缩的法线贴图数据解压缩为实际的法线向量。

fixed3 bump2 = UnpackNormal(tex2D(\_WaveMap, i.uv.zw - speed)).rgb;：这行代码的作用与上一行类似，但是UV坐标被减去了速度值，这样可以模拟波动的反方向。

fixed3 bump = normalize(bump1 + bump2);：这行代码将两个法线向量相加并归一化，得到最终的法线向量。这个向量将被用于计算反射和折射的颜色。

全局雾效：

脚本部分：



文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

Shader部分：

文本

描述已自动生成

图片包含 图形用户界面

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成