什么情况下需要使用深度和法线纹理？

有时候进行屏幕后处理效果时如果仅仅凭借颜色信息会出现偏差。比如在进行边缘检测时，直接利用颜色信息会使检测到的边缘信息受物体纹理和光照等外部因素影响，并且会得到很多不需要的边缘点，而此时的解决方法就是在深度纹理和法线纹理上进行边缘检测，这样图像就不会受纹理和光照影响，得到的边缘会更加可靠。

获取深度和法线纹理

1. 原理

深度纹理记录的是深度值而不是像素值，深度值在0~1之间且是非线性分布的。深度值来自于变换后得到的归一化的设备坐标。如果一个模型要想显示在屏幕上，要把它的顶点从模型空间变到齐次空间下，当我们使用透视摄像机时这个投影就是非线性的。

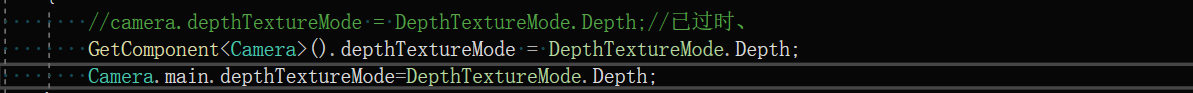
深度纹理可以直接来自于真正的深度缓存，也可以由一个单独的Pass渲染得到。unity会把标签RenderType为Opaque的物体且它使用队列小于2500的，就把它的深度和法线渲染出来，存在深度纹理和法线纹理中。我们可以让一个摄像机生成一张深度纹理或是一张深度+法线纹理。

着色器替换技术，选取需要的不透明的物体，并使用它投射阴影时使用的Pass，即LightMode为ShadowCaster的Pass得来到深度纹理。如果Shader中不包含这种Pass，则不会有深度纹理。

深度纹理通常有24或16位，如果深度+法线纹理则为32位，法线信息会写进R和G通道，深度则是B和A通道。

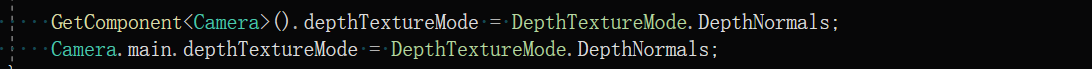
1. 如何获取深度纹理

通过脚本设置摄像机的DepthTextureMode来获取深度纹理

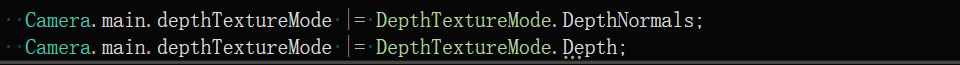


设置摄像机模式后就可以在Shader中声明\_CameraDepthTexture变量来访问它

也可以这样设置来获取深度纹理+法线纹理

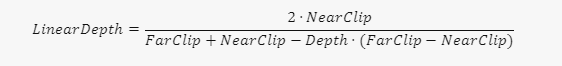


可以使用位标志组合的方式设置DepthTextureMode来获取深度纹理+法线纹理



当通过纹理采样获得深度值之后，这些获得的深度值通常是非线性的，这些非线性来源于透视投影使用的裁剪矩阵（正交投影没有这个问题），但是计算中需要的是线性的深度值，因为线性的深度值在可以直接反映出物体在场景中的实际距离，在计算物体间深度差异时更加直观。

非线性深度值转换为线性深度值：



具体转换过程：

图片包含 图示

描述已自动生成

这是透视投影的，世界空间转换裁剪空间的矩阵：

其中，f是远裁剪面的距离，n是近裁剪面的距离，r是宽高比。当我们用这个矩阵去变换一个齐次坐标(x, y, z, 1)，我们得到的新的齐次坐标是(x', y', z', w')。深度值就是z' / w'。

将z' / w'代入并化简得到：

图片包含 图示

描述已自动生成

这个Depth就是深度值（在近裁剪面和远裁剪面之间[-1,1]）和真实的Z坐标（在世界空间中）之间的关系。显而易见这个深度值是非线性的。为了将其转换为线性深度值需要进行一些数学变换。

首先需要将采样得到的[-1,1]的深度值映射回到[0,1]的范围内，得到深度纹理中的深度值：



把这个[-1，1]的深度值公式除以远平面深度就可以得到线性的范围在[0,1]的深度值。

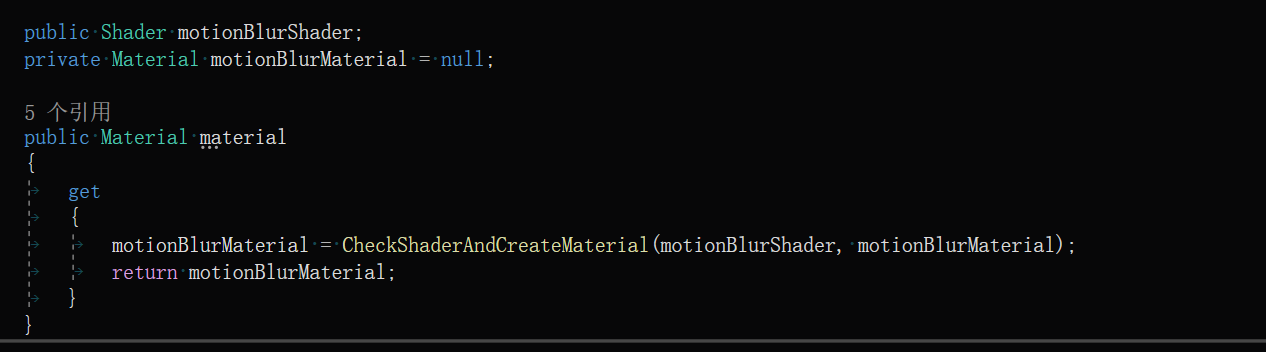
Unity提供了两个函数辅助计算上述过程，

LinearEyeDepth：负责把深度纹理的采样转到视角空间下的深度值。

Linear01Depth：返回一个0~1的线性深度值 。

运动模糊：

脚本部分：



声明需要的Shader并声明和创建相应材质



运动模糊时模糊图像的程度

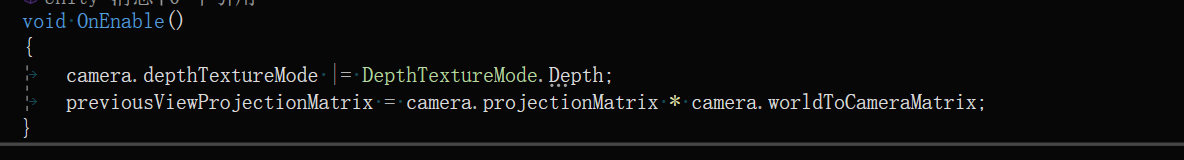
文本

描述已自动生成

获取脚本所在的摄像机组件，得到深度值时需要，可以通过LinearEyeDepth得到摄像机空间（观察空间）下的深度值



存储上一帧的视图投影矩阵

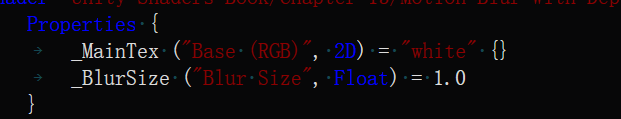


脚本启用时开启摄像机的深度纹理模式，计算并存储上一帧的视图投影矩阵



实现OnRenderImage函数，把模糊程度传给Shader的属性，设置材质的上一帧视图投影矩阵，再计算当前帧的视图投影矩阵并求出其逆矩阵，把这个逆矩阵传给Shader，然后把当前帧的视图投影矩阵设置为前一帧的视图投影矩阵

Shader部分：



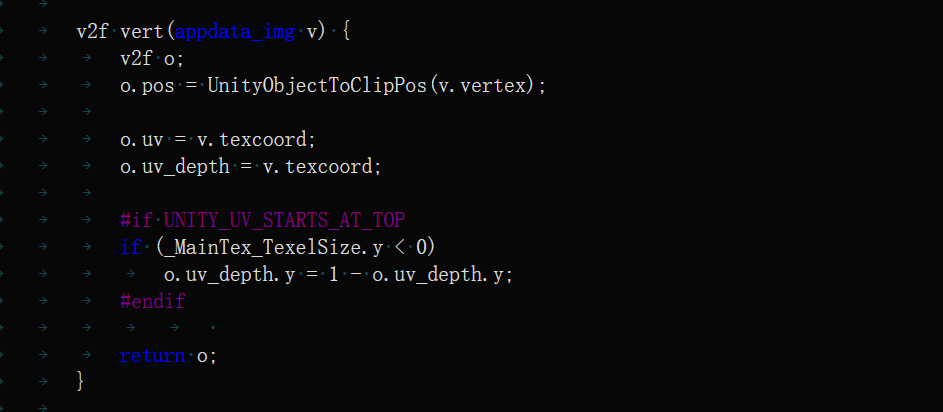
属性里没有声明视图投影矩阵和其逆矩阵属性的原因是Unity没有提供矩阵类型的属性，只能再CG代码块中定义这些矩阵



\_CameraDepthTexture是从Unity获取的深度纹理

\_PreviousViewProjectionMatrix是上一帧的视图投影矩阵

\_CurrentViewProjectionInverseMatrix是这一帧的视图投影矩阵的逆矩阵



顶点着色器中添加了对深度纹理的采样的纹理坐标变量，然后进行了一个平台差异化处理——如果Unity的纹理坐标系的原点在左上角，如果\_MainTex的纹理大小的y值小于0，说明纹理的原点在下方，将o.uv\_depth的y值翻转，使其与Unity的纹理坐标系一致。

文本

描述已自动生成

首先对深度纹理进行采样，获得[0,1]区间的深度值，通过深度值和下面这个公式可以反推裁剪空间下坐标



之后用当前视图投影矩阵的逆矩阵左乘这个裁剪空间下的坐标并齐次化求得世界空间下的坐标。此时就可以使用前一帧的视角投影矩阵对它进行变换，得到前一帧的标准化设备坐标（NDC），然后计算两帧在屏幕空间下的位置差，得到像素的速度velocity。然后根据运动速度和模糊大小调整纹理坐标，使用对像素的邻域像素进行采样，相加后除以迭代次数取平均值得到一个模糊的效果，通过\_BlurSize控制采样距离。

全局雾效

雾效是游戏里经常使用的一种效果。有这些方法实现：

1.Unity内置的基于距离的线性或指数雾效。效果有限，不能个性化操作

2.在Shader中添加#pragma multi\_compile\_fog，并使用相关宏。效果有限，需要为场景中所有物体添加相关渲染代码，不能个性化操作

3.经过一次屏幕后处理的全局雾效。自由度高，可以模拟均匀的雾、基于距离的线性/指数雾、基于高度的雾等

基于屏幕后处理的全局雾效的关键是根据深度纹理来重建每个像素在世界空间下的位置。

如何从深度纹理中重建世界坐标

重建世界坐标的思想：坐标系在的一个顶点坐标可以通过它相对于另一个顶点坐标的偏移量来获得。重建像素坐标同理。

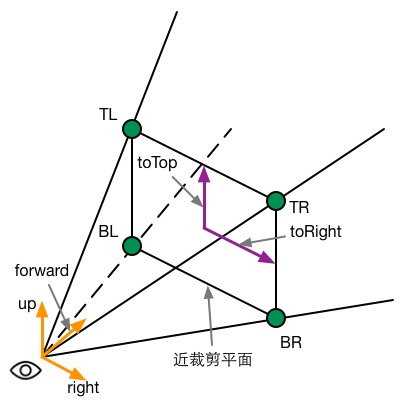
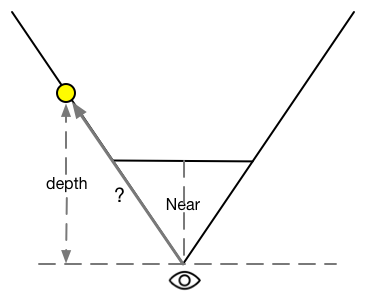
由这个思想，那么只需要知道摄像机在世界空间下的位置以及世界空间下该像素相对于摄像机的偏移量，两者相加即可得到该像素的世界坐标。这个过程可以用下面的代码表示：



\_worldSpaceCameraPos——相机的世界位置

linearDepth——从深度纹理得到的线性深度值

interpolateRay——顶点着色器输出并插值得到的射线（作用：表示方向）

求射线interpolateRay：

计算赋值向量：以相机点为起点，先确定图中的toTop和toRight即相机指向近平面正上边和正右边的投影

图表, 图示, 箱线图

描述已自动生成

计算指向4个角的向量：

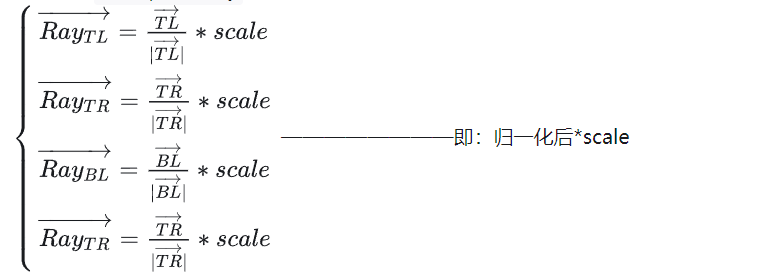
文本, 信件

描述已自动生成

因为深度值是z值depth，我们需要将深度值转变到这4个向量上的距离值dist，根据相似三角形性质获得等式



由此求出scale的大小，然后四个向量的实际长度都是scale，然后就可以求出四边形平面的四个角距离摄像机的实际的向量



雾的计算

简单雾效计算，雾有一个雾效系数f，作为混合原始颜色和雾颜色的混合系数



Unity内置的雾：雾效系数有三种算法：线性Linear、指数Exponential、指数的平方

Linear：根据z的大小线性计算f，d表示距离max为最大距离，min为最小距离



Exponential：d表示控制雾的浓度参数

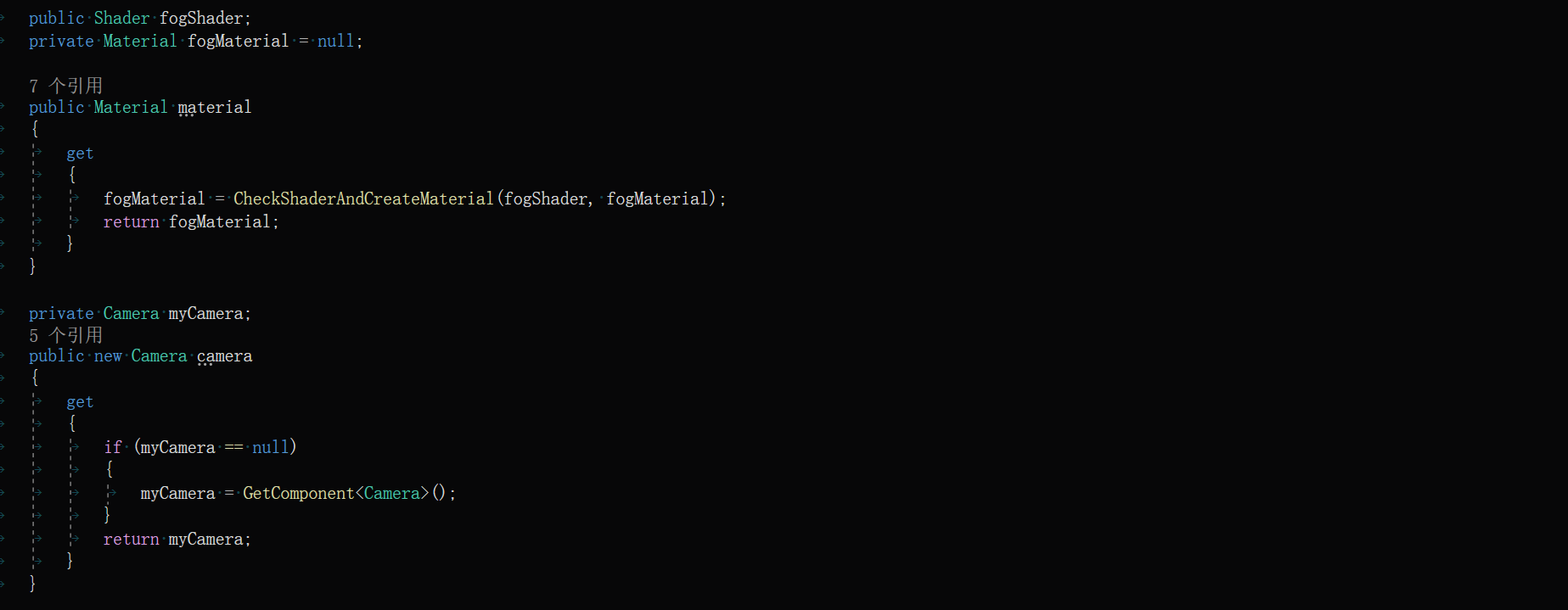


Exponential Squared：d表示控制雾的浓度参数



基于高度的线性雾计算

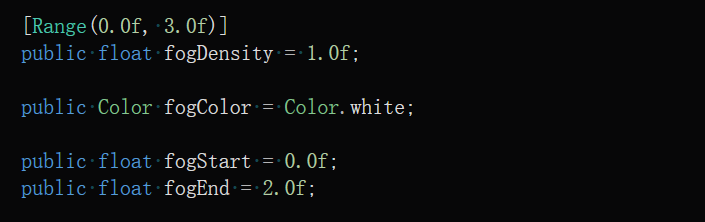
脚本部分：



形状

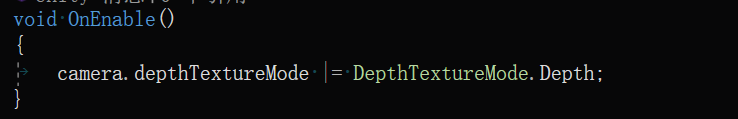
中度可信度描述已自动生成

分别存储Camera组件和它的Transform组件



定义模拟雾效的各个参数

FogDensity是控制雾的浓度，fogColor是雾的颜色，fogStart和fogEnd对应y方向上的min和max



获取摄像机深度纹理

文本

描述已自动生成

文本

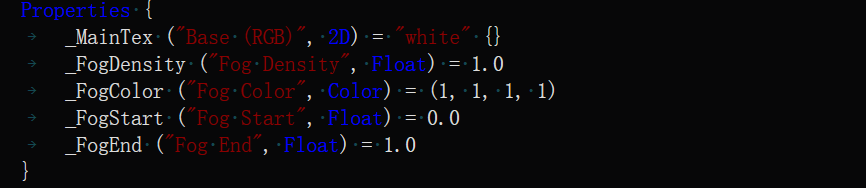
描述已自动生成

文本

描述已自动生成

首先计算近裁剪平面四个角对应的向量，按特定顺序存储在一个矩阵类型的变量中，计算过程就是之前推导过interpolateRay的计算过程，然后把结果和其他参数传递给材质，调用Graphics.Blit把渲染结果显示在屏幕上

Shader部分：

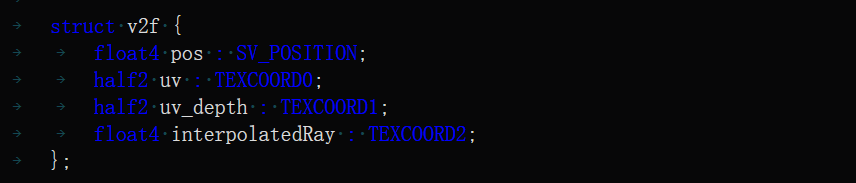


需要从脚本中接收的属性

图片包含 文本

描述已自动生成

存储四边形四个角向量的变量\_FrustumCornersRay， \_CameraDepthTexture是获取的深度纹理，这个获取深度纹理的过程是Unity完成的

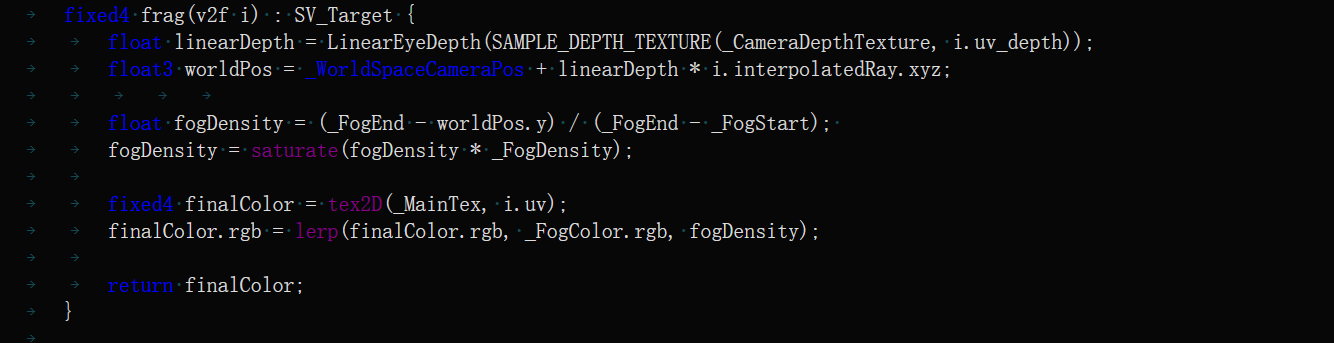


传入片元着色器的结构体中另外定义了一个新的变量interpolatedRay存储插值后的像素向量

文本

描述已自动生成

顶点着色器中首先传递顶点在裁剪空间下的坐标、纹理坐标，进行平台差异化处理，然后把平面分成相等大小的四个部分，决定当前顶点处于哪个部分，把这个部分的角的向量传给片元着色器



片元着色器中首先重建该像素在世界空间的位置，用SAMPLE\_DEPTH\_TEXTURE宏进行深度纹理采样，通过LinearEyeDepth得到视角空间下的线性深度值，然后与interpolatedRay相乘再和摄像机位置相加可以得到世界空间下的位置（也是相似三角形性质，通过它求出该像素在世界空间中的xy平面所在的位置，把摄像机垂直到该平面的向量和摄像机到该平面在视锥空间中对应角的向量用像素坐标进行插值就可以得到该像素在摄像机空间的坐标，用这个坐标加摄像机在世界空间中的坐标就可以得到该像素的世界坐标），然后计算雾效系数f最后将雾的颜色和原始颜色混合返回。

文本

描述已自动生成

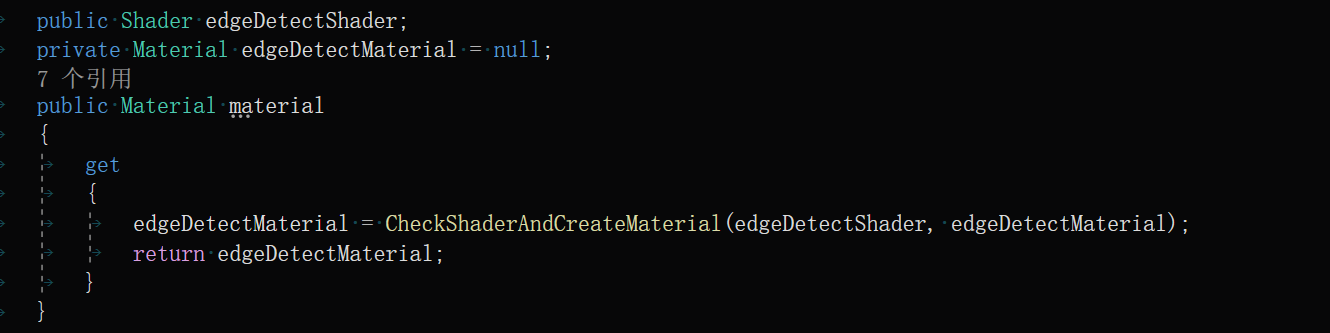
边缘检测

使用Sobel算子进行边缘检测会带来描边线，而如果在法线纹理和深度纹理上进行边缘检测就不会有这些黑边且图像不会受纹理和光照影响，可以避免检测误差

Roberts算子：本质是计算左上角和右下角的差值，乘右上角和左下角的差值。

边缘检测中设置一个可以变动的阈值比较和Roberts算子之间的差值，如果超过了这个阈值就认为存在一条边

脚本部分：



文本

描述已自动生成

edgesOnly：这个参数用于调整边缘强度。它的值范围是0.0到1.0，值越大，边缘效果越明显。

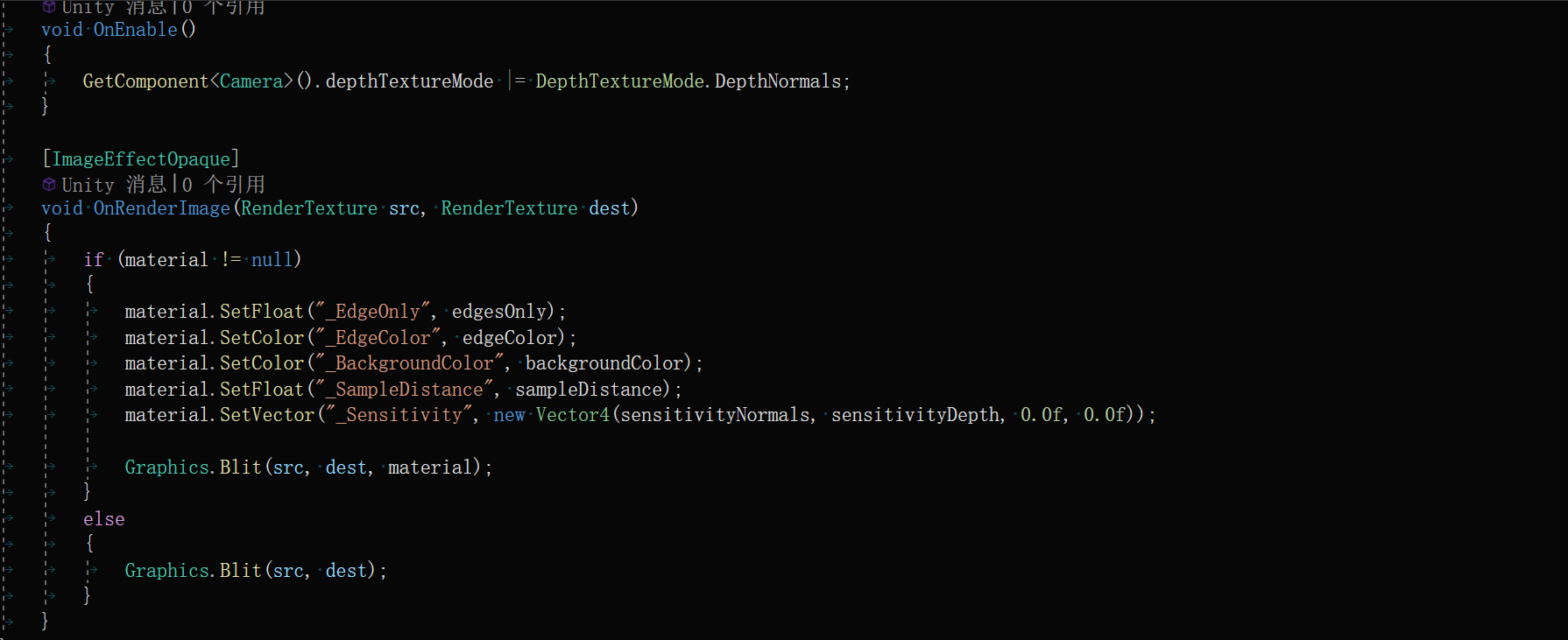
edgeColor用于设置描边的颜色。

backgroundColor用于设置背景色。

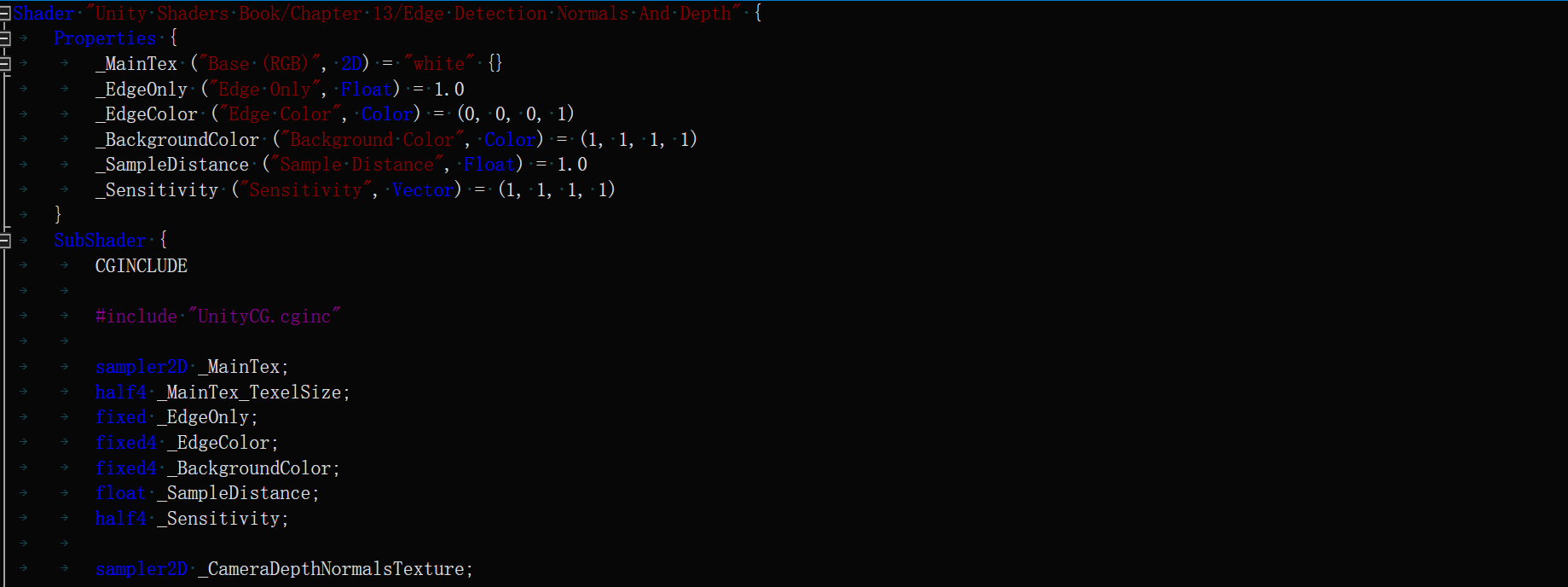
sampleDistance用于控制采样距离，值越大，描边越宽。

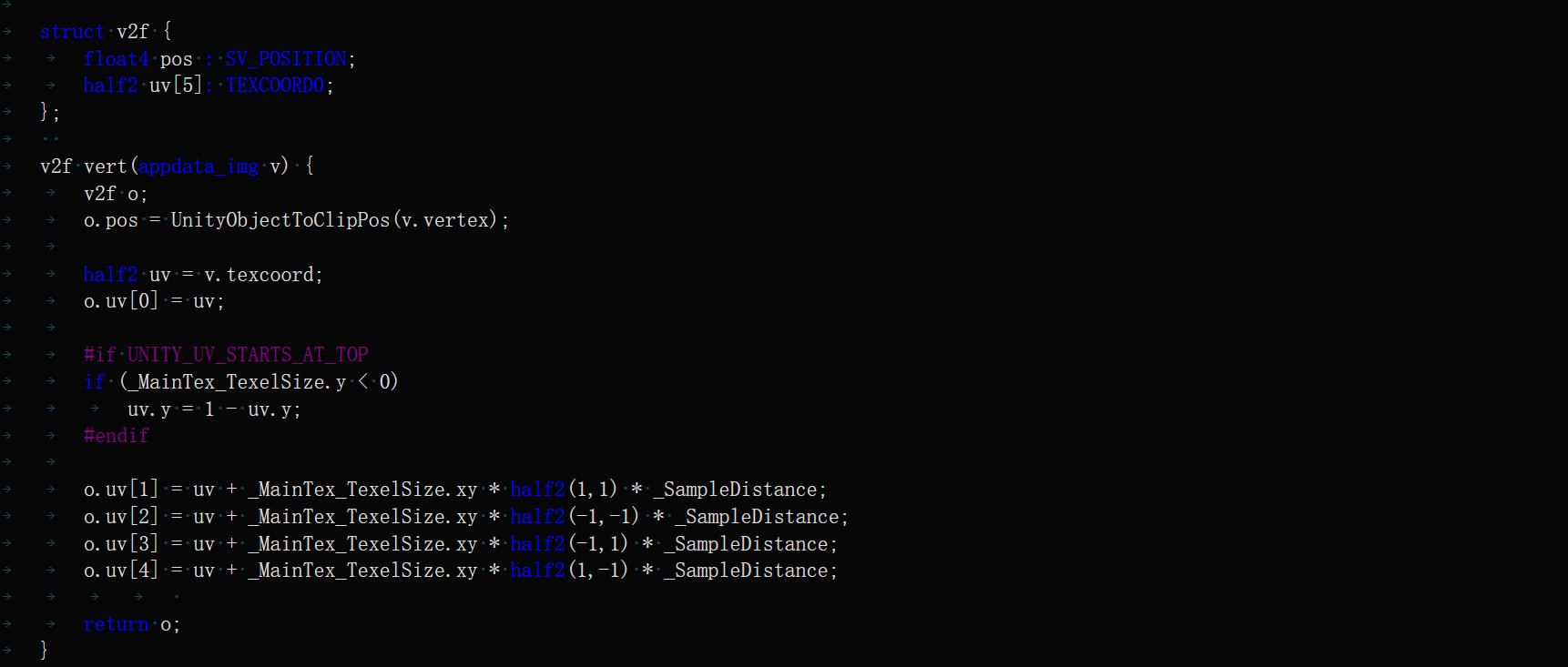
sensitivityDepth是深度灵敏度参数。当灵敏度参数大时，即使深度上很小的变化也会形成一条边。

sensitivityNormals是法线灵敏度参数。当灵敏度参数大时，即使法线上很小的变化也会形成一条边。

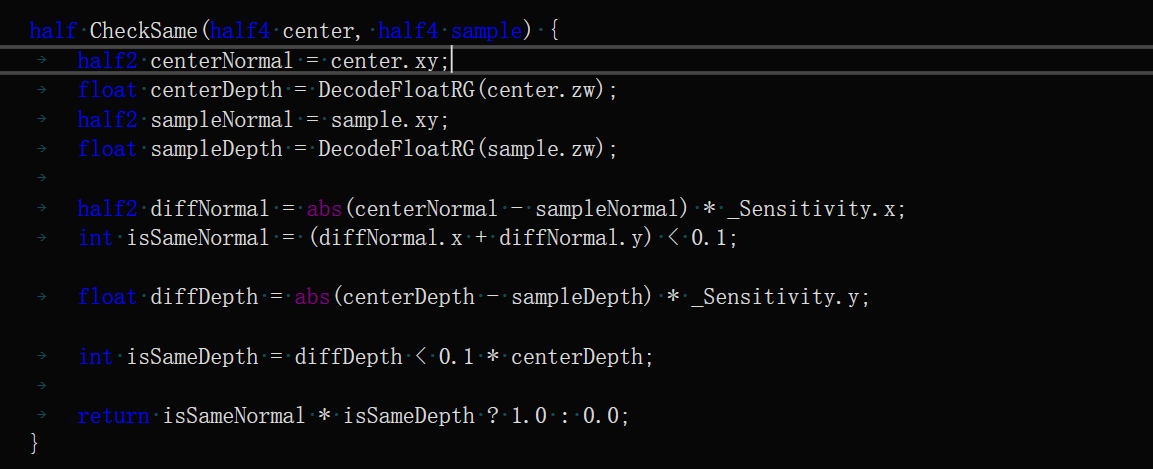


Shader部分：





顶点着色器中定义了一个五个元素的纹理坐标数组，uv[0]存储屏幕颜色图像的采样纹理，后面四个元素是深度采样纹理，并且对深度采样纹理做平台差异化处理



对输入参数进行处理解码得到法线的深度值和法线（并非真正的法线值，因为只需要比较两个采样值间的差异度），把两个采样点的对应值相减取绝对值，乘以灵敏度参数，把差异值的每个分量和阈值相比，如果小于阈值返回1反之0，以此判定差异是否明显

图片包含 文本

描述已自动生成

先用4个纹理坐标对深度法线纹理进行采样，再调用CheckSame函数分别计算对角线上两个纹理值的差值，CheckSame函数返回值要么是0要么是1，返回0表面两点间存在一条边界，反之返回1。

