建立一个基本的屏幕后处理系统

概念：屏幕后处理，通常指的是渲染完整个场景得到屏幕图像后，在对场景进行一系列操作，实现各种屏幕特效，如景深、运动模糊等。

Unity提供了 OnRenderImage 函数帮助我们得到渲染后的屏幕图像，它的声明如下：MonoBehaviour.OnRenderImage (RenderTexture src, RenderTexture dest)。当在脚本中声明此函数后， Unity 会把当前渲染得到的图像存储到第一个参数对应的源渲染纹理中，通过一系列的操作后，再把目标渲染纹理，即第二个参数对应的渲染纹理显示到屏幕上。

在 OnRenderImage 函数中，通常利用Graphics.Blit函数来完成渲染的处理。它有三种重载：

public static void Blit(Texture src, RenderTexture dest);

public static void Blit(Texture src, RenderTexture dest, Material mat, int pass = -1);

public static void Blit(Texture src, Material mat, int pass = -1);

其中参数src对应源纹理，在后处理技术中，这个参数通常就是当前屏幕的渲染纹理或是上一步处理后得到的渲染纹理。参数 dest 是目标纹理，如果它的值为 null，会直接将结果显示在屏幕上。mat 是使用的材质，我们会在它的Unity Shader中今后各种屏幕后处理操作，而src纹理会被传递给Shader 中名为 \_MainTex 的纹理属性。参数 pass 的默认值为 -1，表示会依次便利左右 Pass，否则会调用指定索引的 Pass。

在默认情况下，OnRenderImage 函数会在所有的不透明物体和透明物体的 Pass 执行完后调用，但是我们希望在所有不透明 Pass 执行完立即调用该函数，从而不对透明物体产生影响，此时我们可以在OnRenderImage函数调用前添加 ImageEffectOpaque属性来实现这样的目的。

实现屏幕后处理效果的过程：

1.在相机中添加一个用于屏幕后处理的脚本，在这个脚本中，我们会使用 OnRenderImage 函数来获得当前屏幕的纹理。

2.调用Graphics.Blit函数 使用特定的 Unity Shader 来对当前图像进行处理，再把返回的结果显示到屏幕上。

Ps：对于复杂的效果，可能会多次调用 Graphic.Blit 函数，并将上一次的输出作为这次的输入。

文本

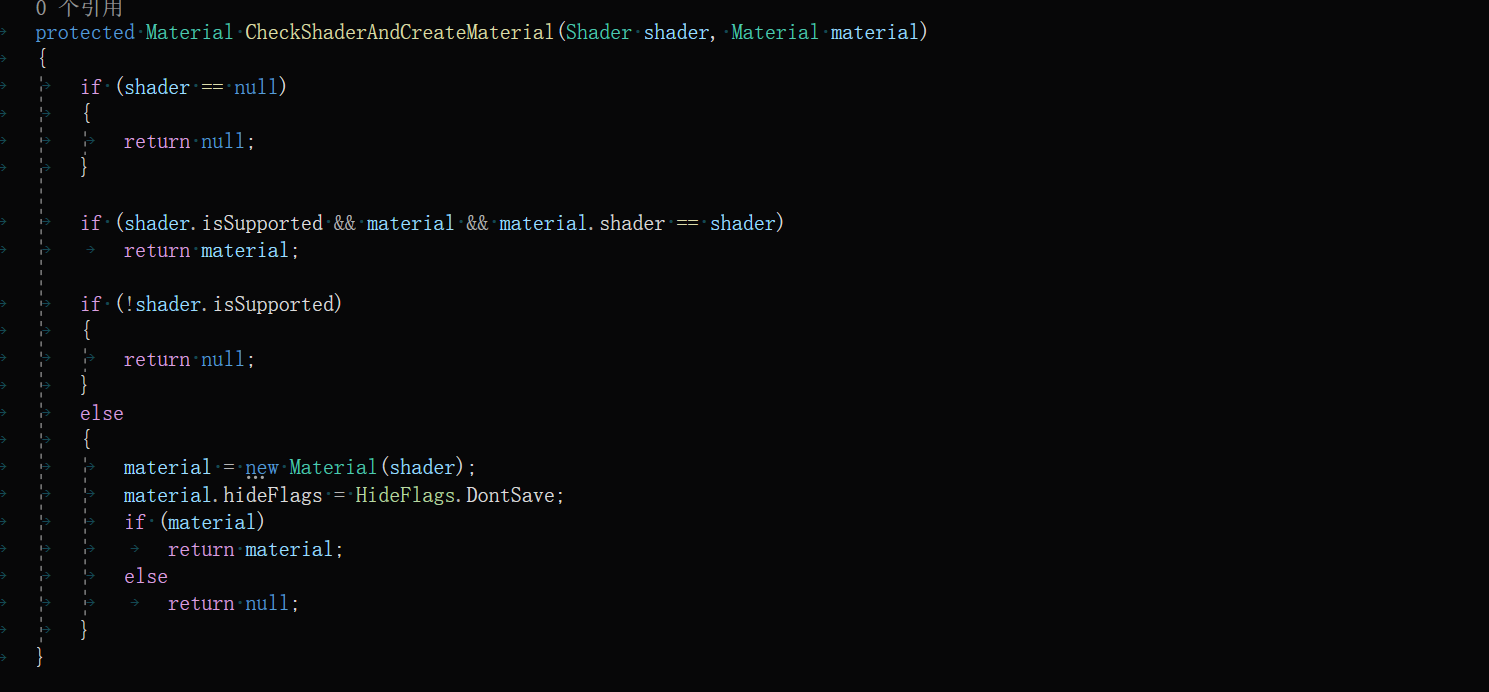
低可信度描述已自动生成

CheckResources在Start中被调用用于检查平台是否支持图像效果

文本

描述已自动生成

CheckResources() 方法调用 CheckSupport() 方法来检查平台是否支持图像效果。如果不支持，则调用 NotSupported() 方法禁用脚本；CheckSupport() 方法使用 SystemInfo.supportsImageEffects 属性来检查平台是否支持图像效果。如果不支持，则输出警告信息，并返回 false；NotSupported() 方法：禁用当前脚本组件。



CheckShaderAndCreateMaterial 的主要目的是指定一个Shader来创建一个用于处理渲染纹理的材质。如果Shader是可用的，它会创建一个新的材质并返回。如果没有任何一个SubShader和FallBack可用或者着色器为空，它会返回null。如果已经存在一个使用该着色器的材质，它会直接返回这个材质。

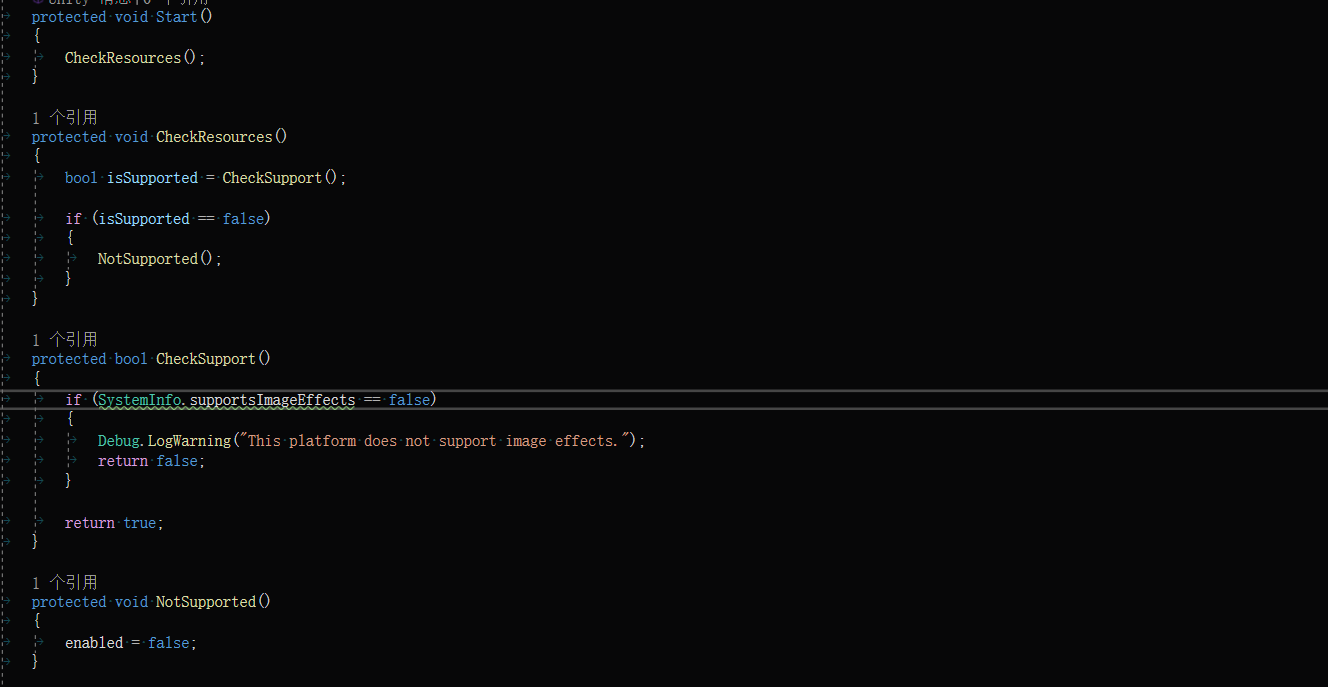
if (shader.isSupported && material && material.shader == shader) { return material; }：这一行代码检查传入的着色器是否被支持，以及传入的材质是否存在且使用的是传入的着色器。如果这些条件都满足，那么方法直接返回这个材质。

Ps：若支持着色器中的任何SubShader或任何默认返回FallBack，则返回true

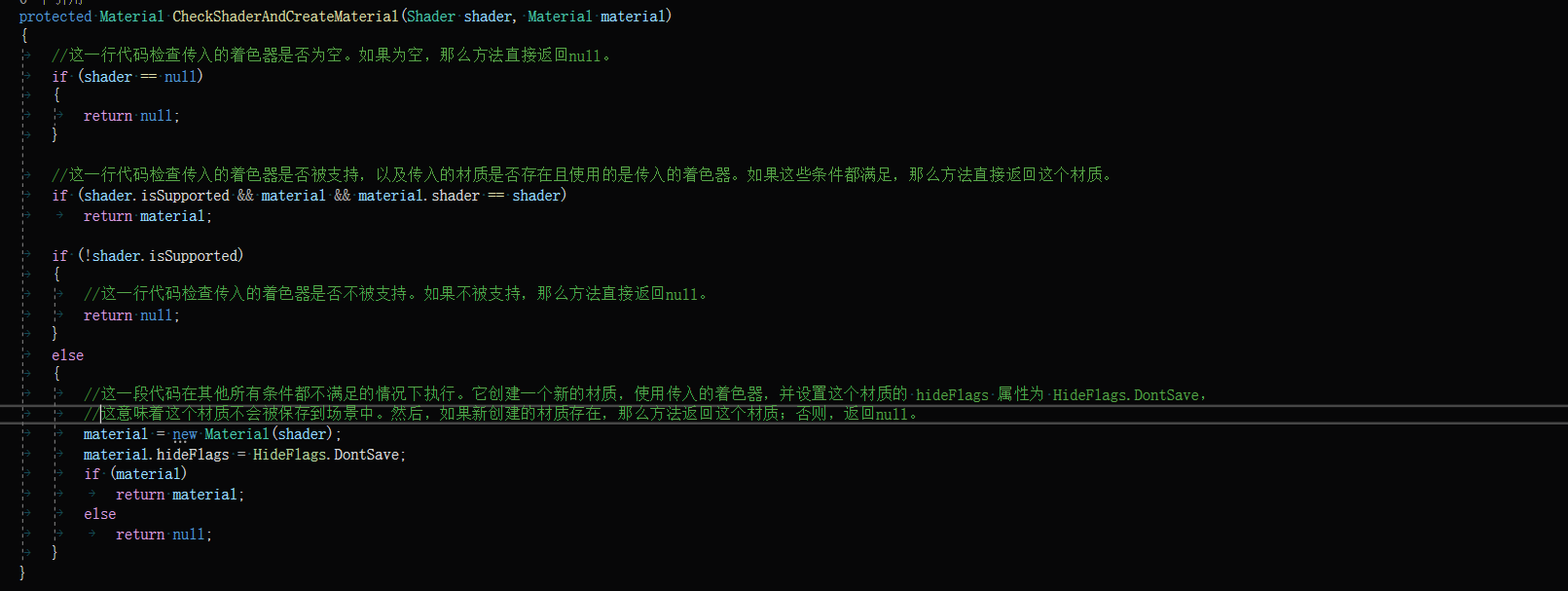
图片包含 文本

描述已自动生成

因为Unity引擎的更新。在早期的Unity版本中，一些平台可能不支持图像效果或渲染纹理，所以需要进行检查。但是在现代的Unity版本中，几乎所有的平台都支持图像效果和渲染纹理，所以 SystemInfo.supportsImageEffects 和 SystemInfo.supportsRenderTextures 这两个API已经被标记为过时，并且总是返回 true。因此，原本用于检查平台是否支持图像效果的 CheckSupport 方法现在总是返回 true，而 NotSupported 方法因为 CheckSupport 总是返回 true 而永远不会被调用。因此这些代码在更新后变得没有作用：



所以脚本中最终只剩下下面这个函数：



调整屏幕的亮度、饱和度和对比度

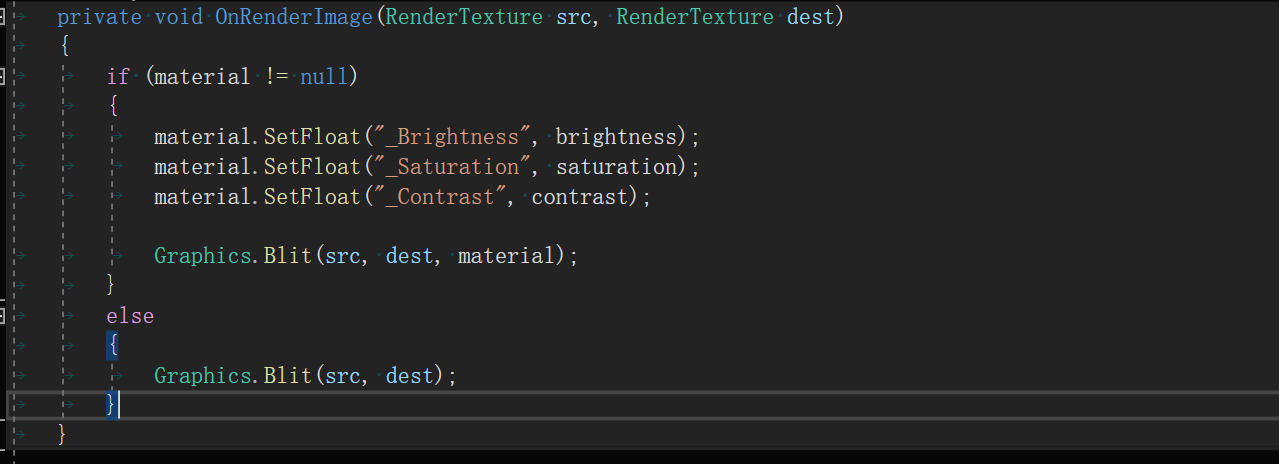
脚本部分：

文本

描述已自动生成

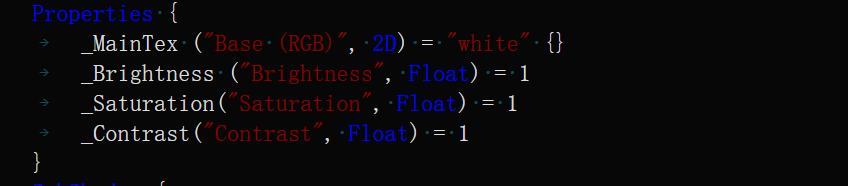
briSatConShader是被传入CheckShaderAndCreateMaterial的shader，briSatConMaterial是CheckShaderAndCreateMaterial的返回材质，brightness、

saturation、contrast分别对应可调整的亮度、饱和度、对比度，利用Range特性提供了合适的变化区间，然后定义了OnRenderImage进行真正的特效处理：

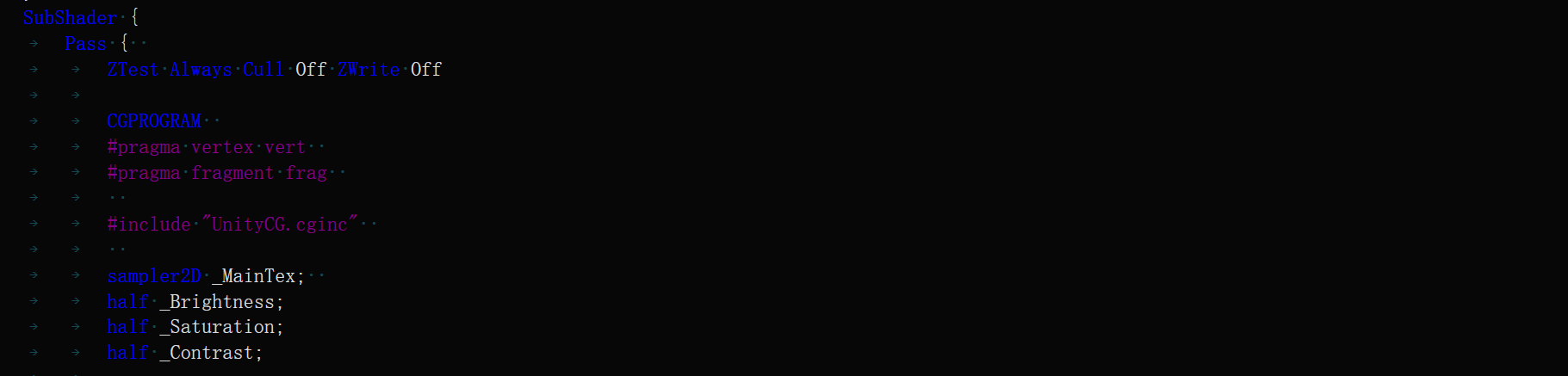


如果材质可用，就把亮度、饱和度、对比度的各项参数传递给材质，然后使用Graphics.Blit函数对材质进行处理，否则直接把图像显示到屏幕上

Shader部分：



下面三个属性分别用于接收脚本传入的亮度、对比度、饱和度，\_MainTex用于接收Graphics.Blit的传入的shader，即src参数

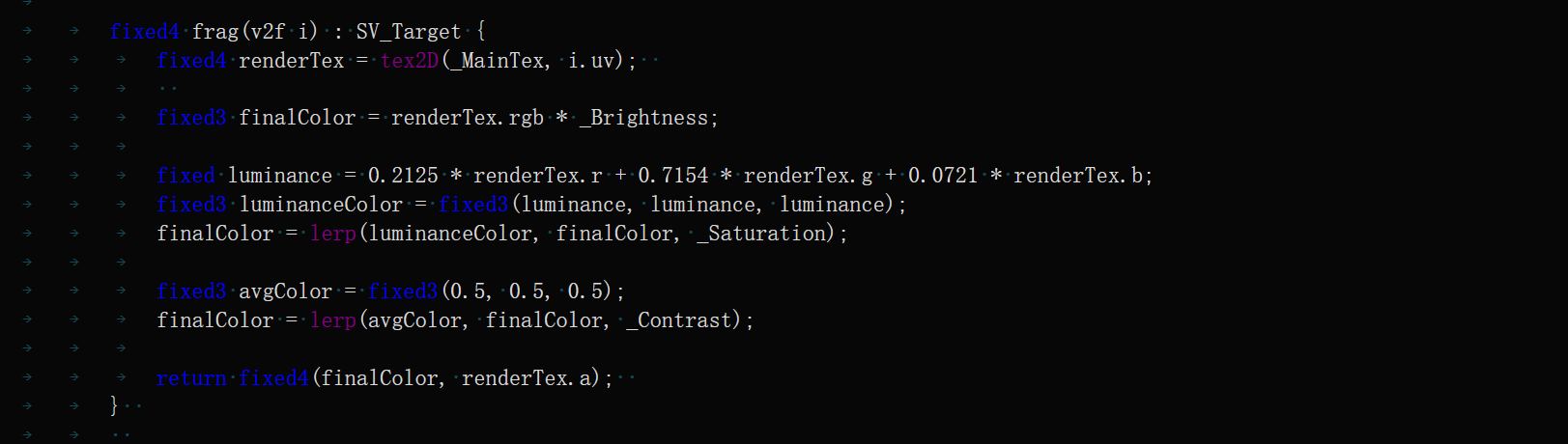


关闭深度测试（ZTest Always）和深度写入（ZWrite Off）是因为这个着色器主要用于处理图像的亮度、饱和度和对比度，而不是用于渲染具有深度感的3D物体。

文本

描述已自动生成

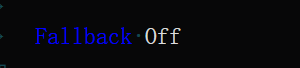
屏幕特效使用的顶点着色器代码通常比较简单，只需要把顶点转换到裁剪空间下、把正确的纹理坐标传递给片元着色器即可，以便对屏幕图像进行采样。



片元着色器中线把采样得到的颜色和亮度相乘用于，这一步得到灰度图；然后使用rgb三个分量分别乘一个对应的系数求出像素颜色的亮度吗，这一步得到一个灰度图；然后把图像亮度颜色和灰度颜色使用饱和度插值以调整饱和度，；最后定义了一个平均颜色avgColor，它的R、G、B三个通道的颜色值都是0.5，这个颜色可以看作是一个中性的灰色，因为它的亮度和饱和度都是中等的；将上一步调整对比度过后的颜色与平均颜色用对比度系数插值，最后将调整好的颜色和透明度组合成一个新的颜色返回。

Ps：。

1. 在一个灰度颜色中，R、G、B三个通道的颜色值都是相等的，等于亮度值
2. 2. 当插值系数大于1时，可以增加图像的亮度、饱和度和对比度，使图像更亮、更鲜艳或更清晰



最后把Fallback设置为Off，希望在shader无法运行时报错而不是切换到默认shader

边缘检测效果

边缘的特性：相邻像素间梯度较大。

什么是卷积：

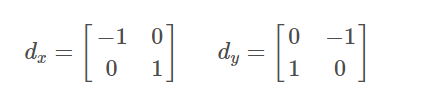
图像处理中卷积操作指使用一个卷积核对一张图像中的每个像素进行一系列操作。卷积核通常是一个四方形网状结构，该区域内每个方格都有一个权重值。当对图像中的某个像素进行卷积时，会把卷积核的中心放置在该像素上，翻转核（旋转180度）之后再以此计算核中每个元素和其覆盖的图像像素值的乘积并求和（每个像素加权相乘累加），得到的结果就是该位置的新像素值。

[数字图像处理:边缘检测(Edge detection) - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/59640437)

常见的边缘检测的算子（卷积核）

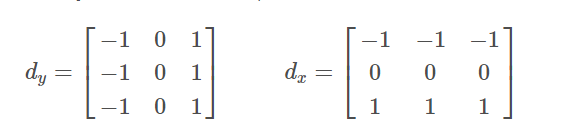
用卷积核表示梯度

Roberts算子：



Roberts算子的模板分为水平方向和垂直方向，从其模板可以看出，Roberts算子能较好的增强正负45度的图像边缘。

Prewitt算子：



Prewitt算子的边缘检测结果在水平方向和垂直方向均比Robert算子更加明显。Prewitt算子适合用来识别噪声较多、灰度渐变的图像。

Sobel算子：

电脑屏幕的照片

中度可信度描述已自动生成

Sobel算子在Prewitt算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓。Sobel算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息。

整体梯度计算

进行边缘检测时需要对每个像素分别进行一次卷积计算，即每个像素加权相乘累加，得到两个方向上的梯度值Gx和Gy，整体的梯度可由这个公式计算得到：

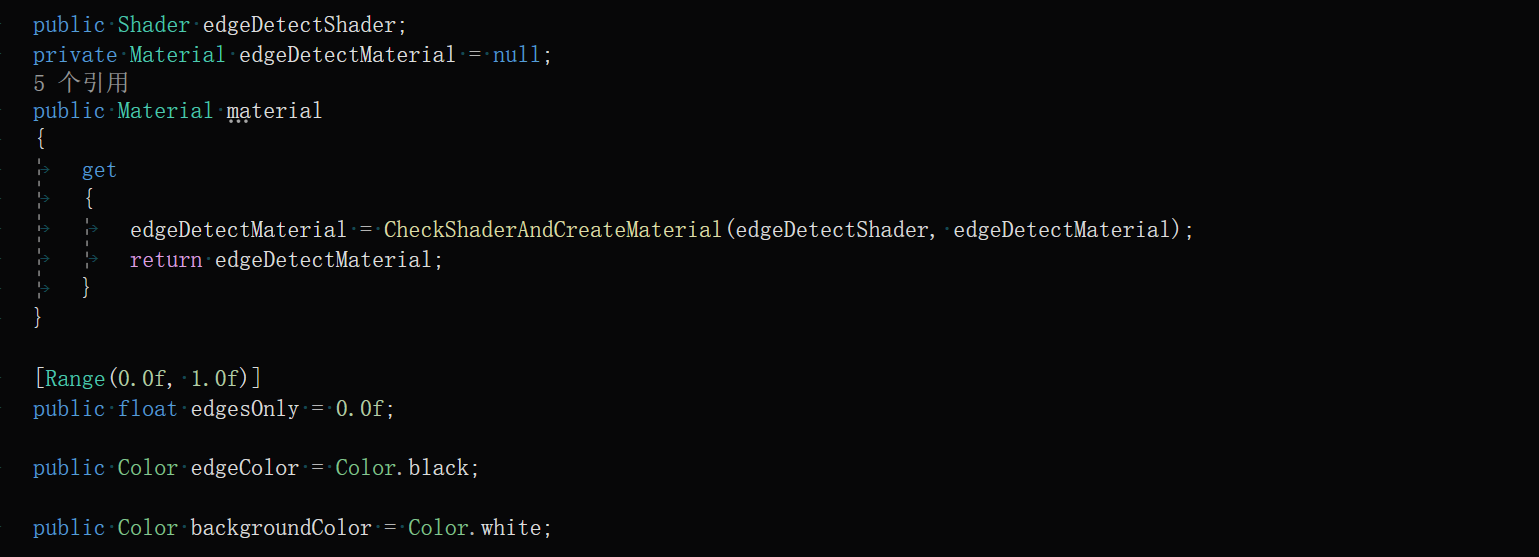


为了避免开根号导致性能消耗增加，使用下面的公式代替这个开根号的公式：

图片包含 图示

描述已自动生成

脚本部分：

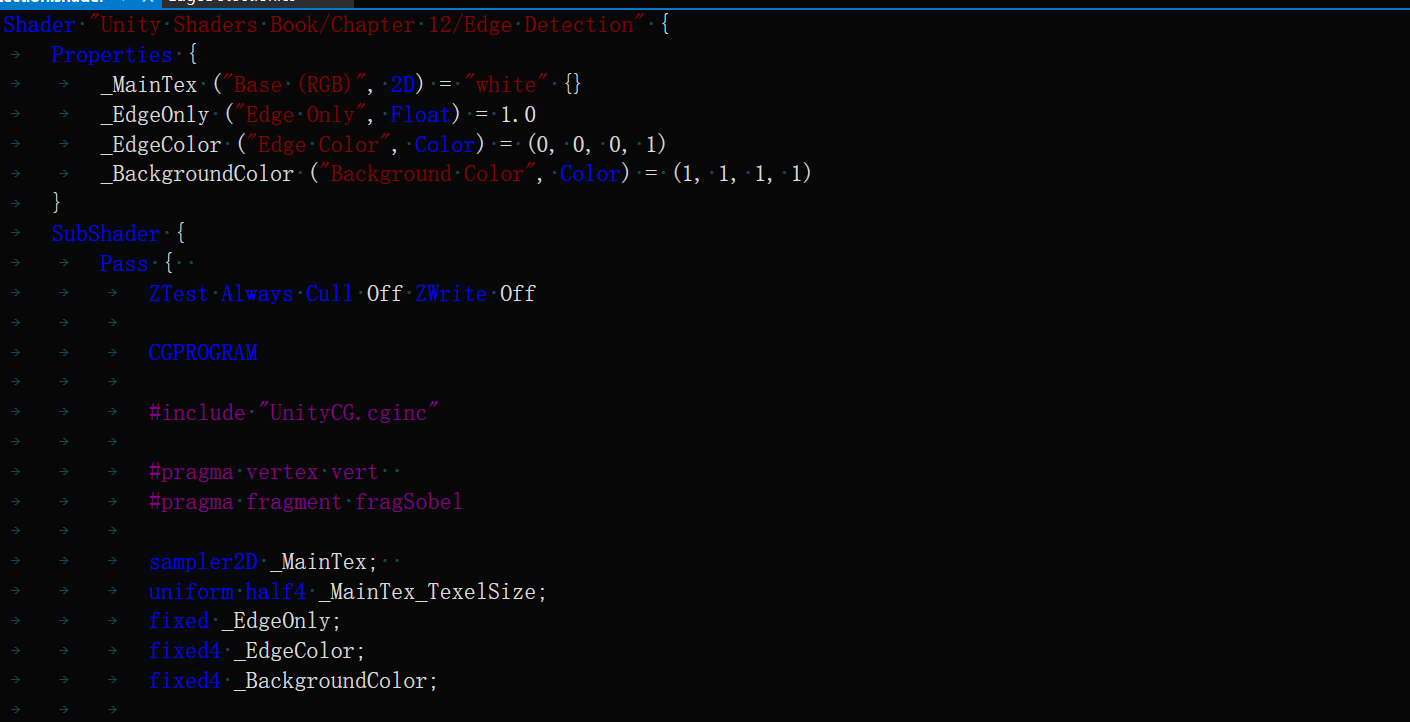


EdgeDetectShader是赋给创建的材质shader，edgeDetectMaterial是返回的材质，edgesOnly用于调整边缘线强度，edgeColor用于调整边缘线颜色，backgroudColor用于调整背景颜色



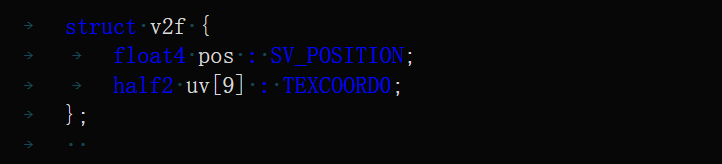
每当OnRenderImage被调用时会检查材质是否可用，若可用就把参数传给材质，再调用Graphics.Blit进行处理，否则把原图像显示到屏幕上

Shader部分：



定义属性接收脚本里的对应参数，设置Pass的相关模式，定义变量访问各项属性，除了对应的四个变量以外，额外声明了一个变量\_MainTex\_TexelSize，由于卷积需要对相邻区域内的纹理进行采样，因此需要利用\_MainTex\_TexelSize来计算各个相邻区域的纹理坐标。

Ps：xxx\_TexelSize是Unity为我们提供的访问xxx纹理对应的每个纹素的大小，例如一张512x512的纹理，该值大小为1/512。



v2f结构体中定义了纹理数组，作为一个卷积核，对应使用Sobel算子采样时需要的9个邻域纹理坐标。由于顶点着色器到片元着色器的插值是线性的，因此这样的转移并不影响纹理坐标的计算结果。

文本

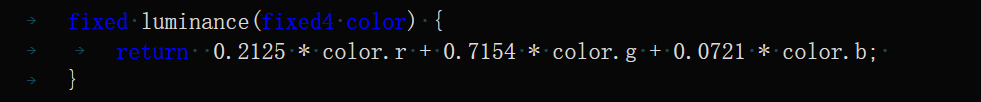
描述已自动生成

每个顶点作为卷积核的中心，相邻的八个顶点的纹理坐标也都被存入纹理数组中

文本

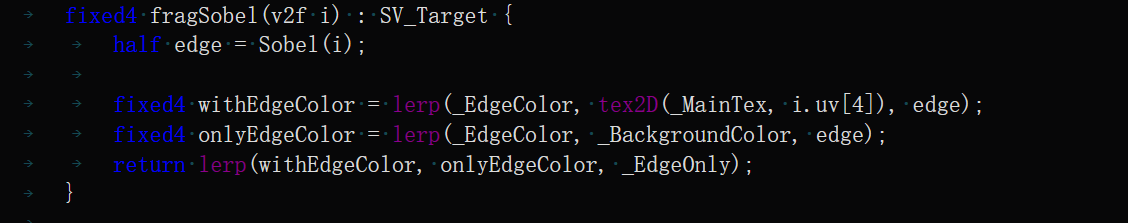
描述已自动生成

Sobel函数用于计算当前像素的梯度值，计算了for函数里计算了每个像素横向和纵向上的梯度值并且求出其整体梯度，edge越小则越可能是个点。



求灰度luminance被封装成函数

Ps：luminance=0.2125 \* color.r + 0.7154 \* color.g + 0.0721 \* color.b是一个常用的求灰度的公式，灰度公式的rgb分量系数加起来为1



片元着色器首先获取该像素的梯度值，并且利用该值分别计算了背景为原图和纯色下的颜色值，然后利用边缘线强度\_EdgeOnly在两者之间插值得到最终的像素值。

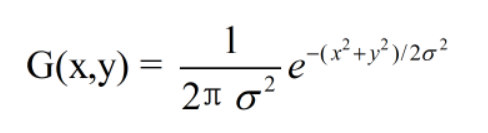


关闭FallBack

高斯模糊

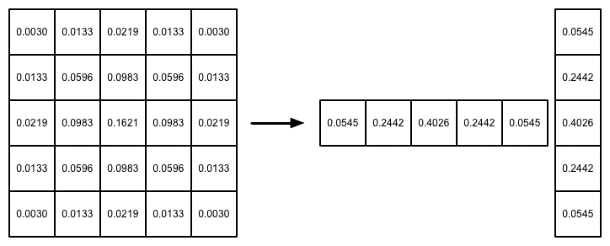
模糊的实现也需要用到卷积，比如均值模糊的卷积核中各个元素的权重都相等，比如中值模糊卷积后的像素值是卷积核中所有元素排序后的中位数，高斯模糊相比前两者更高级。

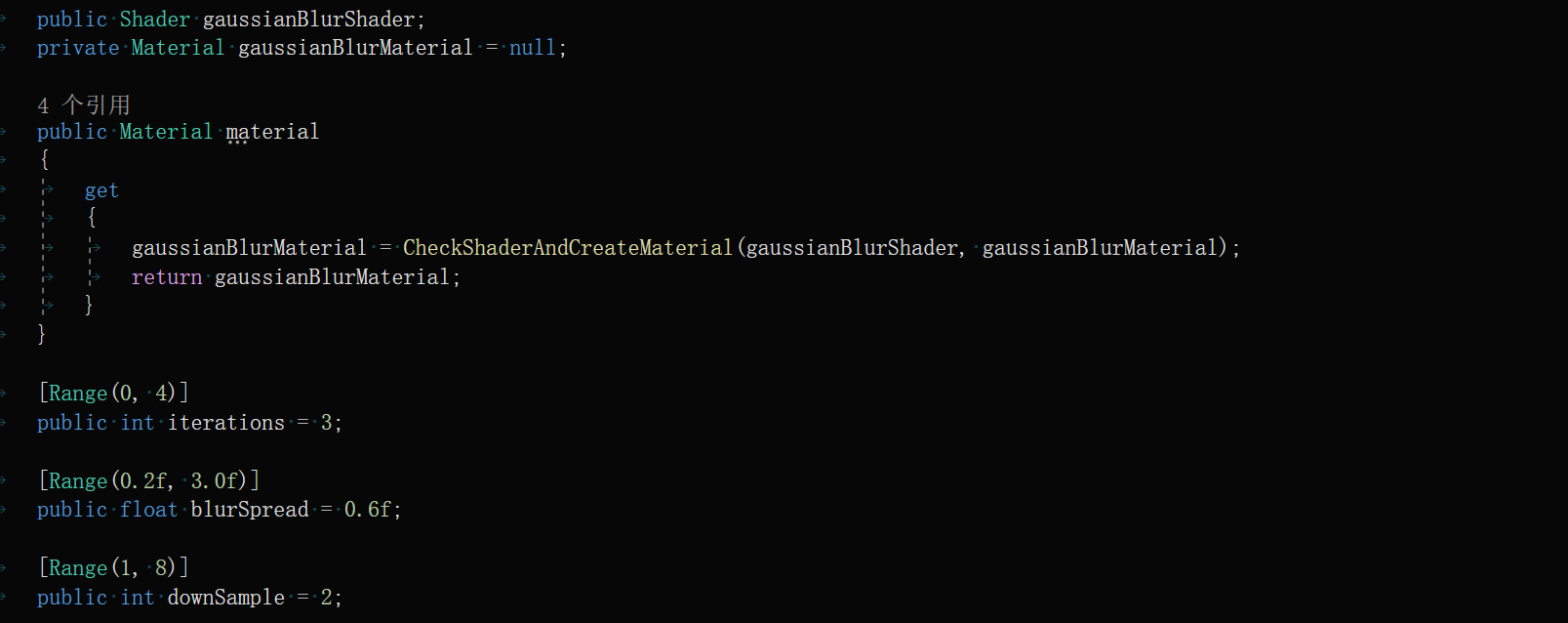
高斯模糊使用的卷积核名为高斯核，高斯核也是一个正方形大小的滤波核，其中每个元素的计算都是基于下面的高斯方程：



σ 是标准差，x和y对应当前位置到卷积核的整数距离。为了保证滤波后图像不会变暗。需要对高斯核中的权重进行归一化，这样可保证所有权重的和为标准方差。高斯方程模拟邻域每个像素对当前处理像素的影响程度——距离越近，影响越大。

高斯核的维数越高、模糊程度越大。当维数过高时会带来很大的性能负担，这个问题可以把二维的高斯函数拆分成两个一维的高斯函数，每个一维高斯核中每一格存入对应层的权重值之和，因为两个一维高斯函数中有很多重复的函数，所以存储一个一维高斯核实际只用存层数n/2+1个数据



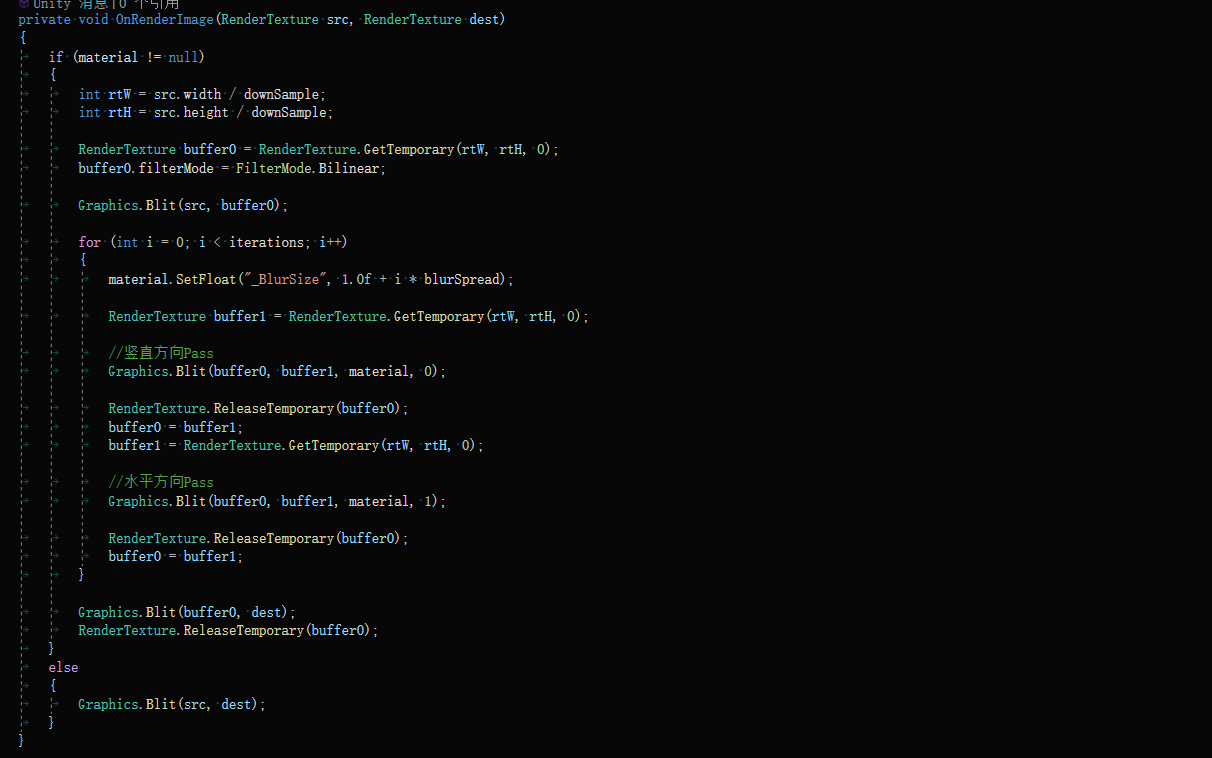


iterations、blurSpread、downSample分别是高斯模糊迭代次数（维数）、模糊范围、缩放系数

ps：

1.什么是降采样？对过滤高频后的讯号降采样M倍，换言之，保留原讯号中间隔为M的取样点

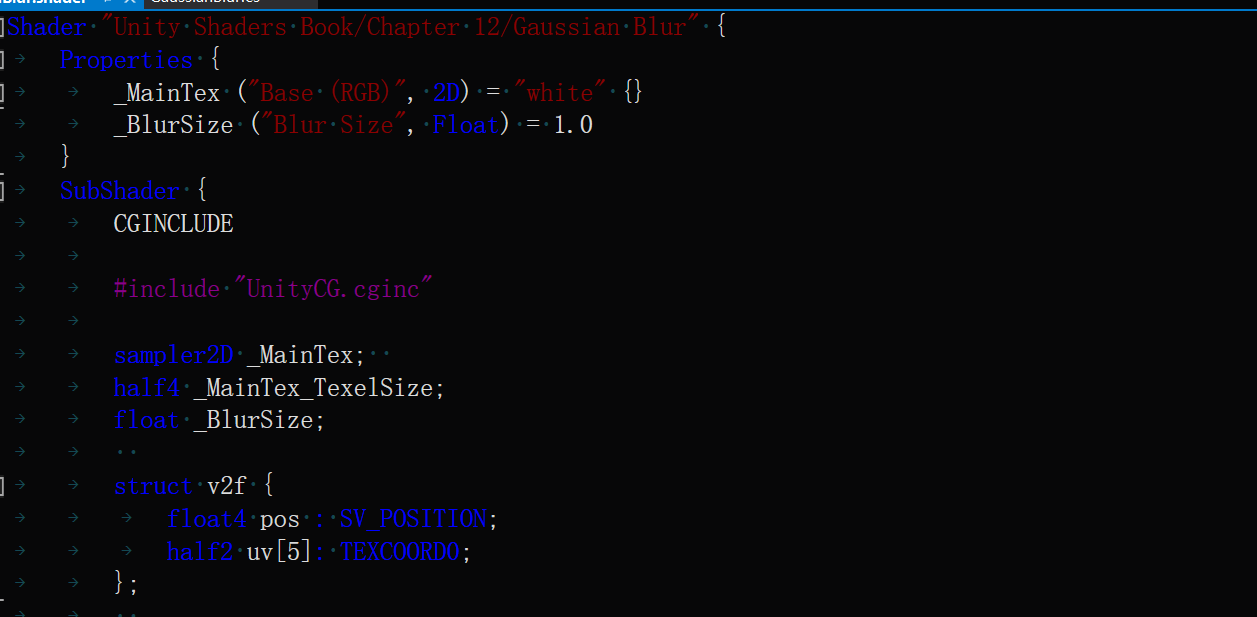
2.在高斯核维度不变的情况下，blurSpread越大模糊程度越高，但采样数不会受到影响，但过大会造成虚影，而downSample越大需要处理的像素数越少，也能进一步提高模糊程度，但过大会使得图像像素化。



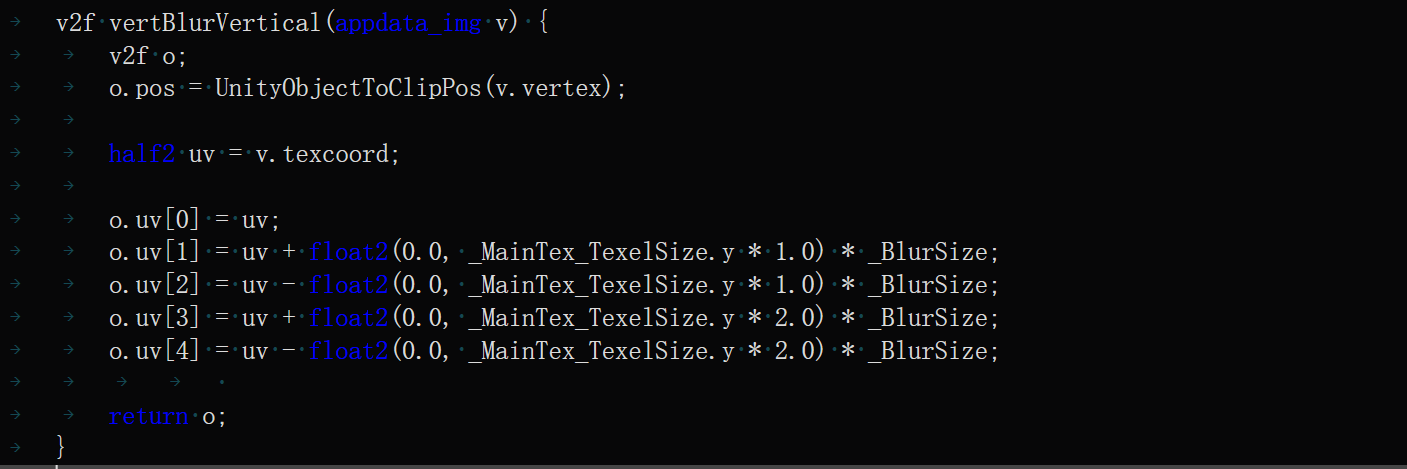
首先计算了进行降采样之后的横向和纵向的像素数目，然后在迭代开始前定义了第一个缓存buffer0，然后把该临时渲染纹理的滤波模式设置为双线性，并把src中的图像存储到其中，。迭代过程中定义了第二个缓存buffer1，执行第一个pass的时候，输入是buffer0，输出是buffer1，执行完毕后释放buffer0再把buffer1中存的结果存储到buffer0中，然后重新分配buffer1，再调用第二个pass重复上述过程。结束迭代后，buffer0将存储的最终的图像显示到屏幕上并释放缓存。

Ps：双线性滤波：双线性滤波是一种常用的纹理过滤方法，它在处理图像时可以提供较好的视觉效果。当图像被缩放或者旋转时，双线性滤波可以通过对周围的四个像素进行加权平均来生成新的像素值，从而使得图像看起来更加平滑，减少了锯齿或者马赛克的出现。在这段代码中，双线性滤波被用于处理降采样后的图像，以保证模糊效果的质量。

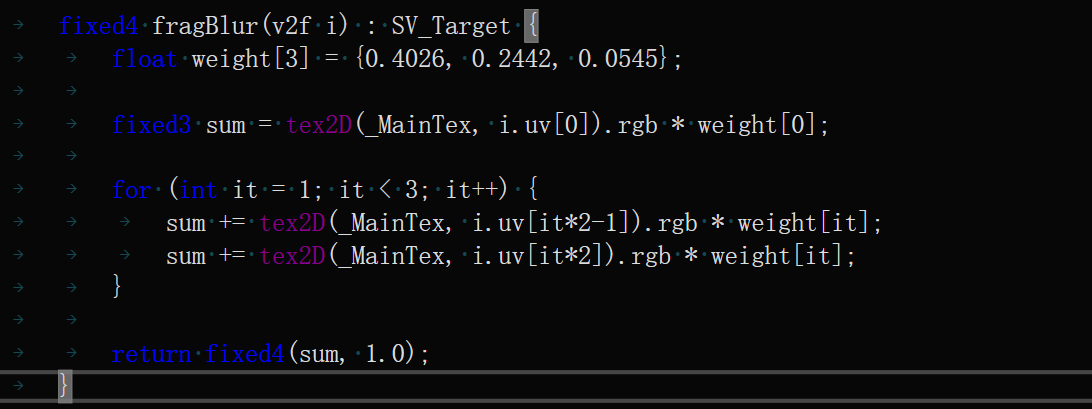
Shader部分（5x5的高斯核）：



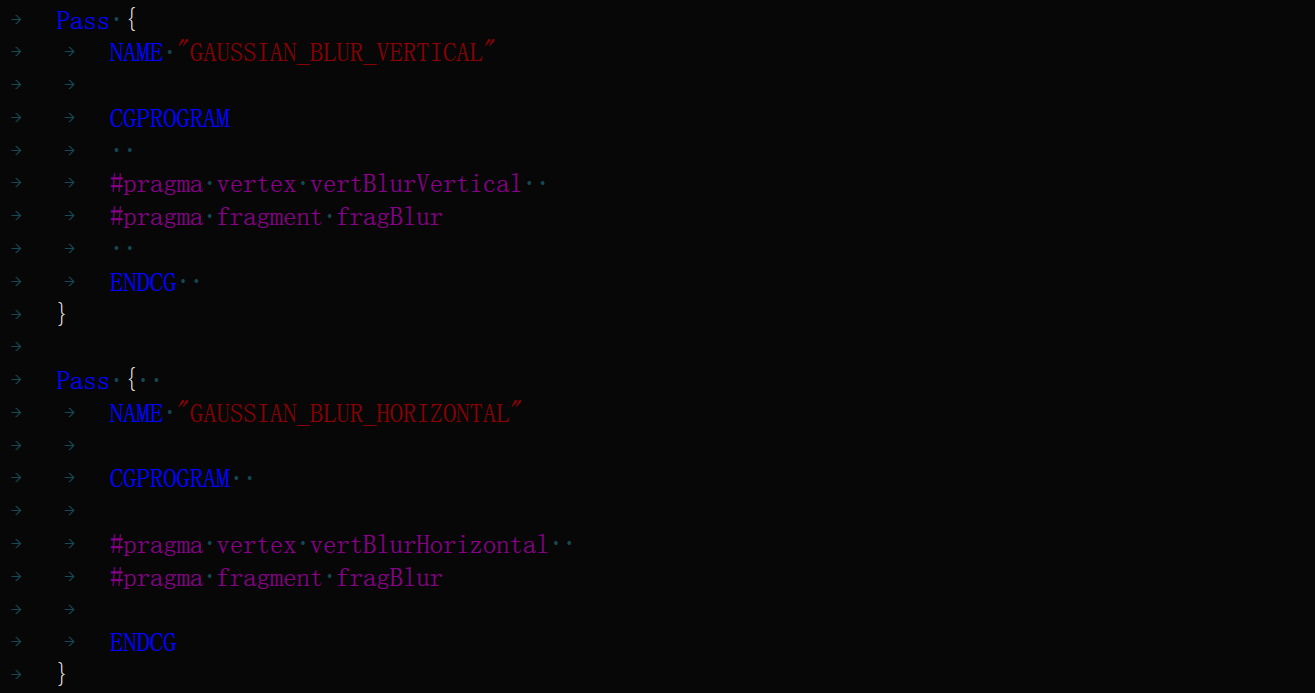
\_BlurSize控制高斯模糊的范围。定义顶点着色器输出的结构体中的uv用于存储一维的高斯核，其中uv[0]存储当前的采样纹理，剩余四个坐标则是高斯模糊中对邻域采样时使用的纹理坐标。



uv[0]存储了当前像素的纹理坐标，而uv[1]到uv[4]存储了当前像素在竖直方向上偏移一定距离的邻域像素的纹理坐标



因为是5x5的高斯核，可以被拆成两个一维的高斯核，且两者相乘得到结果和二维的高斯核一样，又由于它的对称性，所以只用存储三个权重，然后对目标像素进行采样然后乘权重，再对邻域像素进行采样乘权重，最终五个值相加，就可以得到当前方向上的滤波结果。



定义横向和纵向上的pass进行高斯模糊处理。

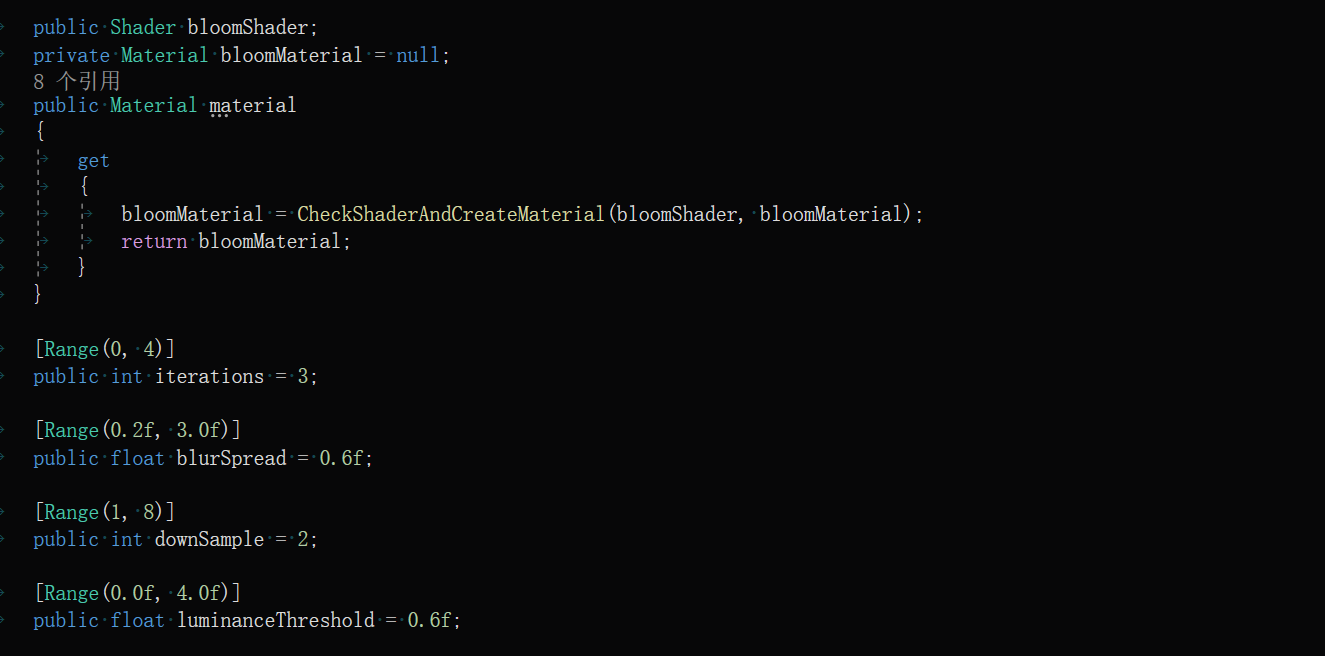
Ps：Pass可以定义名字，这样可以再其他Shader中直接通过它们的名字来使用该Pass，而不需要重新编写代码。



Bloom效果

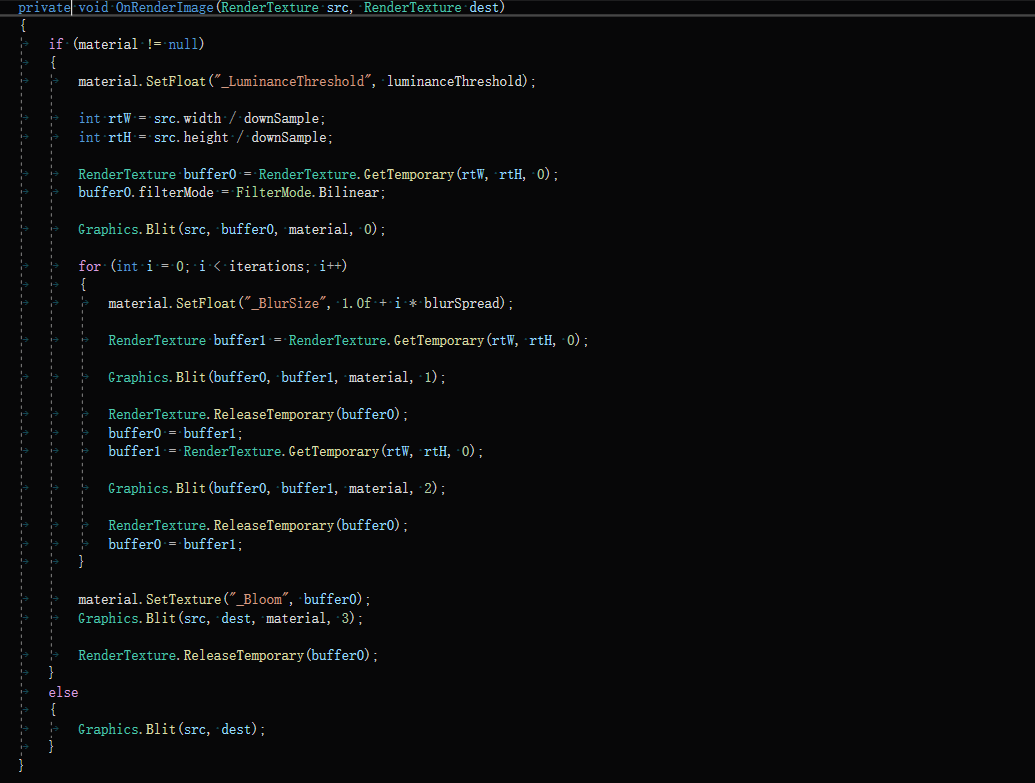
脚本部分：

这种特效可以模拟真实相机的一种图像效果，它让画面中较亮的区域“扩散”到周围区域中，造成一种朦胧的效果



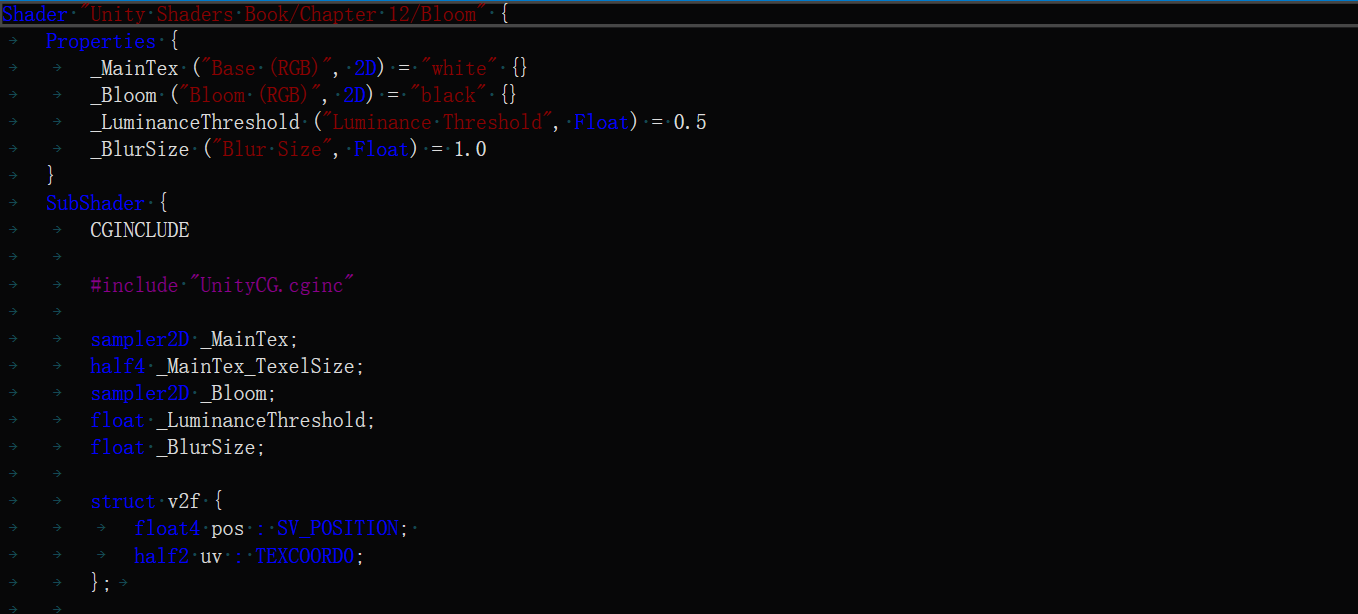
Bloom建立在高斯模糊的基础上，因此在高斯模糊的脚本的基础上加了一个luminanceThreshold变量来控制提取较亮区域时使用的阈值大小

Ps：如果开启HDR，图像的亮度值是可以高于1的



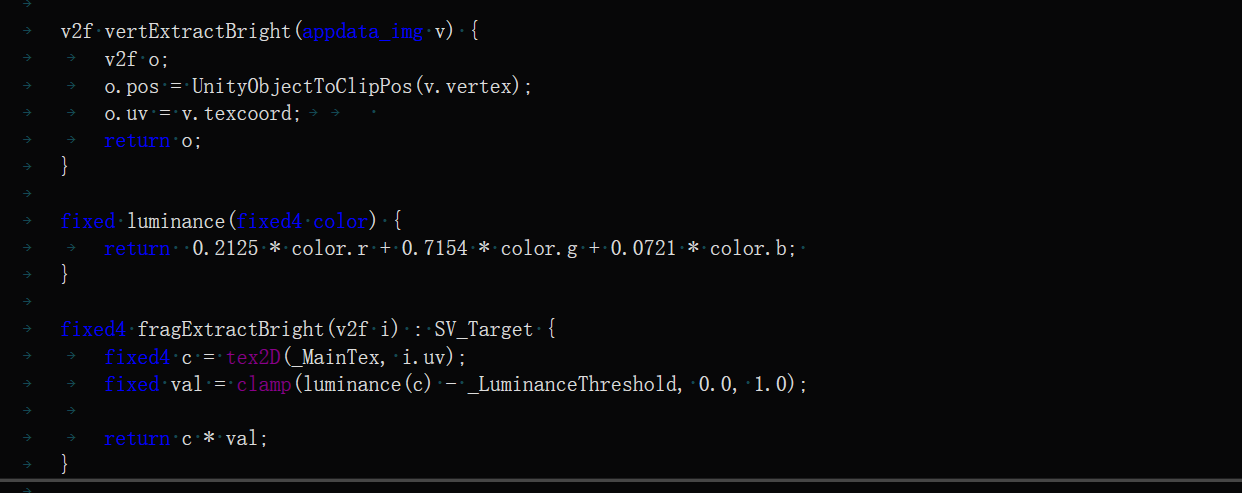
OnRenderImage中将图像亮度的变量赋值给shader的\_LuminanceThreshold属性，然后没有直接进行降采样，而是调用Graphics.Blit(src, buffer0, material, 0)来使用Shader中的第一个Pass提取图像中较亮的区域并储存在buffer0中，然后进行高斯模糊迭代处理，这些Pass对应shader的第二三个pass，模糊后的较亮区域存储在buffer0中，再把buffer0传递给材质的\_Bloom纹理属性，并使用第四个pass进行混合，将结果输出到屏幕，同时释放buffer

Shader部分：

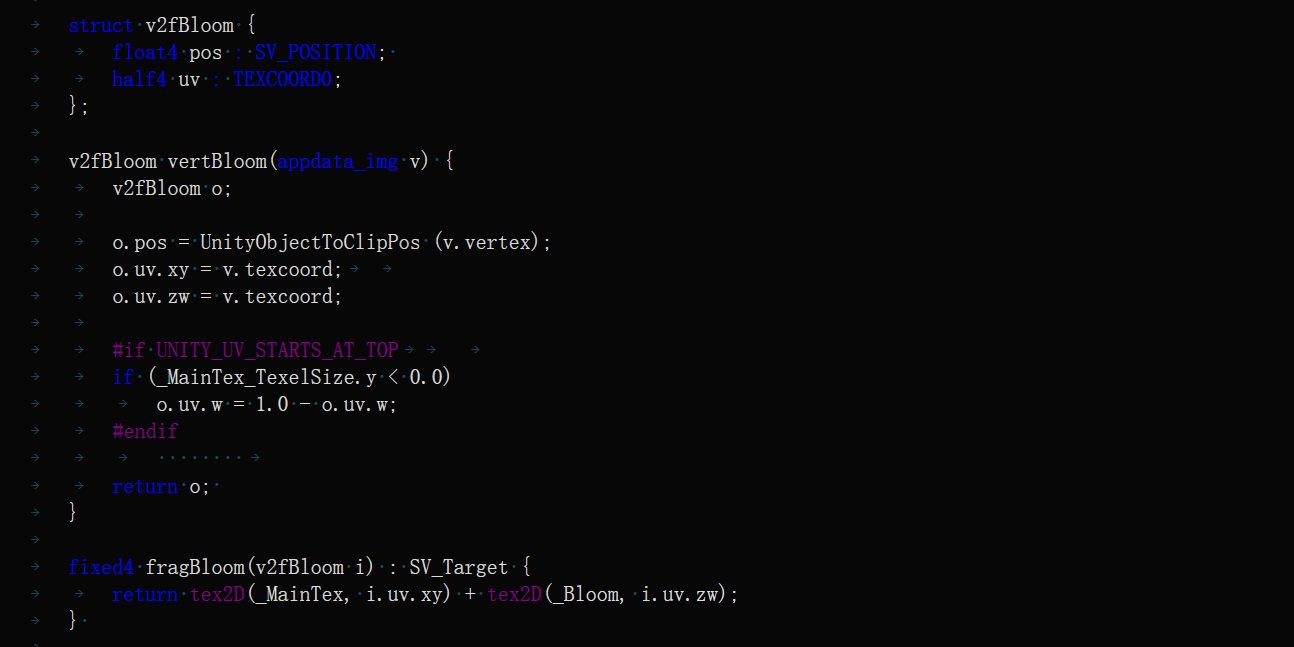


形状

中度可信度描述已自动生成



这是第一个pass的顶点着色器部分和片元着色器部分，luminance函数同样是用来求灰度颜色的，顶点着色器与之前相同，片元着色器中将采样得到的灰度颜色（亮度）减去阈值，并把结果截取到0-1范围内。然后把该值和原像素值相乘得到提取后的亮度区域。



这是第四个pass，顶点着色器与之前不同，定义了两个纹理坐标并且都存储在同一个类型为half4的变量uv中，xy分量对应主纹理，zw分量对应Bloom纹理（模糊后较量区域的纹理坐标），然后进行平台差异化处理



第二第三个Pass直接使用了高斯模糊中定义的两个Pass

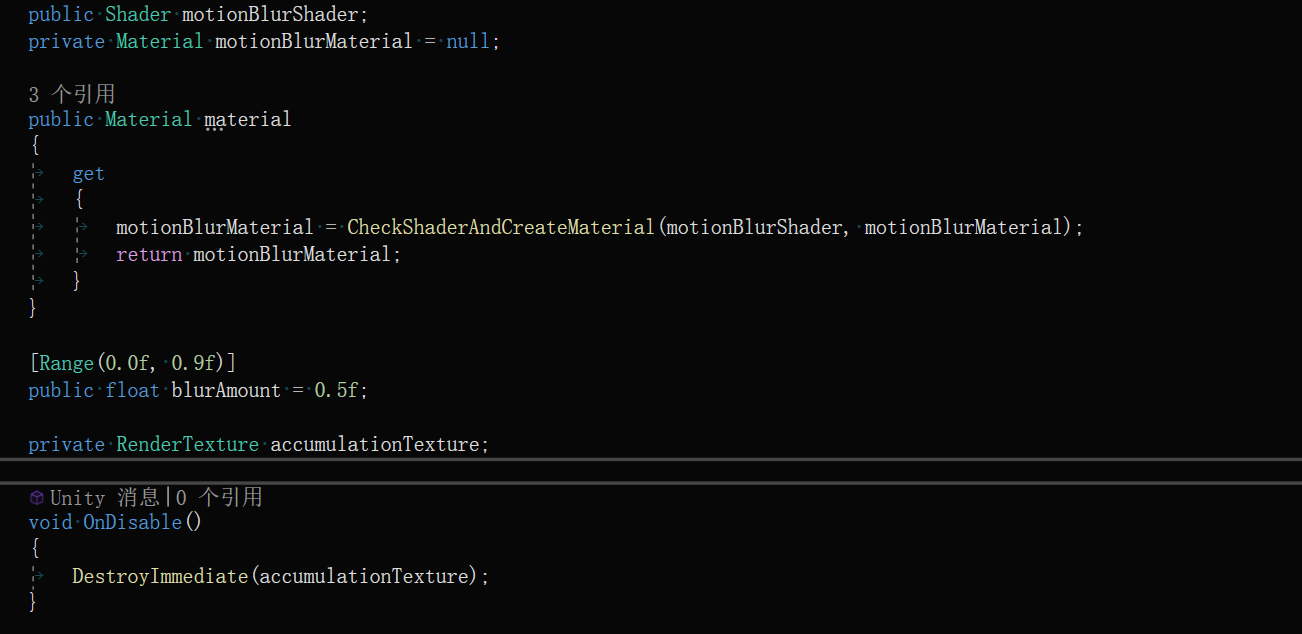
运动模糊

实现运动模糊的方法：累积缓存，混合多张连续的图像然后取它们之间的平均值作为最后的图像，但是这样性能消耗很大，因为在一帧中需要渲染多次场景；速度缓存，缓存中存储各个像素当前运动速度，然后利用这个速度决定模糊的方向和大小。

利用累积缓存实现运动模糊：

不需要一帧内渲染多次场景，但需要保存之前的渲染效果。即不断把当前的渲染图像叠加到之前的渲染图像中，从而产生一种运动轨迹的效果。这样做不会有那么大的性能开销，但是模糊效果会受到影响。

脚本部分：

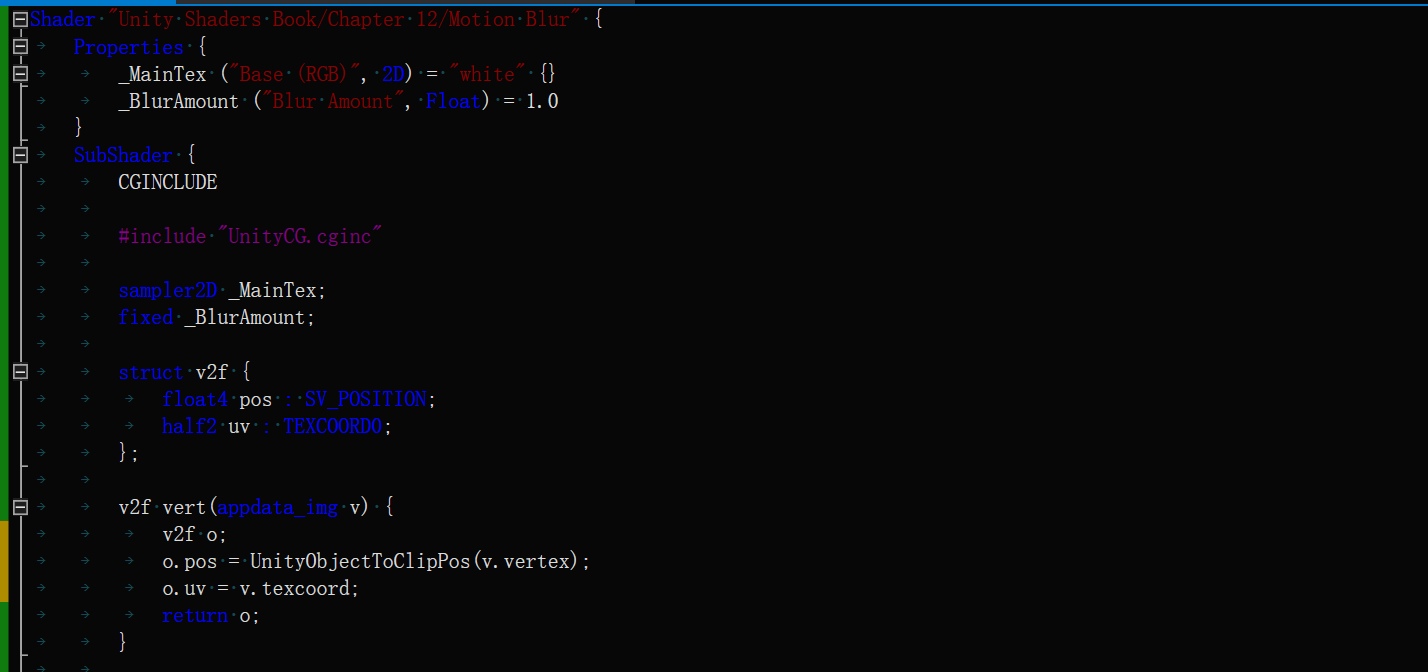


定义了blurAmount用于控制模糊程度的数量和accumulationTexture用于保存之前图像叠加的效果

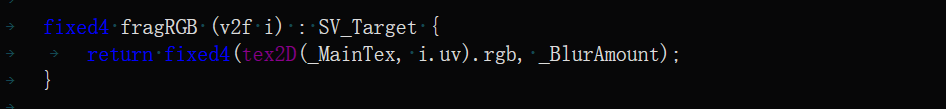


RenderTexture.MarkRestoreExpected()在旧版本的Unity中用于指示预期将进行RenderTexture恢复操作。在移动图形仿真模式下，当执行RenderTexture“恢复”操作时，Unity会发出警告。如果在不先进行清除或丢弃的情况下渲染到纹理，就会执行恢复操作。对于许多移动GPU和多GPU系统来说，这是一项代价高昂的操作，应该予以避免。但是，如果渲染效果要求必须进行RenderTexture恢复，则可以调用该函数来指示Unity恢复操作是预期行为，不要发出警告。然而，在新版本的Unity中，RenderTexture.MarkRestoreExpected()方法已经被标记为过时。可以直接删除这行代码，不会对你的程序产生任何影响。这是因为Unity的渲染管线和资源管理机制已经进行了更新，使得RenderTexture.MarkRestoreExpected()方法不再需要。

Shader部分;



另一个属性为混合系数，顶点着色器与之前一样



Pass0的片元着色器，用于更新渲染纹理的RGB通道部分

图形用户界面, 应用程序

描述已自动生成

用于更新渲染纹理的A通道部分

