**屏幕后处理效果**

**屏幕后处理**

**What:**

在渲染完整个场景得到屏幕图像后，再对图像进行一系列操作，实现各种屏幕特效。

**Why:**

景深，运动模糊等效果

**How:**

**Unity中如何实现屏幕后处理效果？**

1. 在OnRenderImage(RenderTexture src, RenderTexture dest)函数中，第一个参数对应的是源渲染纹理，第二个参数对应的是渲染目标纹理；
2. 在OnRenderImage函数中，我们通常是利用Graphics.Blit函数来完成对渲染纹理的处理。

Blit(Texture src, RenderTexture dest, Material mat, int pass = -1)

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，表示将会一次调用Shader内的所有Pass。否则，只会调用给定索引的Pass。

1. 默认情况下，OnRenderImage函数会在所有的不透明和透明的Pass执行完毕后被调用，但有时我们希望在不透明的Pass执行完毕后立即调用OnRenderImage函数，从而不对透明物体产生影响。我们可以在OnRenderImage函数前添加ImageEffectOpaque属性来实现这个目的。

**例子：用于屏幕后处理效果的基类**

**实例：调整屏幕的亮度，饱和度和对比度**

**卷积**

**What:**

卷积操作指的就是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格结构，该区域内每个方格都有一个权重值。

**Why:**

**How:**

**边缘检测**

卷积操作的神奇之处在于选择的卷积核。

边是如何形成的？如果相邻像素之间存在差别明显的颜色，亮度，纹理等属性，我们就会认为它们之间应该有一条边界。这种相邻像素之间的插值可以用梯度来表示，边缘处的梯度绝对值会比较大。有3种常见的边缘检测算子。我们对每个像素分别进行一次卷积计算，得到两个方向上的梯度值Gx和Gy，整体梯度用G = |Gx| + |Gy|表示。

**实例：边缘检测**

本例的边缘检测仅仅利用屏幕颜色信息，而在实际应用中，为了更准确的边缘信息，我们往往会在屏幕的深度纹理和法线纹理上进行边缘检测。

**高斯模糊**

高斯模糊同样利用了卷积计算，它使用的卷积核名为高斯核。高斯核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高斯方程。

我们可以把二维高斯函数拆分成两个一维函数。进一步观察，一维高斯核包含了很多重复的权重(对称相等)。

**实例：高斯模糊**

**Bloom效果**

**What:**

模拟真实摄像机的一种图像效果，它让画面中较亮的区域扩散到周围的区域中，造成一种朦胧的效果。

**Why:**

**How:**

**Bloom的实现原理**

根据阈值提取出图像中的较亮区域，把它们存储在一张渲染纹理中，再利用高斯模糊对这张渲染纹理进行模糊处理，模拟光线扩散的效果，最后再将其和原图像进行混合，得到最终效果。

运动模糊

What:

如果在摄像机曝光时，拍摄场景发生了变化，就会产生模糊的画面。

Why:

How:

如何实现运动模糊？

运动模糊的实现有多种方法，一种实现方法是利用一块累积缓存来混合多张连续的图像。这种方法性能消耗较大。另一种应用广发的方法是创建和使用速度缓存，这个缓存中存储了各个像素的当前的运动速度，然后利用该值来决定模糊的方向和大小。

本例我们使用类似上述第一种方法的实现来模拟运动模糊效果。我们不需要在一帧中把场景渲染多次，但需要保存之前的渲染结果，不断把当前的渲染图像叠加到之前的渲染图像中，产生一种运动轨迹的视觉效果。

Unity的Image Effect包中包含更多特效的实现。