# **在Unity中做一个淡入式的屏幕虚化**

作者：[枸杞忧天](http://gad.qq.com/user/index?id=1075523)

链接：<http://gad.qq.com/article/detail/41102>

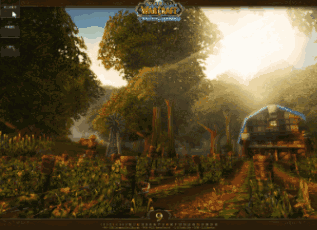
**屏幕虚化**是非常简单的屏幕后处理特效之一，它的基本做法是对图片中的像素进行**加权平均**，也就是让每个像素的颜色都或多或少的受到周围像素颜色的影响，进而模糊画面，产生撸多了的效果，网上屏幕虚化的教程很多，在此就不再赘述了。

相较传统的“**咔嚓虚化**”，“**淡入式虚化**”的差别是：

1、前戏足，过渡自然，不生硬。

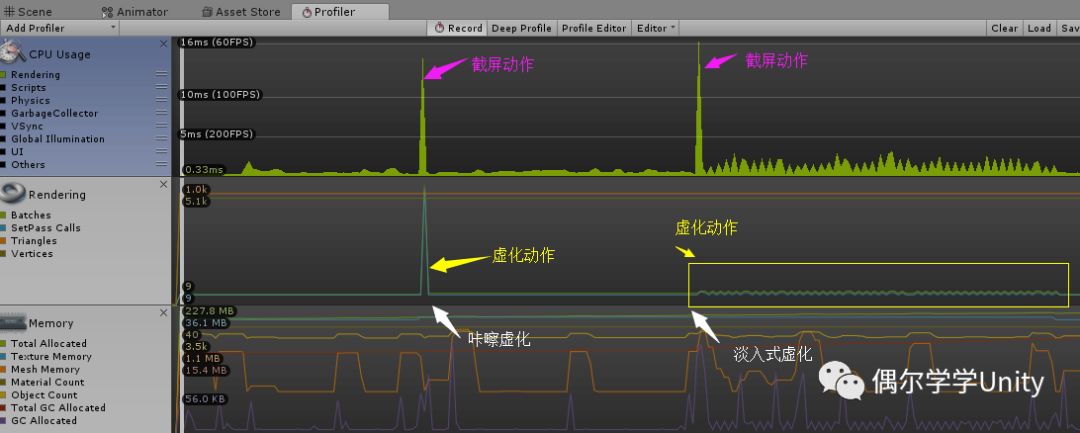


咔嚓虚化

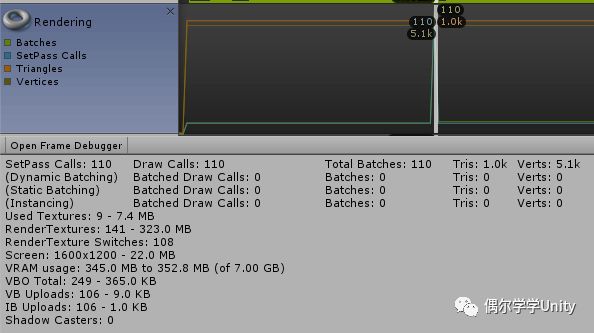


淡入式虚化

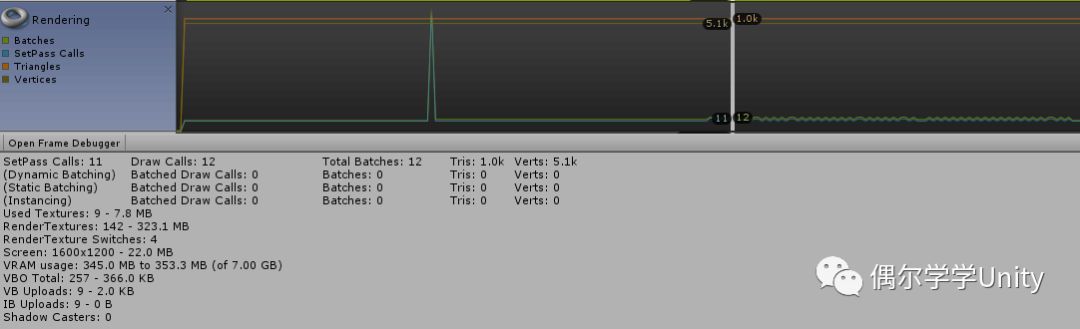
2、分担了下GPU的DrawCall，将DC的峰值稍稍降低了些。



50次迭代虚化的渲染情况



咔嚓虚化DC



淡入式虚化DC

当然，“咔嚓虚化”和“淡入式虚化”这个名字是我起的，知道意思就行，**不用较真**。

其实，“**淡入式虚化**”与“**咔嚓虚化**”本质上**没有区别**，只是把多次虚化的动作放在了**Coroutine**中，大概流程是这样的：

**OnRenderImage函数调用时，保存一张屏幕截图，并激活淡入式虚化。**

**↓**

**在Coroutine中，按照预设的频次迭代虚化屏幕截图。**

**↓**

**达到预设的虚化次数后，停止虚化并结束Coroutine。**

到这里就讲完了，实现过屏幕虚化的朋友肯定已经明白了，再见。

如果对于屏幕虚化还不甚了解，或者自己以后年纪大了忘了思路，可到后面看一下几个关键点来辅助回忆，请吧。

本文讨论的两种虚化，有以下几个**特点**：

**相同点：**

1、通过适当的缩小截屏后的图片，可在提升效率的基础上加强模糊效果。这是因为**缩小图片尺寸**等于减少需要处理的像素点，因此效率更高；且图片先缩小，丢掉部分像素信息后再拉伸回原尺寸也能起到一定模糊的效果(因为使用了**Bilinear**的过滤方式)。



截屏时减小图片尺寸的虚化结果比对（迭代10次，采样半径1.2）

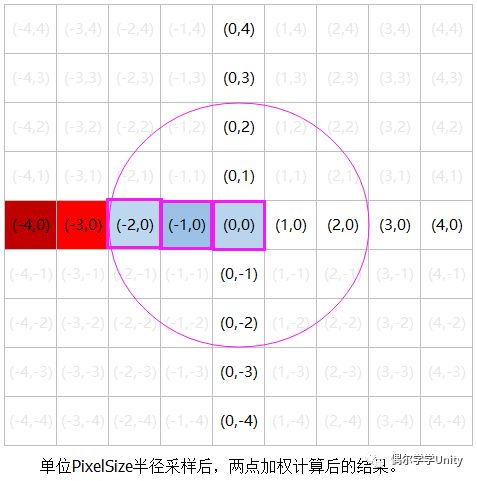
（注：这里所谓的“截屏”就是将相机渲染结果的颜色信息取出。后面的虚化处理就是对这个拷贝出来的颜色信息而做的。截屏是通过OnRenderImage函数完成的，这个函数调用的时机可参看Unity函数生命周期或上篇文章《ClearFlags 与 OnRenderImage》。）

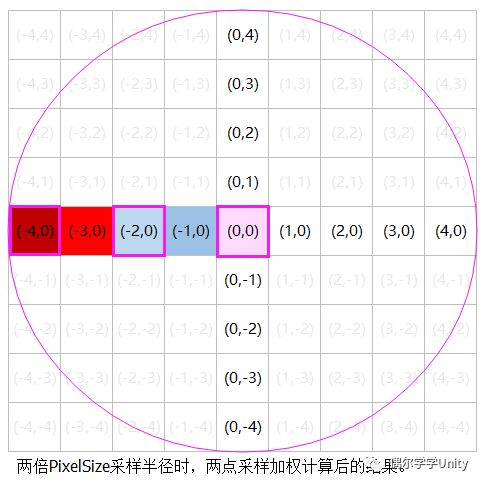
2、**多次迭代**虚化图片，即每次都在上一次虚化的基础上再次虚化，可以提升虚化品质。



整迭代次数的虚化结果比对（采样半径1.2，降采样2倍）

3、合理**调整虚化半径**，可以提升虚化品质。







调整采样半径的虚化比对（迭代20次，降采样1倍）

注：如果虚化半径调的过高，会让图片变得“瓷砖化”，正所谓过犹不及，适合自己的才是最好的。

4、**使用两个Pass通道，分横纵两次进行虚化**，减少DrawCall时的采样次数。



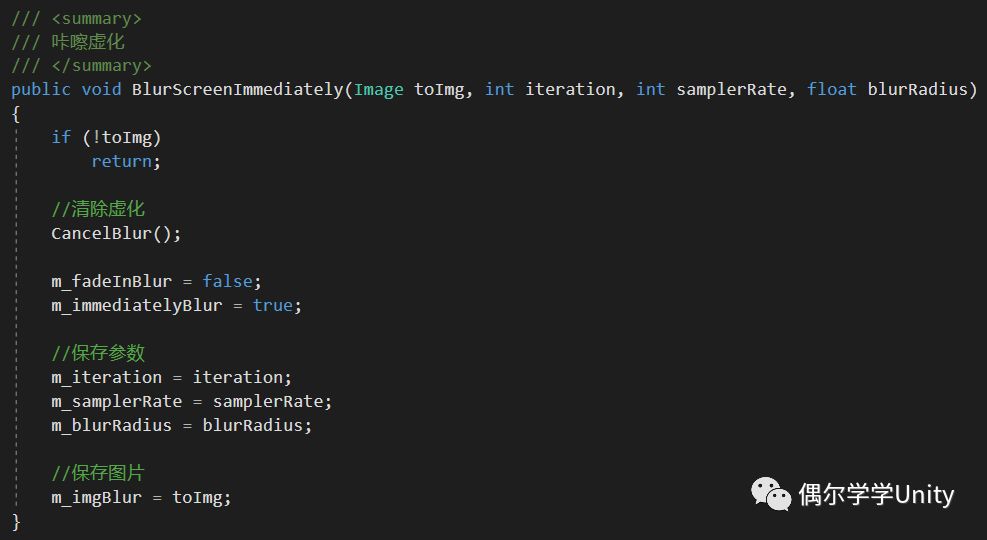


**不同点：**

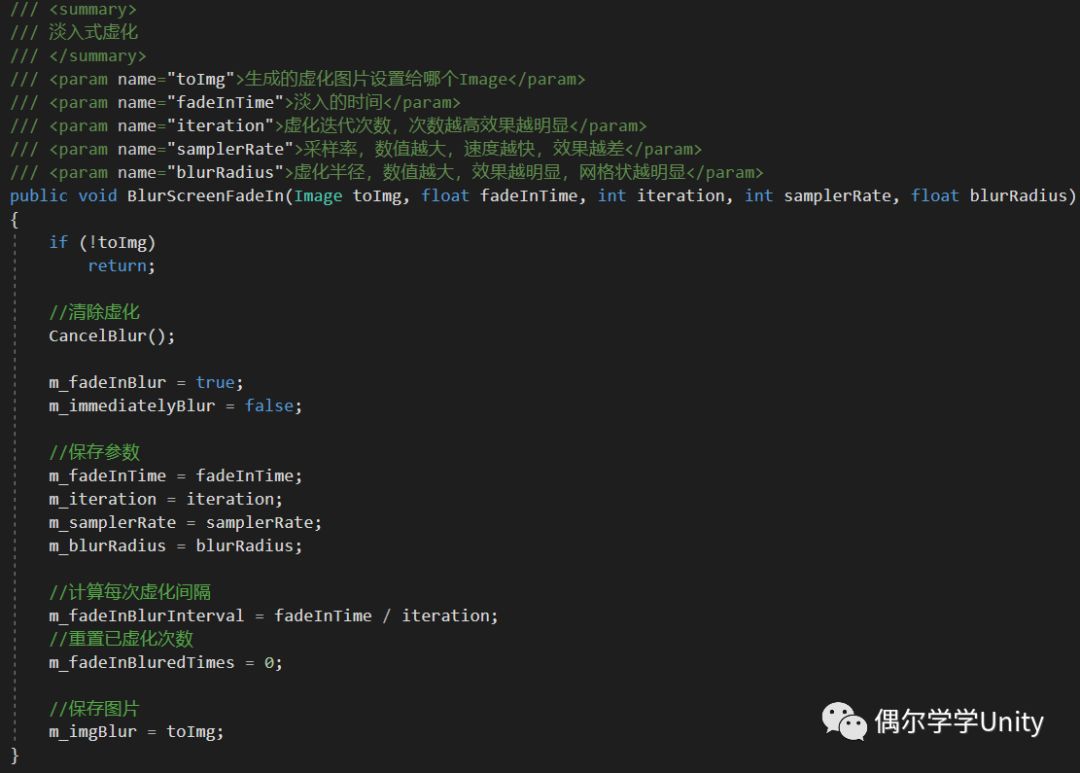
“咔嚓虚化”的虚化动作**直接放在OnRenderImage中**。因此若迭代次数较高，这一渲染帧的DrawCall必然高。

“淡入式虚化”在OnRenderImage中仅做截屏操作，虚化动作则放在Coroutine中，即根据虚化强度及时间**平均在各渲染帧中**。

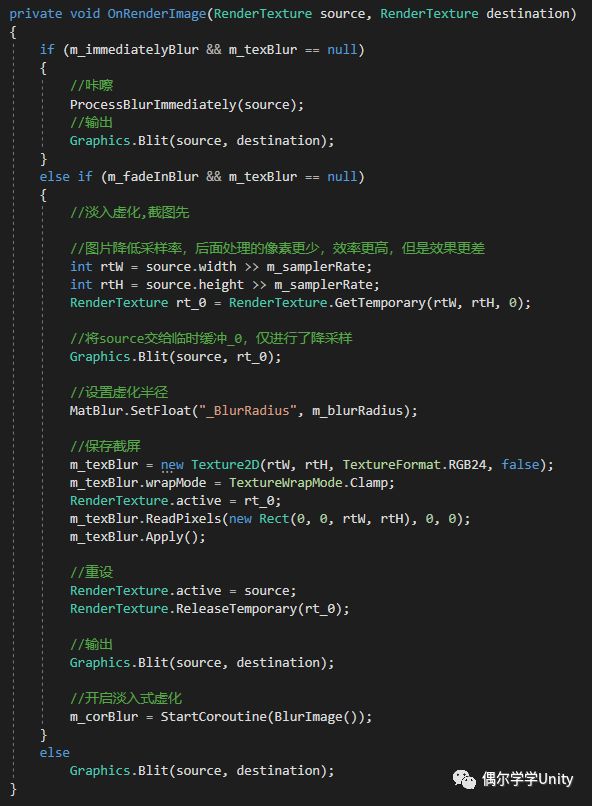
**关键部分代码：**



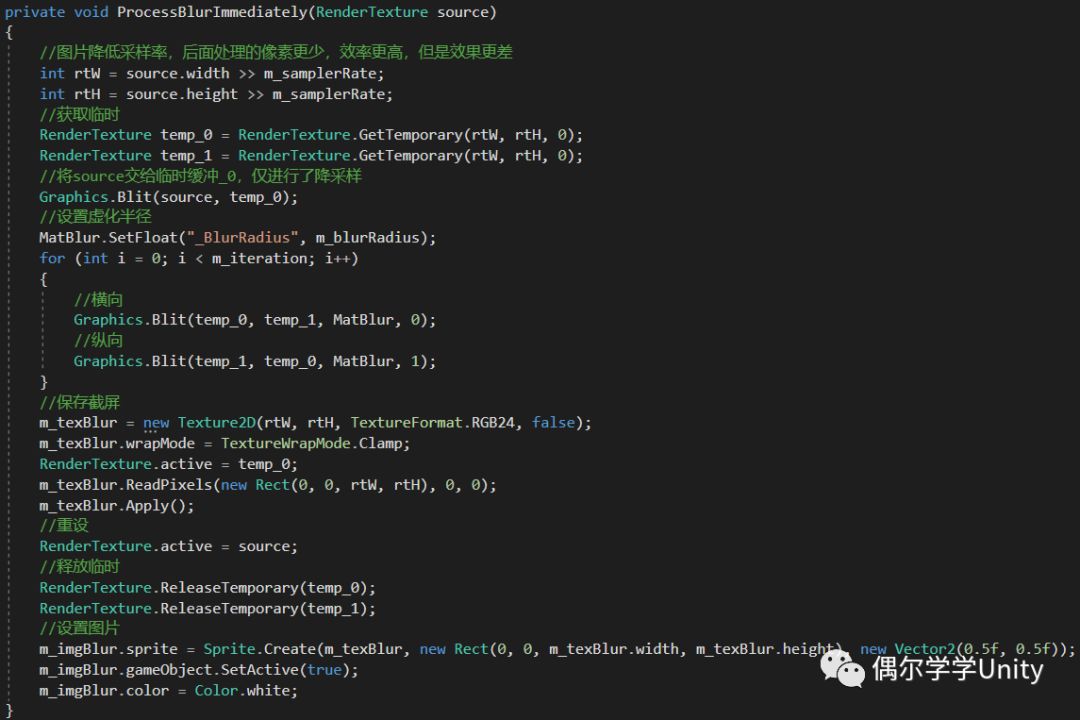
咔嚓虚化调用接口



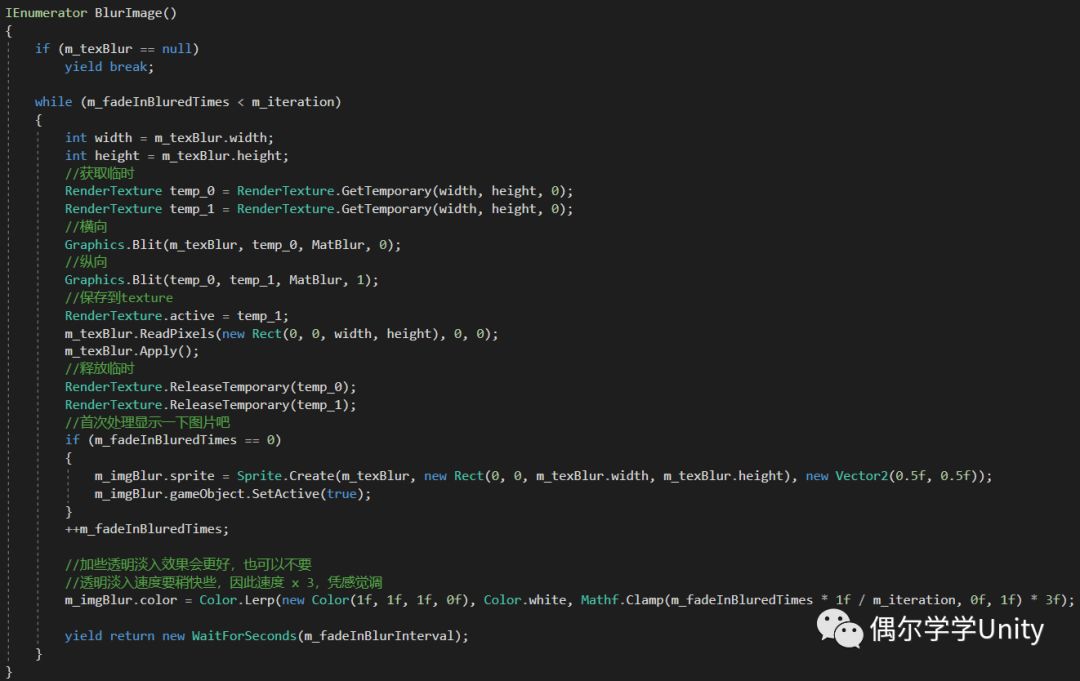
淡入式虚化调用接口



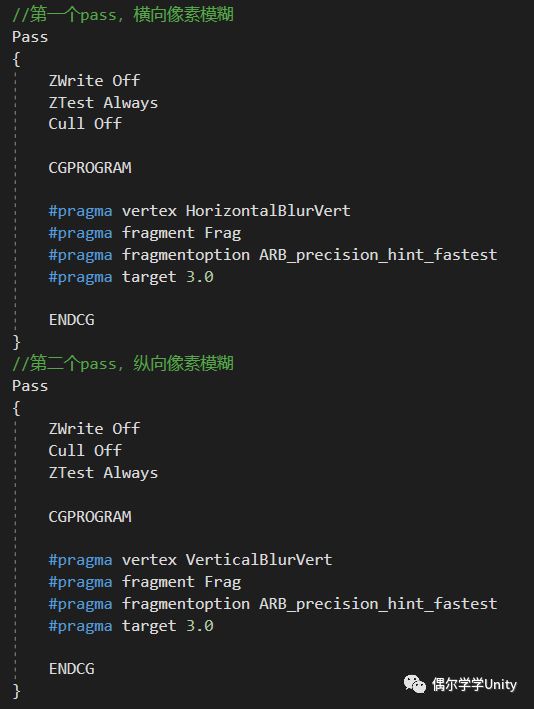
OnRenderImage操作



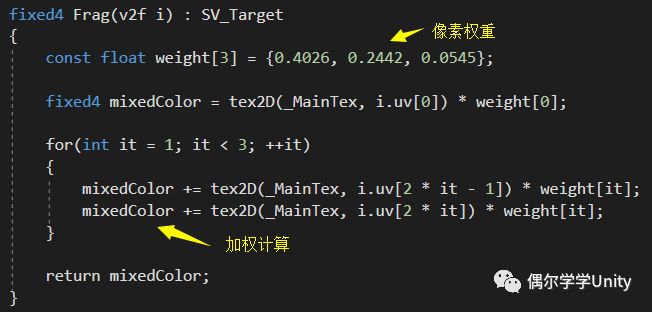
咔嚓虚化直接迭代虚化图片



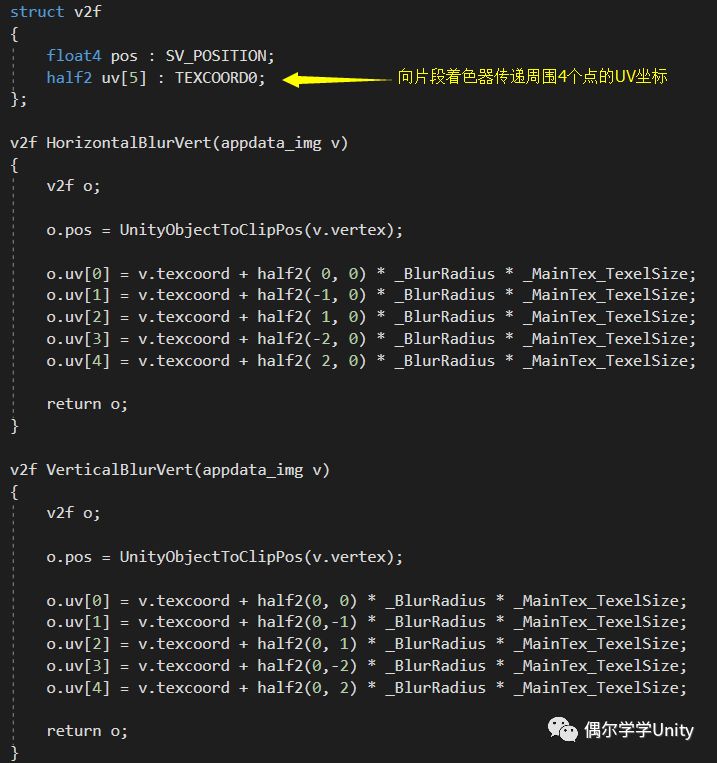
淡入式虚化分帧虚化图片



虚化Shader的两个Pass通道



仅额外采样像素周围4个点的信息，可增加。



进行像素颜色的加权计算

**最后值得注意一个地方：**

要对保存虚化图片的Texture2D进行些属性设置：

