https://www.cnblogs.com/Hichy/p/9274866.html

<https://blog.csdn.net/puppet_master/article/details/72669977>

**屏幕图像捕捉**

1）Shader的GrabPass

GrabPass可以很方便地捕获当前渲染时刻的FramBuffer中的图像，其原理就是从当前FrameBuffer中copy一份纹理，通过SetTexture的方式设置纹理，至于GrabPass的性能问题，一般认为是对FrameBuffer进行的一些pixel copy operations造成的。

GrabPass{}不带参数的默认名字为”\_GrabTexture”会为每一个使用当前shader的obj抓取一次

GrabPass{“TextureName”}不带参数的默认名字为”\_GrabTexture”会为每一个使用当前shader的obj抓取一次，带名字

**注意：并不是所有硬件都支持GrabPass**

2)CommandBuffer

使用Graphics.ExecuteCommandBuffer(commandBuffer)的方式手动抓取，也可以添加到事件队列中，在某个时刻抓取。

在Unity中，我们可以在Unity Shader中使用一种特殊的Pass来完成获取屏幕图像的目的，这就是GrabPass，当我们在shader中定义了一个GrabPass后，Unity会把当前屏幕的图像绘制在一张纹理中，以便我们在后续的Pass中访问它，我们通常会使用GrabPass来实现诸如玻璃等透明材质的模拟，与使用简单的透明混合不同，使用GrabPass可以让我们对该物体后面的图像进行更复杂的处理。

需要注意的是，在使用GrabPass的时候，我们需要额外小心物体的渲染队列设置，正如之前所说，GrabPass通常用于渲染透明物体，尽管代码里并不包含混合指令，但我们往往仍然需要把物体的渲染队列设置成透明队列（即”Queue”=”Transparent”），这样才可以保证当渲染该物体时，所有的不透明物体都已经被绘制在屏幕上，从而获取正确的屏幕图像。

Command Buffer是unity 5新增的一个非常强大的功能，我们在渲染的时候，给opengl或者dx的就是一系列指令，比如glDrawElement、glClear等等，这些东西目前是引擎调用的，而unity也为我们封装了更高一级的api，也就是command buffer，可以让我们更加方便灵活地实现一些效果。Command buffer最主要的功能时可以预定义一些渲染指令，然后将这些指令在我们想要的实际进行执行。比如说之前要做外发光效果的时候，必须要单独设立一个摄像机去做RT，这个相机的属性其实和主相机没什么太大区别，但是你要为了维护这个东西写很多代码。有了CommandBuffer就可以不必再建立额外的摄像机，CommandBuffer的好处就是让你给一个摄像机添加命令，好完成需要N个摄像机才能做到的工作。

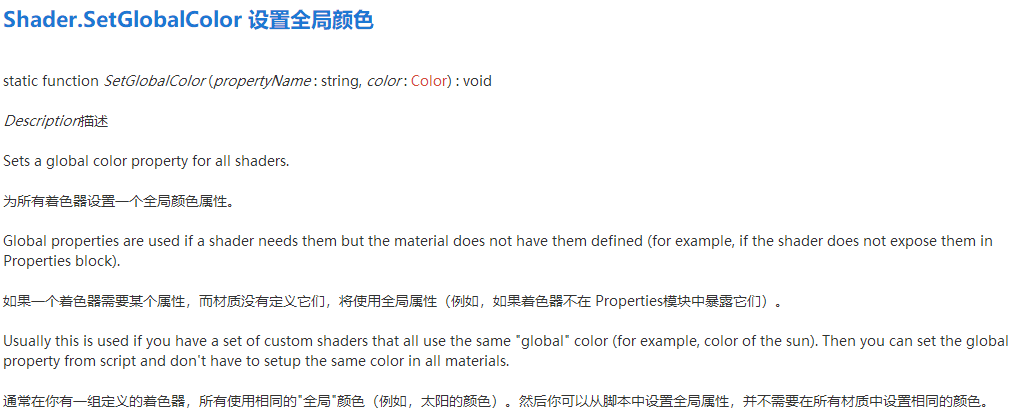
1）Unity里面的RenderTexture有两个身份看，一个是RenderToTexture类型的变量，另一个只是一个int类型的ID（identifier），很多相关的函数中都有这个identifier的变量

2）在镜头上添加Command Buffer，需要注意“镜头事件”（CameraEvent），这个东西是控制CommandBuffer执行的时机。比如在官方的例子中，CameraEvent用的是CameraEvent.AfterSkybox，指的就是在渲染天空盒之后执行CommandBuffer中的命令。根据自己的需求安排CommandBuffer的执行时机。比如在非透明物体渲染之后、透明物体渲染之后、RenderImage之后等

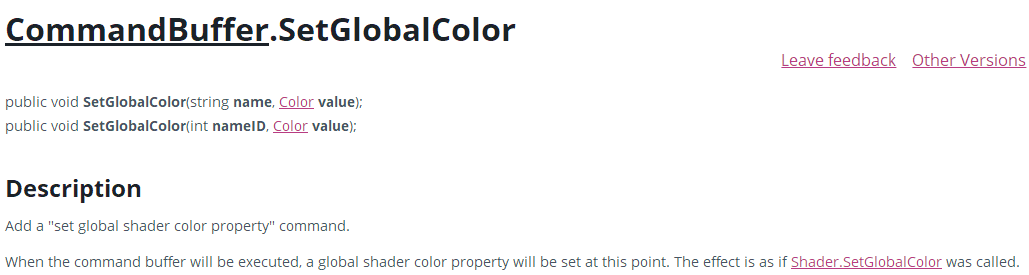
3）使用CommandBuffer做RenderTexture最重要的语句：即使声明一个RenderTexture，并且在当前“镜头事件”触发的时候把屏幕中的物体渲染到这个RenderTexture中。

首先看看SetGlobalColor方法

总共有Shader.SetGlobalColor与CommandBuffer.SetGlobalColor方法



实际上这个函数要生效，必须在shader的Properties没有定义该属性，否则不生效（已经验证）



也就是说CommandBuffer.SetGlobalColor设置的属性也必须没有在属性中设定（已经验证）

Shader.PropertyToID方法实际上是在Unity中每个属性名字都对应一个ID，这个ID在游戏过程中不会变化，我们可以在设置属性的时候通过它来进行设置，**即使是不同的shader，如果名字一样，也对应相同的ID**

// 创建一个属性

int screenCopyID = Shader.PropertyToID("\_ScreenCopyTexture");

// 添加一个命令(获取一个RenderTexture，并将它赋给全局shader属性，这个RenderTexture如果没有手动释放，

// 则会在当前相机渲染完成之后释放，或者在执行Graphics.ExecuteCommandBuffer之后释放)

buffer.GetTemporaryRT(screenCopyID, -1, -1, 0, FilterMode.Bilinear);

// 添加一个命令(设置激活的RenderTarget，将颜色写入CurrentActive，将深度写入CameraTarget)

buffer.SetRenderTarget(BuiltinRenderTextureType.CurrentActive, BuiltinRenderTextureType.CameraTarget);

// 将RenderTexture CurrentActive写入到ID为screenCopID的RenderTexture中

buffer.Blit(BuiltinRenderTextureType.CurrentActive, screenCopyID);

buffer.SetGlobalTexture("\_GrabBlurTexture", screenCopyID);

camera.AddCommandBuffer(CameraEvent.AfterForwardOpaque, buffer);

