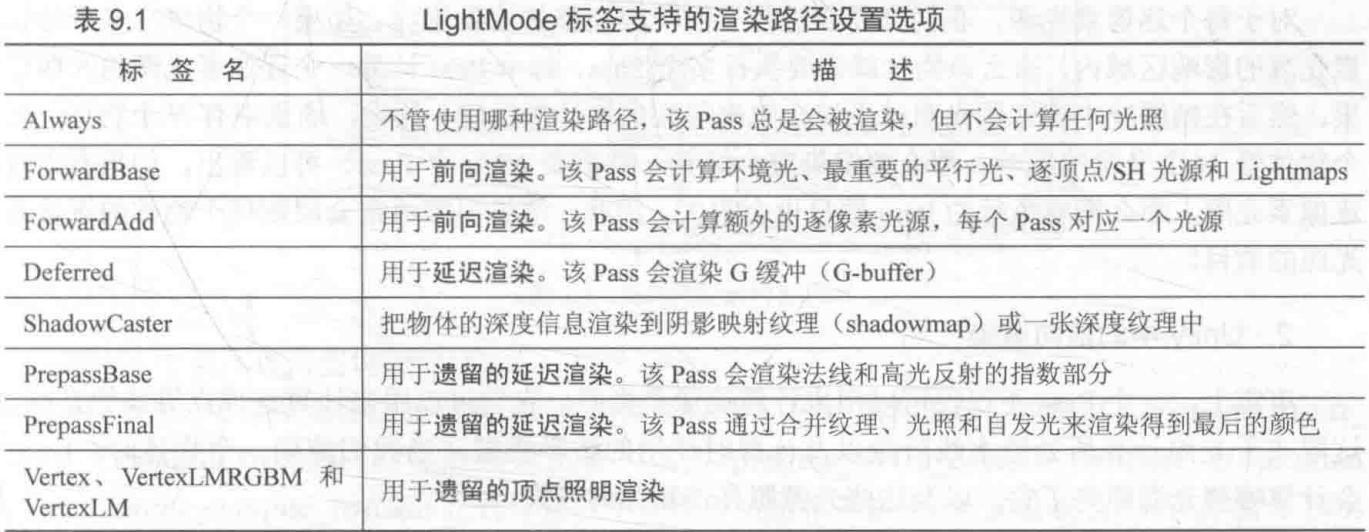
**Unity的渲染路径**

在Unity中，渲染路径决定了光照是如何应用到Unity Shader中的。因此，如果要和光源打交道，我们需要为每个Pass指定它使用的渲染路径，只有这样才能让Unity知道，“哦，原来这个程序员想要这种渲染路径，那么好的，我把光源和处理后的光照信息都放在这些数据里，你可以访问啦”，也就是说，我们只有为Shader正确的选择和设置了需要的渲染路径，该Shader的光照计算才能被正确执行。

Unity支持多种类型的渲染路径。在Unity 5.0版本以前，主要有3种：前向渲染路径（Forward Rendering Path）、延迟渲染路径（Deferred Rendering Path）和顶点照明渲染路径（Vertex Lit Rendering Path）。但在Unity 5.0以后，Unity做了很多更改，主要有两个变化：S后弦，顶点照明渲染路径已经被Unity抛弃（但目前仍然可以对之前使用了顶点照明渲染路径的Unity Shader兼容）；其次，新的延迟渲染路径代替了原来的延迟渲染路径（同样，目前也提供了对旧版本的兼容）

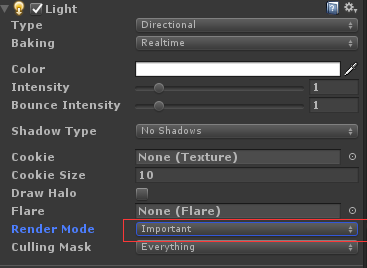
**需要注意的是，如果当前的显卡并不支持所渲染的渲染路径，Unity会自动使用更低一级的渲染路径。例如，如果一个GPU不支持延迟渲染，那么Unity就会使用前向渲染。在Unity 5.x中如果使用了前向渲染又没有为Pass指定任何前向渲染适合的标签，就会被当成一个和顶点照明渲染路径等同的Pass，那么一些光照变量很可能不会被正确赋值，我们计算出的效果也很可能有错误**



**Unity中的前向渲染**

事实上，一个Pass不仅仅可以用来计算逐像素光照，它也可以用来计算逐顶点等其他光照。这取决于光照计算所处流水线阶段以及计算时使用的数学模型。当我们渲染一个物体时，Unity会计算那些光照照亮了它，以及这些关照照亮该物体的方式。

在Unity中，前向渲染路径有3种处理光照的方式：逐顶点处理、逐像素处理、球谐喊函数处理。而决定一个光源使用哪种处理模式取决于它的类型和渲染模式。光源类型指的是该光源是平行光还是其他类型的光源，而光源的渲染模式指的是该光源是否是重要的。如果我们把一个光照的模式设置为Important，意味着我们告诉Unity，“嘿老兄，这个光源很重要，我希望你可以认真对待它，把它当成一个逐像素光源来处理”，我们可以在光源的Light组件中设置这些属性：



在前向渲染中，当我们渲染一个物体时，Unity会根据场景中各个光源的设置以及这些光源对物体的影响程度（例如，距离该物体的远近、光源强度等）对这些光源进行一个重要度排序。其中，一定数量的光源会按逐像素的方式处理，然后由最多4个光源按逐顶点的方式处理，剩下的光源可以按照SH方式处理。Unity使用的判断规则如下：

1）场景中最亮的平行光总是按逐像素处理的

2）渲染模式被设置成Not Important的光源会按照逐顶点或者SH处理

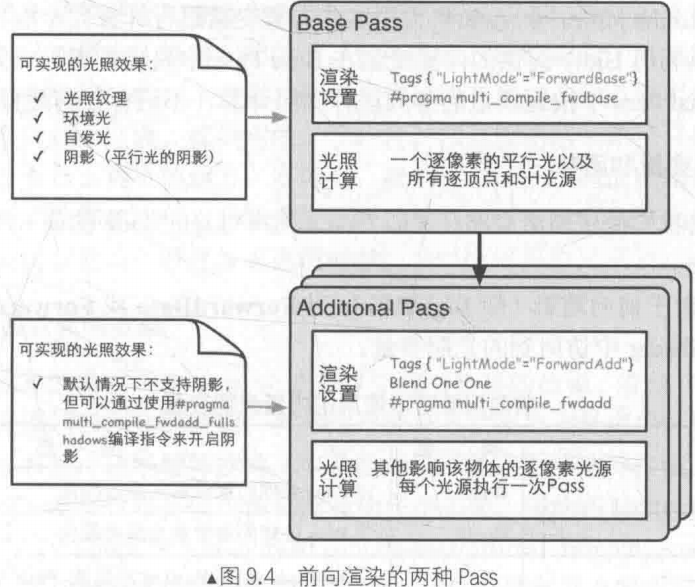
3）渲染模式被设置成Important的光源，会按逐像素处理

4）如果根据以上规则得到的逐像素光源数量小于Quality Setting中的逐像素光源数量（Pixel Light Count），会有更多的光源以逐像素的方式进行渲染

在Quality Setting中有：



那么，在哪里进行光照计算？当然是在pass里，前面提到过，前向渲染有两种Pass：Base Pass和Additional Pass。通常来说，这两种Pass进行的标签和渲染设置以及常规光照计算如下图所示：



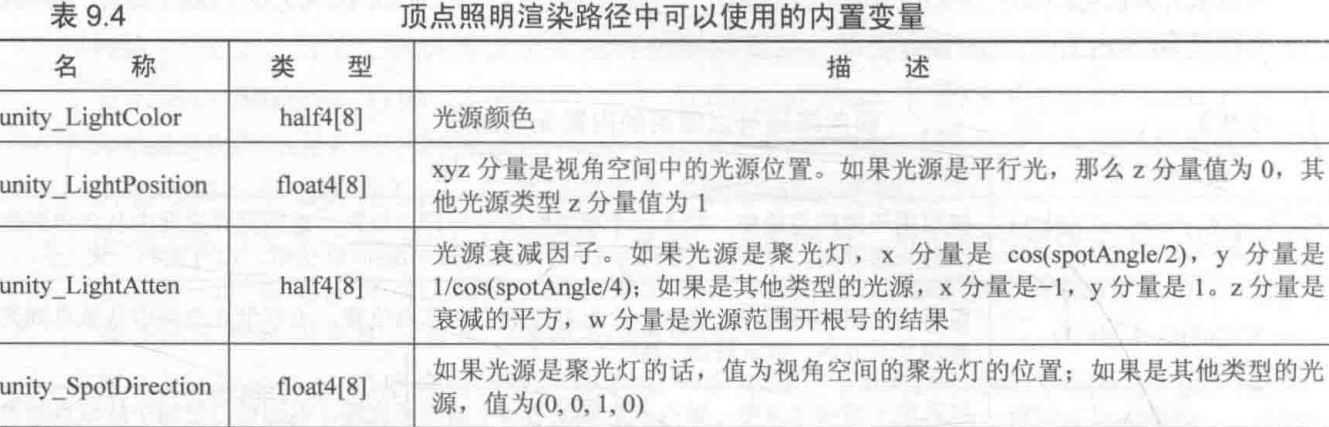
对于上图有几点需要说明的地方：

1. 首先可以发现在渲染设置中，我们除了设置了Pass的标签外，还使用了#pragma multi\_compile\_fwdbase这样的编译命令。虽然#pragma multi\_compile\_fwdbase和#pragma multi\_compile\_fwdadd在官方文档中还没有给出相关说明，但实验表明，只有分别为Base Pass和Additional Pass使用这两个编译指令，我们才可以在相关的Pass中得到一些正确的光照变量，例如光照衰减值
2. Base Pass旁边的注释给出了Base Pass中支持的一些光照特性。例如，在Base Pass中，我们可以访问光照纹理。
3. Base Pass中渲染的平行光默认是支持阴影的（如果开启了光源的阴影功能），而Additional Pass中渲染的光源在默认情况下是没有阴影效果的，即便我们在它的Light组件在宏设置了有阴影的Shadow Type。但我们可以在Additional Pass中使用#pragma multi\_compile\_fwdadd\_fullshadows代替#pragma multi\_compile\_fwdadd编译指令，为点光源和聚光灯开启阴影效果，但这需要Unity在内部使用更多的Shader变种
4. 环境光和自发光也是在Base Pass中计算的。这是因为，对于一个物体来说，环境光和自发光我们只希望计算一次即可，而如果我们在Additional Pass中计算着两种光照，就会造成叠加多次环境光和自发光，这不是我们想要的
5. 在Additional Pass的渲染设置中，我们还开启和设置了混合模式。这是因为，我们希望每个Additional Pass可以与上一次的光照结果在帧缓冲中进行叠加，从而得到最终的有多个光照的渲染效果。如果我们没有开启和设置混合模式，那么Additional Pass的渲染结果会覆盖掉之前的渲染结果，看起来就好像改物体只受到该光源的影响。通常情况下我们选择的混合模式是Blend One One
6. 对于前向渲染来说，一个Unity Shader通常会定义一个Base Pass（Base Pass也可以定义多次，例如需要双面渲染等情况）以及一个Additional Pass。一个Base Pass仅会执行一次（定义了多个Base Pass的情况除外），而一个Additional Pass会根据影响该物体的其它逐像素光源的数目被多次调用，即每个逐响度光源会执行一次Additional Pass。

**Unity中的顶点照明渲染路径**

顶点照明渲染路径是对硬件配置要求最少、运算性能最高，但同时也是得到的效果最差的一种类型，它不支持哪些逐像素才能得到的效果，例如阴影、法线映射、高精度的高光反射等。实际上，它仅仅是前向渲染路径的一个子集，也就是说，所有可以在顶点照明渲染路径中实现的功能都可以在前向渲染路径中完成。就如它的名字一样，顶点照明渲染路径只是使用了逐顶点的方式来计算光照，并没有什么神奇的地方。实际上，我们在上面的前向渲染路径中也可以计算一些逐顶点的光源。但如果选择使用顶点照明渲染路径，那么Unity只会填充那些逐顶点相关的光源变量，意味着我们不可以使用一些逐像素光照变量。

在Unity中，我们可以在一个顶点照明Pass中最多访问到8个逐顶点光源，如果我们只需要渲染其中两个光源对物体的照明，可以仅仅使用下表内置光照数据的前两个。如果影响该物体的光源数量小于8，那么数组中剩下的光源颜色会设置成黑色。



**Unity中的延迟渲染路径**

除了前向渲染中使用的颜色缓冲和深度缓冲外，延迟渲染还会利用额外的缓冲区，这些缓冲区被统称为G缓冲，G是应为geometry的缩写。G缓冲区存储了我们所关心的表面（通常指的是离摄像机最近的表面）的其他信息，例如该表面的法线、位置、用于光照计算的材质属性等。

延迟渲染主要包含了两个Pass。在第一个Pass中，我们不进行任何光照计算，而是仅仅计算那些片元是可见的，这主要是通过深度缓冲技术来实现，当法线一个片元是可见的，我们就把它的相关信息存储到G缓冲区中。然后，在第二个Pass中，我们利用G缓冲区的各个片元信息，例如表面法线、视角方向、满发射系数等，仅仅真正的光照计算。我们可以看出，延迟渲染使用的Pass数目通常就是两个，这跟场景中包含的光源数目是没有关系的。换句话说，延迟渲染的效率不依赖于场景的复杂度，而是和我们使用的屏幕空间的大小有关。这是因为，我们需要的信息都存储在缓冲区中，而这些缓冲区可以理解成是一张张2D图像，我们的计算实际上就是在这些图像空间中进行的。

延迟渲染有一些缺点：

1. 不支持真正的抗锯齿功能
2. 不能处理半透明物体
3. 对显卡有一定要求。如要要使用延迟渲染的话，显卡必须支持MRT（Multiple Render Targets）、Shader Mode 3.0以上、深度渲染纹理以及双面的模板缓冲

默认的G缓冲区（注意，不同Unity版本的渲染纹理存储内容会有所不同）包含了一下几个渲染纹理（Render Texture，RT）

1. RT0：格式是ARGB32，RGB通道用于存储漫反射颜色，A通道没有被使用
2. RT1：格式是ARGB32，RGB通道用于存储高光反射颜色，A通道用于存储高光发射的指数部分
3. RT2：格式是ARGB2101010，RGB通道用于存储法线，A通道没有被使用
4. RT3：格式是ARGB32（非HDR）或ARGBHalf（HDR），用于存储自发光+lightmao+反射探针（reflection probes）
5. 深度缓冲和模板缓冲