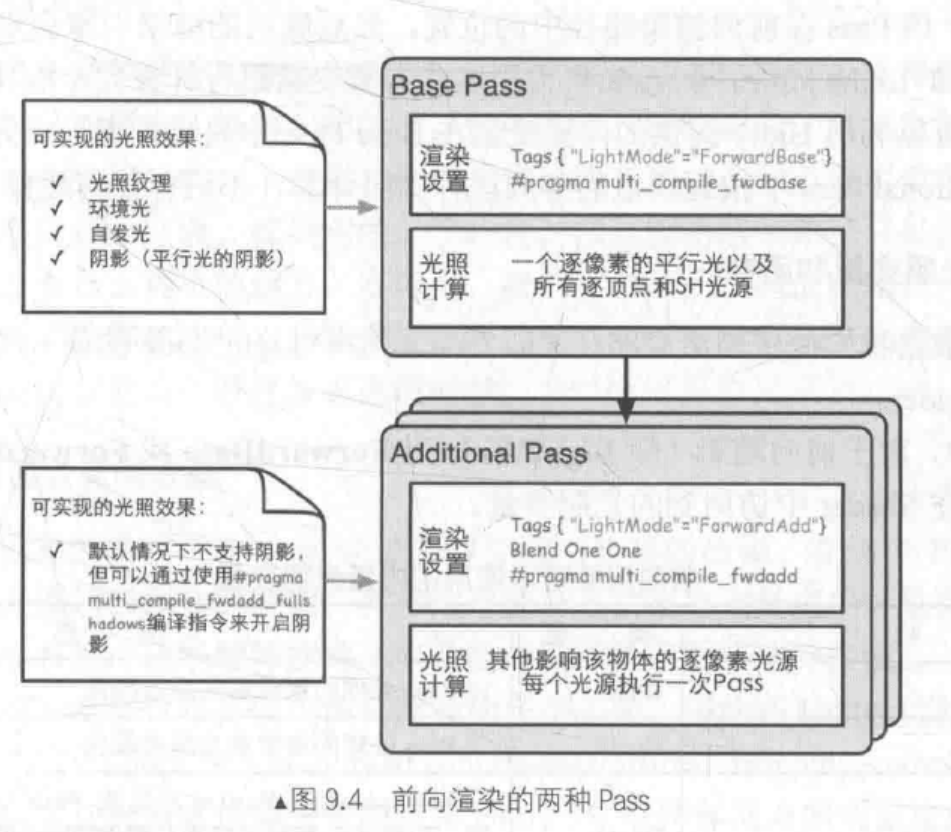
第九章 更复杂的光照 笔记

1. Lightmode标签所支持的渲染路径设置选项

|  |  |
| --- | --- |
| 标签名 | 描述 |
| Always | 不管使用哪种渲染路径，该Pass总被渲染，但不计算任何光照 |
| ForwardBase | 用于前向渲染。该Pass会计算环境光、最重要的平行光、逐顶点/SH光源和lightmaps |
| ForwardAdd | 用于前向渲染。该Pass会计算额外的逐像素光源、每个Pass对应一个光源 |
| Deferred | 用于延迟渲染。该Pass会渲染G缓冲（G-buffer） |
| ShadowCaster | 把物体的深度信息渲染到阴影映射纹理（shadowmap）或一张深度纹理中 |
| PrepassBase | 用于遗留的延迟渲染。该Pass会渲染法线和高光反射的指数部分 |
| PrepassFinal | 用于遗留的延迟渲染。该Pass通过合并纹理、光照和自发光来渲染得到最后的 |
| Vertex、VertexLMRGBM  和VertexLM | 用于遗留的顶点照明渲染 |

1. 通俗来讲，指定渲染路径，Unity会把对应渲染路径的光照属性准备好
2. Unity 5.x版本中，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
3. 前向渲染路径
   1. 如果一个物体在多个逐像素光源的影响区域内，那么该物体就需要执行多个Pass，每个Pass计算一个逐像素光源的光照结果，然后在帧缓冲中把这些光照结果混合起来得到最终的颜色
   2. 假设场景中有N个物体，每个物体受M个逐像素光源影响，那么渲染整个场景需要N\*M个Pass。
   3. 可以看出，如果有大量逐像素光源，需要执行的Pass数目会很大，因此，渲染引擎通常会限制逐像素光源的数目
   4. Unity中前向渲染路径有3种处理光照的方式：逐像素处理、逐顶点处理、球谐函数（Spherical Harmonics，SH）处理
   5. 决定一个光源使用哪种处理模式取决于它的类型（平行光/点光源等）和渲染模式（是否重要Important）
   6. 一定数目的光源会按照逐像素光源处理，然后最多有4个光源按逐顶点的方式处理，剩下的光源按SH方式处理。
   7. Unity光源处理模式判断规则如下：
      1. 场景中最亮的平行光总是按逐像素处理
      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，会按照逐像素处理
      4. 如果根据以上规则得到的逐像素光源数量少于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
   8. 
   9. 除了设置Pass的标签外，还使用了#pragma\_multi\_compile\_fwdbase和#pragma\_multi\_compile\_fwdadd这样的编译指令。实验表明，只有分别为Base Pass和Add Pase使用这两个编译指令，我们才可以在相关Pass中得到一些正确的光照变量，例如光照衰减
   10. Base Pass中可以访问光照纹理；可以计算环境光和自发光，因为一个物体的环境光和自发光我们只希望计算一次即可，如果在Additional Pass中计算这两种光照，就会造成叠加多次环境光和自发光，这不是我们想要的
   11. Base Pass中渲染的平行光默认支持阴影，而Additional Pass中渲染的光源默认没有阴影效果，即便我们在Light组件设置了有阴影的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，但这需要Unity在内部使用更多shader变种
   12. 对于前向渲染来说，一个Unity Shader通常会定义一个Base Pass（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（定义多个Base Pass的情况除外），而一个Additional Pass会根据影响该物体的逐像素光源的数目被调用多次，即其他每个逐像素光源会执行一次Additional Pass
   13. 在additional Pass中，我们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。如果没有开启混合模式，那么Additional Pass的渲染结果会覆盖掉之前的渲染结果，看起来就好像该物体只受该光源的影响。通常情况下，我们选择的混合模式是Blend One One。
   14. 上图（9.4）给出的光照计算是通常情况下我们在每个Pass中进行的计算。实际上，渲染路径的设置用于告诉unity该Pass在前向渲染路径中的位置，底层渲染引擎会进行相关计算并填充一些内置变量（比如\_LightColor0等），如何使用这些内置变量进行计算完全取决于开发者的选择。例如，我们完全可以利用Unity提供的内置变量在Base Pass中只进行逐顶光照；同样，我们也完全可以在Additional Pass中按逐顶点的方式进行光照计算，不进行任何逐像素光照计算。
   15. 
   16. 
4. 延迟渲染路径
   1. 原理：延迟渲染主要包含两个Pass。第一个Pass中，不进行任何光照计算，仅仅计算哪些片元是可见的，这主要通过深度缓冲技术实现，当发现一个片元可见，就把它的相关信息存储到G-buffer中；第二个Pass中，利用G-buffer的各个片元信息，例如表面法线、视角方向、漫反射系数等，进行真正的光照计算。
   2. 缺点：
      1. 不支持真正的抗锯齿（anti-aliasing）功能
      2. 不能处理半透明物体
      3. 对显卡有一定要求。显卡必须支持MRT（Multiple Render Targets）、Shader Mode 3.0以上、深度渲染纹理、双面的模板缓冲。
   3. 默认的G缓冲区（不同Unity版本的渲染纹理存储内容有所不同）包含了以下几个渲染纹理（Render Texture，RT）。
      1. RT0：ARGB32格式，RGB通道用于存储漫反射颜色，A通道没有被使用。
      2. RT1：ARGB32格式，RGB通道用于储存高光反射颜色，A通道用于存储高光反射的指数部分。
      3. RT2：ARGB2101010格式，RGB通道用于存储法线，A通道没有使用。
      4. RT3：ARGB32（非HDR）格式或ARGBHalf（HDR），用于存储自发光 + lightmap + 反射探针（reflection probes）。
      5. 深度缓冲、模板缓冲。
   4. 当在第二个Pass中计算光照时，默认情况下仅可以使用Unity内置的Standard光照模型。如果我们想要使用其他的光照模型，就需要替换掉原有的Internal-DeferredShading.shader文件。
   5. 