UnityRenderingPipeline

着色器(shaders)定义了物体本身的样子(材质属性)，也定义了物体是如何应对光照的。
(本文主要讲Unity Surface Shaders背后 的光照和渲染管线)

渲染路径(Rendering Paths)

Rendering path决定了光照是如何完成的，哪些Passes被使用。Shader的pass通过Pass Tags来说明光照类型。

> *Shader的一个Pass块会引起一次对应GameObject的几何结构(一般是mesh)的渲染。也就是一个pass对应一次渲染。*

Forward Rendering path(前向渲染路径)

Forward Rendering path可能用一个或多个pass渲染每个物体，这由影响该物体的光源决定。

在前向渲染中，在影响物体的光中，一些最亮的光使用逐像素的方式渲染。可以有至多4个点光源，以逐顶点的方式计算。别的光使用 Spherical Harmonics lighting的方式计算(SH Lighting比较快，但只是一种近似)。

一个光(light)是否为逐像素光，取决于:

- 渲染模式(Render Mode)被设置为(**Not Important**)的光总是为逐顶点(per-vertex)光或者SH光。
- 最亮的平行光(或者方向光，directional light)总是逐像素光(per-pixel)。
- 渲染模式(Render Mode)被设置为(**Important**)的光总是为逐像素光。
- 若经过上面三步后，已确定的逐像素光数量仍小Quality Setting中设置的**Pixel Light Count**，则更多的光可以使用per-pixel方式渲染，根据光的亮度递减顺序决定。

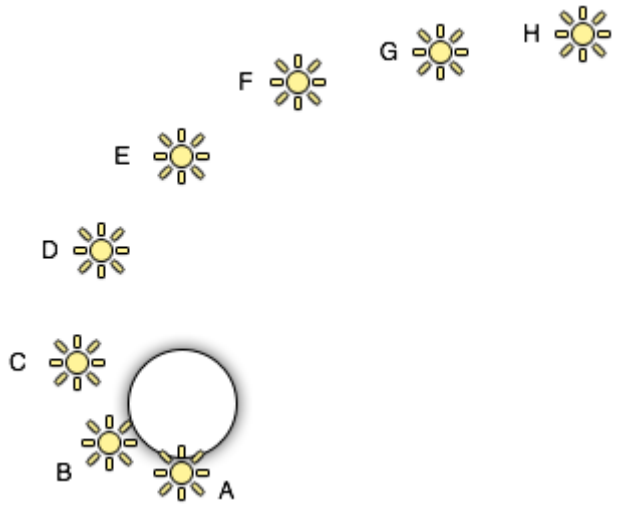
每个物体按照如下方式渲染：

BasePass

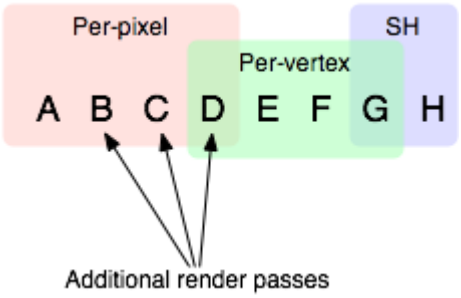
Base Pass中按per-pixel方式计算一个平行光(directional light)，这个平行光可以有阴影，和所有逐顶点光和SH光。若pass flag中设置了“OnlyDirectional”，则只计算一个平行光。这个Pass中可以使用lightmaps、环境光、自发光(emissive lighting)等。

Additional Passes

除最亮的平行光外，别的每一个per-pixel光分别用一个pass计算，这些passes被称为additional passes。这个Pass中的per-pixel光默认是没有阴影的，除非使用multi_compile_fwdadd_fullshadows 。



举例：如上图，有一个物体(用图中的圆表示)，它受到光A-H影响。假设光A-H有相同的颜色和强度，渲染模式都为Auto,则最亮的光A-D使用per-pixel模式渲染，最多4个顶光源D-G使用per-vertex渲染，最终剩余的光(G-H)使用SH渲染，如下图，注意这几个光组是有重叠的(怎么实现的？)。



Deferred shading rendering path(延迟渲染路径)

在三维计算机图形学领域，deferred shading, 延迟渲染，是一种屏幕空间shading技术，最早由 Michael Deering 于1988年提出。之所以称之为“延迟”(deferred)是因为在它的第一个pass中并没有渲染(shading)发生，而是推迟到第二个pass才进行渲染(shading)。
deferred shader的第一个pass中，只是计算了一些渲染需要的一些数据，如：位置、法线、材质。这些数据被渲染到几何缓存G-buffer(geometry buffer)中。然后，在pixel shader(也即fragment shader)中利用第一个pass中得到的G-buffer中的信息计算直接或间接光照信息。

延迟渲染的主要优势是从光照(lighting)中解耦了几何(geometry)信息。只需要一个pass来计算几何信息。每一个光(light)只对它影响到的像素进行计算。因此可以在场景中添加大量光而不会造成显著的性能影响。

延迟渲染的一个主要缺点是不能处理透明(transparency)(开启Z-buffer的场景都不能)。另一个严重的问题是不能使用多种材质(multiple materials)，因为大量不同材质要求更多的数据存储在G-buffer中(G-buffer中本来就有大量数据，占用大量存储空间了)。还有一个重要的问题是硬件反走样(hardware anti-aliasing)不能产生正确的结果，因为光照计算阶段(lighting stage)和几何计算阶段(geometric stage)是分开的(在不同Pass中完成)，采样插值会得到错误的位置、法线、切线等信息。

Overview

在Unity中，当使用延迟渲染时，影响一个GameObject的光(light)的数量没有限制。并且所有的光是按per-pixel方式计算的，可以使用法线贴图(normal map)。另外，所有的光可以有cookies和阴影。

优点

由上文已知，在deferred shading中，光照的处理开销和光所影响的像素的数量成正比，而和光点亮的GameObjects的数量无关。因此，可以通过尽量减小光照的区域来提升性能。Deferred shading具有高度一致且可预测的行为。由于每一个光都是按照per-pixel方式计算的，在大的三角形面片上不会有光照计算瑕疵。

缺点

和上文提到的一样，deferred shading不能真正地支持反走样，不能处理半透明物体(这些是在forward rendering中处理的)。另外，在deferred shading中Mesh Renderer也不可以使用Receive Shadows 标志，且culling masks也会受限制，最多可以使用4个culling masks。

使用要求

要求显卡支持多个渲染目标(Multiple Render Targets, MRT)，Shader Model 3.0以上，且支持深度渲染纹理(Depth render textures)。大部分在2006年以后生产的PC上的显卡(以GeForce 8xxx, Radeon X2400, Intel G45等开头)支持deferred shading。

运行OpenGL ES3.0以上的所有移动平台支持deferred shading。

注意： Deferred rendering不支持正交投影(Orthographic projection)。若相机投影模式设置为Orthographic，则相机会使用Forward rendering。

性能注意事项

由于在延迟渲染中光的渲染开销和它“点亮的”像素个数成正比，和场景复杂度无关。因些小的点光源或聚光灯(spot lights)是非常廉价的，如果它们被别的物体遮挡的话，性价比就更高了。

当然，支持阴影的光会比不支持阴影的光开销大很多。

实现细节

具有不支持deferred shading的shaders的物体在deferred shading完成后，按照forward rendering方式渲染。

G-buffer中渲染目标(render targets, RT0-RT4)的默认布局如下。数据类型放在每个render target的各个通道中。通道放在括号中：

- RT0, ARGB32 格式: Diffuse color(RGB), occlusion(A).
- RT1, ARGB32 格式: Specular color(RGB), roughness(A).
- RT2, ARGB2101010 格式: World space normal(RGB), unused(A).

- RT3, ARGB2101010 (non-HDR)或者ARGBHalf(HDR)格式: Emission+lighting+**lightmaps**+**reflection probes** buffer.
- Depth+**Stencil buffer**.

因此g-buffer默认布局: 160 bits/pixel (non-HDR) 或者 192 bits/pixel(HDR).

若使用Shadowmask 或者 Distance Shadowmask模式用于混合光照(Mixed lighting)，将会用到第5个渲染目标:

- RT4, ARGB32 格式: Light occlusion values(RGBA).

因此，g-buffer布局为: 192 bits/pixel (non-HDR) 或者 224 bits/pixel (HDR).

若硬件不支持五个并行渲染目标，则使用shadowmasks的对象将使用forward rendering path. 当相机未使用HDR时，Emission+lighting buffer (RT3) 将被对数编码以提供更大的动态范围(相比于通常使用ARGB32纹理)。

注意当相机使用HDR rendering时，不会有独立于RT3的渲染目标被创建；相反，使用的渲染目标是RT3。

G-Buffer pass

G-buffer pass只渲染每个对象一次。Diffuse colors, specular colors, surface smoothness, world space normal和emission+ambient+reflections+lightmaps被渲染进g-buffer 纹理。g-buffer纹理被设置为全局的shader属性，以便稍后由shaders来访问(CameraGBufferTexture0 .. CameraGBufferTexture3)。

Lighting pass

Lighting pass根据 g-buffer和depth计算光照。光照是在屏幕空间计算的，因此其处理时间不依赖于场景复杂度。光照被添加到emission buffer。

唯一可以使用的光照模型(lighting model)是Standard。若想使用一个不同的模型，可以修改lighting pass shader：把修改版的Internal-DeferredShading.shader(来自Built-in shaders)放到项目的"Assets/Resources"目录下，然后打开 **Edit>Project Settings**，点击Graphics category，修改"Deferred"下拉列表为"Custom Shader"，最后把Shader选项设为自己定义的shader。
参考文献：

1. Unity’s Rendering Pipeline
2. ShaderLab: Pass
3. https://en.wikipedia.org/wiki/Deferred_shading
4. Spherical Harmonics lighting
5. Deferred shading rendering path