[**http://docs.unity3d.com/540/Documentation/Manual/RenderTech-DeferredShading.html**](http://docs.unity3d.com/540/Documentation/Manual/RenderTech-DeferredShading.html)

**Deferred shading rendering path**

**延迟着色的渲染路径**

This page details the deferred shading [rendering path](http://docs.unity3d.com/540/Documentation/Manual/RenderingPaths.html). See [Wikipedia: deferred shading](http://en.wikipedia.org/wiki/Deferred_shading) for an introductory technical overview.

这个界面详细说明了延迟着色渲染路径。参见[Wikipedia: deferred shading](http://en.wikipedia.org/wiki/Deferred_shading)  做一个技术概览

**Overview**

**概述**

When using deferred shading, there is no limit on the number of lights that can affect a GameObject. All lights are evaluated per-pixel, which means that they all interact correctly with normal maps, etc. Additionally, all lights can have cookies and shadows.

当使用了延迟着色，能影响一个游戏物体灯的数量没有限制。所有的光会计算每一个像素，意味着他们都会相互影响法线贴图等。另外，所有的光可以有cookies和阴影。

Deferred shading has the advantage that the processing overhead of lighting is proportional to the number of pixels the light shines on. This is determined by the size of the light volume in the Scene regardless of how many GameObjects it illuminates. Therefore, performance can be improved by keeping lights small. Deferred shading also has highly consistent and predictable behaviour. The effect of each light is computed per-pixel, so there are no lighting computations that break down on large triangles.

延迟着色具有的优点是，光照的像素数量和照明处理的开销是成正比的。这是通过场景光量的多少决定，而不是照亮多少GameObject确定。因此，可以改变光的大小可以提高性能。延迟着色有始终如一和可预测的行为。每个光的效果是逐像素单位计算的。因此不会在大规模三角形情况下，光照计算发生性能崩溃的情况。

On the downside, deferred shading has no real support for anti-aliasing and can’t handle semi-transparent GameObjects (these are rendered using [forward](http://docs.unity3d.com/540/Documentation/Manual/RenderTech-ForwardRendering.html) rendering). There is also no support for the Mesh Renderer’s Receive Shadows flag and culling masks are only supported in a limited way. You can only use up to four culling masks. That is, your culling layer mask must at least contain all layers minus four arbitrary layers, so 28 of the 32 layers must be set. Otherwise you get graphical artefacts.

缺点是，延迟着色没有真正支持抗锯齿，并且不能处理半透明物体（他们使用前向渲染）。也不支持网格渲染器接受阴影的标志，裁剪mask只能有一种限制方式。你最多可以使用四个的裁剪蒙版。也就是说你的裁剪蒙版层包含在所有层减掉任意四层，因此32层有28层要设置。否则你会得到图形文物。

**Requirements**

**要求**

It requires a graphics card with Multiple Render Targets (MRT), Shader Model 3.0 (or later) and support for Depth render textures. Most PC graphics cards made after 2006 support deferred shading, starting with GeForce 8xxx, Radeon X2400, Intel G45.

要求显卡支持多重渲染目标，Shader Model3.0（或者更高）和支持深度渲染图。大多个人电脑在2006年后的支持延迟渲染，从GeForce 8xxx, Radeon X2400, Intel G45开始。

On mobile, deferred shading is not supported, mostly due to MRT formats used (some GPUs which do support multiple render targets, still only support very limited bit counts).

在手机上，延迟渲染是不支持的。主要是由于多重渲染目标使用的格式（一些手机支持多渲染目标，还是支持非常有限的比特数）。

Note: Deferred rendering is not supported when using Orthographic projection. If the Camera’s projection mode is set to Orthographic, the Camera falls back to Forward rendering.

注解：延迟渲染不支持正交投影。如果相机的模式是正交，相机会回落到正向渲染。

**Performance considerations**

**性能注意事项**

The rendering overhead of realtime lights in deferred shading is proportional to the number of pixels illuminated by the light and not dependent on Scene complexity. So small point or spot lights are very cheap to render and if they are fully or partially occluded by Scene GameObjects then they are even cheaper.

实时光在延迟着色中的消耗，正比于光照的像素数量而不依赖于场景的复杂度。所以对于渲染来说小的点光或者聚光灯消耗很少，如果他们被场景的物体遮蔽，那么消耗会更小。

Of course, lights with shadows are much more expensive than lights without shadows. In deferred shading, shadow-casting GameObjects still need to be rendered once or more for each shadow-casting light. Furthermore, the lighting shader that applies shadows has a higher rendering overhead than the one used when shadows are disabled.

当然，带阴影的光照要比不带阴影的消耗要大，在延迟着色中，接受阴影的每个物体需要为每个阴影投射光进行一次或者多次渲染。此外，使用阴影照明的着色器要比不带阴影照明的着色器有更多的渲染开销。

**Implementation details**

**实现细节**

When deferred shading is used, the rendering process in Unity happens in two passes:

当使用了延迟渲染，Unity渲染会产生两个passes：

1. G-buffer Pass: GameObjects are rendered to produce screen-space buffers with diffuse color, specular color, smoothness, world space normal, emission and depth.

G-buffer Pass：物体会在屏幕空间的缓冲区渲染，带有漫反射，反射颜色，光滑度，世界空间法线，自发光和深度。

1. Lighting Pass: the previously generated buffers are used to add lighting into emission buffer.

光照Pass:先前产生的缓存会增加光照到发射缓存中

Objects with shaders that can’t handle deferred shading are rendered after this process is complete, using the [forward rendering](http://docs.unity3d.com/540/Documentation/Manual/RenderTech-ForwardRendering.html) path.

带着不能使用延迟渲染的着色器物体会等这个过程完成后，使用正向渲染路径渲染。

The default g-buffer layout is as follows:

默认的g-buffer布局：

* RT0, ARGB32 format: Diffuse color (RGB), occlusion (A).
* RT1, ARGB32 format: Specular color (RGB), roughness (A).
* RT2, ARGB2101010 format: World space normal (RGB), unused (A).
* RT3, ARGB2101010 (non-HDR) or ARGBHalf (HDR) format: Emission + lighting + lightmaps + reflection probes buffer.
* Depth+Stencil buffer.

深度模板缓冲区

So the default g-buffer layout is 160 bits/pixel (non-HDR) or 192 bits/pixel (HDR).

默认的g-buffer布局是160位像素（非HDR）或者192位像素（HDR）。

Emission+lighting buffer (RT3) is logarithmically encoded to provide greater dynamic range than is usually possible with an ARGB32 texture, when the Camera is not using HDR.

当相机没有使用HDR时，自发光的缓冲区对数解码比ARGB32贴图有更大的动态范围（dynamic range）。

Note that when the Camera is using HDR rendering, then there’s no separate render target being created for Emission+lighting buffer (RT3); instead the render target that the Camera renders into (that is, the one that is passed to the image effects) is used as RT3.

需要注意当相机使用HDR进行渲染时，系统不会为Emission+lightning缓冲（RT3）单独创建渲染目标（Render Target）。其次，相机使用的渲染目标（即被传入到图像效果的渲染目标）与RT3使用的相同。

**G-Buffer pass**

**G-Buffer pass**

The g-buffer pass renders each GameObject once. Diffuse and specular colors, surface smoothness, world space normal, and emission+ambient+reflections+lightmaps are rendered into g-buffer textures. The g-buffer textures are setup as global shader properties for later access by shaders (CameraGBufferTexture0 ..CameraGBufferTexture3 names).

G-buffer pass会渲染每一个物体。漫反射和镜面颜色，曲面平滑，世界空间的法线和自发光+环境光+反射光+光照贴图渲染到G-buffer贴图中。

G-buffer贴图被设置为全局的着色器属性，之后的shader可以访问。

**Lighting pass**

**光照pass**

The lighting pass computes lighting based on g-buffer and depth. Lighting is computed in screen space, so the time it takes to process is independent of Scene complexity. Lighting is added to the emission buffer.

光照的pass是基于g-buffer和深度的计算。光照在屏幕空间计算，因此他处理的时间依赖于场景的复杂度。光照会被添加到自发光的缓冲区中。

Point and spot lights that do not cross the Camera’s near plane are rendered as 3D shapes, with the Z buffer’s test against the Scene enabled. This makes partially or fully occluded point and spot lights very cheap to render. Directional lights and point/spot lights that cross the near plane are rendered as fullscreen quads.

当场景的Z缓冲区的测试（Z buffer’s test）开启时，没有穿过摄像机的近平面的点光源会渲染成3D形状。部分或者全部遮蔽的灯消耗很少。穿过近截面的方向光和点光会呈现为全屏幕的四边形。

If a light has shadows enabled then they are also rendered and applied in this pass. Note that shadows do not come for “free”; shadow casters need to be rendered and a more complex light shader must be applied.

如果一个灯光带有阴影，他们也会渲染到这个批次中。注意使用阴影不够自由，阴影接收者需要一个更复杂的着色器才能渲染。

The only lighting model available is Standard. If a different model is wanted you can modify the lighting pass shader, by placing the modified version of the Internal-DeferredShading.shader file from the [Built-in shaders](http://unity3d.com/support/resources/assets/built-in-shaders) into a folder named “Resources” in your “Assets” folder. Then go to the Edit->Project Settings->Graphics window. Changed the “Deferred” dropdown to “Custom Shader”. Then change the Shader option which appears to the shader you are using.

只有标准的光照模型是有效的，如果你想要一个不同的光照模型，你可以更改光照Pass的着色器，替换内置的延迟着色器到Assets下一个名为Resources文件夹下。选择Edit->Project Setting->Graphics window.更改“Deffered”下拉到“Custom Shader”。改变Shader选项到你使用的着色器。