Unity 光照

deepwaterooo

April 4, 2018

Contents

1 Unity Lights Lighting related

1.1 渲染管线中的 Rendering Path

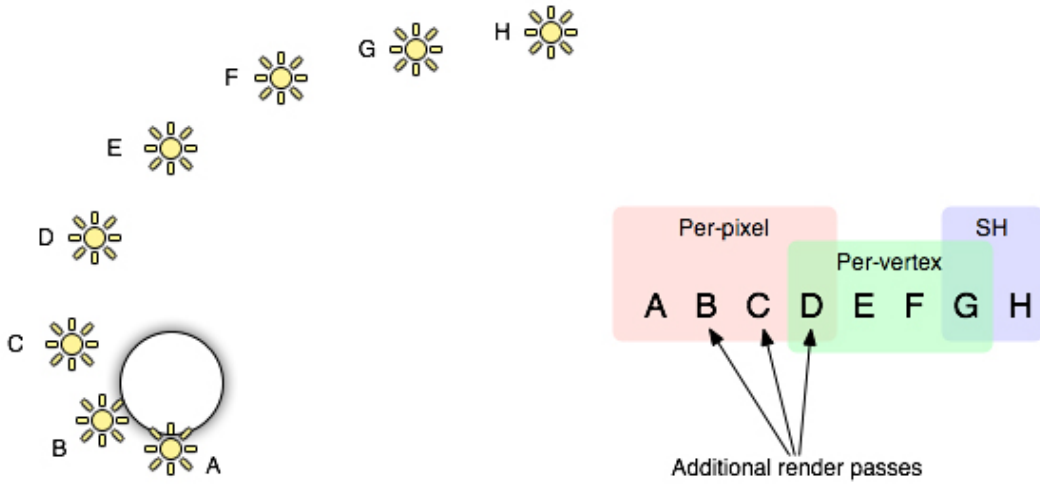
- https://yq.aliyun.com/articles/49406
- Unity3D 中，目前保存了 4 种 Rending Path.
  - 1、Deferred Shading Rendering Path(延迟渲染)
  - 2、Forward Rendering Path(前向渲染)
  - 3、Lagacy Deferred Lighting Rendering Path(延迟光照)
  - 4、Vertex Lit Rendering Path(顶点光照)

1.2 Forward Rendering Path(前向渲染)

- 官方文档：http://docs.unity3d.com/Manual/RenderTech-ForwardRendering.html
- 传统的前向着色渲染方式是一种简单粗暴的方式。就是一次性将一个物体绘制完再绘制下一个。光照计算也是如此。前向着色最主要的问题就是光照问题。当光源过多的时候，就需要进行处理。而 Unity3D 在光源处理方面，也结合了像素光照 (per-pixel lighting) ，顶点光照 (per-vertex lighting)，球谐光照 (Spherical Harmonics lighting) 等多种方案，其复杂程度可见一斑。

1.2.1 Unity3D 中前向渲染的光源策略

- 前向着色渲染由于效率和寄存器限制等原因。不可能老老实实地去处理所有的光源。因此，需要根据光源的远近，范围，重要程度等因子决定计算方式。
  - 1、如果一个光源的 RenderMode 没有设置为重要 (Important)，那么它将永远是逐顶点或者球谐光照计算方式。
  - 2、最亮的 (Brightest) 方向光源会做为逐像素光照计算。
  - 3、如果一个光源的 RenderMode 设置为重要 (Important), 那么它将会采用逐像素光照计算方式
  - 4、如果上面的结果中的用于逐像素计算的光源数目小于了 Project Settings->Quality 面板中的 Pixel Light Count 值。为了减少亮度误差，有一些光源会被当作像素光源计算（PS：非像素光源无法完全计算阴影关系。会导致场景偏亮）。
  - 5、Unity3D 对顶点光照，像素光照做了最大限制。分别为 4 个。而场景中不可能只有这么多光源。因此，使用了一个优先级判定。



  - 左：光源位置图
  - 右：光源分配图
- 从图中我们可以看到。D 参与了像素和顶点光照。G 参与了顶点光照和球谐光照。这样的处理应该是为了做一个光照信息的平滑衔接，不至于变得很突兀。

1.2.2 Unity3D 中前向渲染的光照的计算过程

- 前向渲染中，光照计算被分成了多个 PASS。通常它们由 1*FarwardBase Pass + N*FarwardAdd Pass 组成。可以看出，光源的数目在前向渲染管线中是一个致命的效率杀手。
  - 1、Base Pass 提交一个逐像素方向光和所有的逐顶点光照和球谐光照。
  - 2、Additional Pass 提交一个逐像素的光照计算。如果一个物体受更多的光源影响。那么就会有许多 PASS 需要进行绘制。
    * 注：在 Shader 中。可以指定 FarwardBase 来关掉 Additional 功能。
    * 注：在前向渲染中，只能有一个方向光实时阴影。

1.2.3 Unity3D 中的球谐光照

- 球谐光照是一个 CPU 光照算法。它可以处理大规模的光源信息，非常适合一些很小的动态点光源。但是它存在以下几个问题。
  - 1、球谐光照是基于物体顶点的算法，并非像素级。这就意味着它不能使用一些像素级的效果增强手段。比如法线贴图。
  - 2、基于效率的考虑，球谐光照的更新频率非常低。如果光源移动过快，就会穿邦。
  - 3、球谐光照是基于全局空间的计算，当一个面离点光源或者聚光灯很近时，效果是错的。

1.3 Deferred Shading Rendering Path(延迟渲染)

- 官方文档：http://docs.unity3d.com/Manual/RenderTech-DeferredShading.html

1.3.1 延迟渲染技术简介

- 延迟渲染技术已经是一个成熟且稳定的技术。已经广泛用于各种商业引擎中。它主要解决的是当光源数目过多时带来的复杂光照计算的开销。在传统前向渲染中，假设一个场景中有 M 个物体，N 个光源。那么理论上进行的光照计算次数为 M*N。而 DS 管线可以使其变为 M+N。
- 但是 DS 管线并非万能的。有以下情况需要注意

1.3.2 延迟渲染技术缺陷

- 1、DS 需要多渲染目标 (MRT) 的支持、深度纹理、双面模板缓存。
  - 这个特性，需要显卡支持 SM3.0 的 API。2006 年以后的 PC 显卡应该不成问题。比如 GeForce 8xxxx,AMD Radeon X2400,Intel G45 以后的显卡。
- 2、DS 管线会消耗更多的显存，用于缓存 G-Buffer。同时，会要求显卡拥有更大的位宽 (bit counts)。
  - 在手机上目前（2016 年）还不实现。
- 3、DS 管线无法处理半透明物体。
  - 半透明物体需要退回到传统前向渲染中进行
- 4、硬件抗锯齿（MSAA）无法使用。
  - 只能使用一些后期算法，如 FXAA 等。
- 5、这句话中的内容目前未测试，故无法准确体会其中的意思：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 will get graphical artefacts.

1.3.3 延迟渲染技术在 Unity3D 中的实现方案

- Unity3D 中，延迟渲染管线为分两个阶段进行。G-Buffer 阶段和光照计算 (Lighting) 阶段。

1.3.4 G-Buffer 阶段

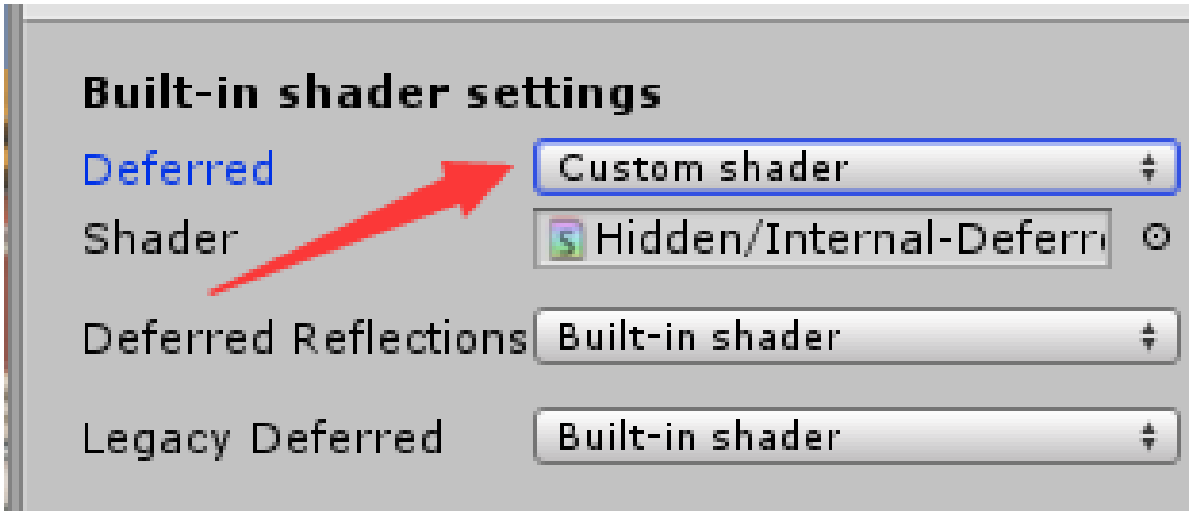
- Unity3D 渲染所有的非透明对象到各个 RT 中，RT 的内容分布见 G-Buffer 内容。Unity3D 将各个 RT 做成了全局变量，方面 Shader 中进行操作。像这样：CameraGBufferTexture0 .. CameraGBufferTexture3

1.3.5 光照计算阶段

- 这个阶段的主要目的就是根据 G-Buffer 的内容进行光照计算。由于是屏幕空间的计算，显然要比之前的前向渲染来得容易得多。
  - 注：实时阴影的计算是在光照计算之前的。DS 管线并不能减少实时阴影的开销。应该怎么整还得怎么整。最后，每个物体受到的实时阴影的影响会叠加到 RT3 中。一些特殊的爆光效果，Lightmap 光影贴图效果等都会进入 RT3。参见 G-Buffer 内容
- G-Buffer 内容
  - • 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.
  - 注：为了减少显存和渲染开销。当前场景的摄像机开启了 HDR 模式时，RT3 将不会被创建。而是直接使用摄像机的 HDR RT。

1.3.6 自定义 DS 管线

- Unity3D 中，标准的 DS 管线是 Standard 系列。如果你想修改某一个部分。你可以新建一个自己的 Shader。然后替换掉默认的即可。
- 替换的位置在 Edit->Project Settings->Graphics 面板中。把 Deferred 属性下拉框变成 Custom，就可以进行替换操作了。



1.4 Lagacy Deferred Lighting Rendering Path(延迟光照)

- 官方文档：http://docs.unity3d.com/Manual/RenderTech-DeferredLighting.html
- 技术文章：http://www.realtimerendering.com/blog/deferred-lighting-approaches/
  - 注：从 5.0 开始，DL 渲染已经不被 Unity3D 推荐，Unity3D 更推荐大家新项目使用 DS 管线方式。因为 DL 方式不好实现基于 PBR 着色的 Standard 材质，以及场景反射。
  - 注：如果摄像机设置为了正交投影。会强制退回前向渲染
  - 注：DL 也不会降低实时阴影的开销
- 和 DS 渲染管线一样，DL 的出现同样是为了解决光照计算的复杂度问题。然后不同的是，DL 仅仅把光照计算拿出去了。简单说来，DL 管线工作流程如下。
  - 步骤一、渲染场景中的对象，输出光照计算需要的 RT（深度、法线、高光信息）
  - 步骤二、使用上面的 RT 进行光照计算
  - 步骤三、再次渲染场景中的对象，并与计算机来的光照信息结合。
- 不难看出，DL 相比 DS 而言，不需要大量的 G-Buffer 支持。甚至不需要 MRT 的支持。但是对深度图的要求是必须的。如果遇上无法访问深度 BUFFER 的情况。那就需要做一次 Pre-Depth Pass 渲染。
- 比起 DS 而言，DL 由于不需要 MRT 的支持。在硬件特性需求和位宽上，少了许多开销。
  - 1、SM3.0
  - 2、PC:GeForce FX、Radeon X1300、Intel 965 / GMA X3100
  - 3、Mobile: 所有支持 OpenGL ES 3.0 的设备，部分支持 OpenGL ES 2.0 的设备。

1.5 Vertex Lit Rendering Path(顶点光照)

- 官方文档：http://docs.unity3d.com/Manual/RenderTech-VertexLit.html
  - 注：这不是一个通用的技术名词，只有 Unity3D 中有。
  - 注：主机平台这个不顶用。
  - 注：不支持实时阴影和高精度高光计算。

- Vertex Lit Rendering 就是指，所有的光照计算都通过顶点进行。而在 Unity3D 中，它的主要目的是为了支持那些没有可编程管线的设备。VL 也提供了几种方法，用于支持不同的材质类型。
  - 1、Vertex 使用 Blinn-Phong 进行光照处理，针对没有 lightmap 的对象使用。
  - 2、VertexLMRGBM 针对 lightmap 贴图使用 RGBM 加密（PC 和主机平台）的对象，没有额外的光照计算。
  - 3、VertexLMM 针对 lightmap 贴图使用 double-LDR 加密（移动平台）的对象，没有额外的光照计算。

1.6 Unity3D 对几种渲染管线的统一处理

- 在上面我描述中，我们发现，不管哪一种管线。都会涉及到几个部分。
  - 1、光照计算
  - 2、透明渲染
  - 3、阴影计算
  - 4、图像输出。
- 那 Unity3D 又是如何对这些进行统一流程控制的呢。我们看下面的图就明白了。
- 从图中我们可以看到。Unity3D 的传统渲染管线和延迟渲染管线，在非透明物体渲染和光照阶段是分离的。当处理完以后，紧接着处理非透明图像效果-> 天空盒-> 透明物体渲染-> 后期效果。

1.7 结束语

- 总的来说，Unity3D 的渲染管线还算稳定和易用，其的渲染管线包含了常见的三种方案。最主要目的还是在想着光照计算和效果显示的处理。同时保持对上层透明。其 Graphics Commander Buffer 提供了扩展管线的能力。而 Graphics 设置面板里，也提供了自定义延迟管线 Shader 的功能。然而也有诸多的美中不足，比如，正交摄像机模式下，不能使用延迟渲染技术。延迟光照下无法实现 PBR 和实时环境反射抽取等等。

1.8 Unity5 Lighting 面板说明书

- https://blog.csdn.net/u010026134/article/details/53673015

1.8.1 Object 面板

- Lightmap Static: 当前所选物体是否有勾上 Lightmap Static。如果该物体没有勾上 Lightmap Static，仍可用光线探针（Light Probe）来照亮。
- Scale In Lightmap: 该值影响当前物体对应 lightmap 上的像素数。默认值 1 代表像素数只取决于该物体表面区域（也就是说，各区域像素数相同）。该值越大（大于 1），像素数越多；该值越小（小于 1），像素数越少（lightmap 分辨率）。调整该值，有利于优化 lightmap，使重点细节区得到更精确的照射。举个例子，场景是带平坦黑色墙壁的某独立建筑，此时用小一点儿的 lightmap scale（小于 1）会更合适；若场景是一堆五颜六色的摩托车，用大点儿的值更合适。
- Important GI: 勾上，代表告诉 Unity 当前选择物体的光反射/放射会以显眼的方式影响其它物体，不让 Unity 优化该光源效果。
- Advanced Parameters: 当前选择物体的高级 lightmap 设置（可选择或创建）。
- Preserve UVs: 是否保护 UV 坐标（又可以叫” 是否接受 UV 优化”）。为了提高性能，Unity 会重新计算实时光照贴图的 UV 坐标。而在重新计算时，有时会对原 UV 坐标连续性判断出错。举个例子，把锐角错误判断成曲面。若勾上” 保护 UV 坐标”，则保留原有 UV 的效果；若不勾，则 Unity 会基于已烘培 UV，计算实时光照贴图 UV，加入临近 Charts，尽可能使 lightmap 紧凑。计算过程和下方 max distance, max angle 有关。实时 Charts 被半像素的边界包围，保证渲染时不会发生遗漏。
- Auto UV Max Distance: 如果 Charts 之间的 worldspace 距离比该值小，则简化 Charts。
- Auto UV Max Angle: 如果 Charts 之间的角度比该值小，则简化 Charts。
- Ignore Normals: 在实时全局光照检测 Charts 时，请勿比较法线（就是说这里要打勾）。手动编辑 UV 坐标时，可勾上该选项，以避免出现 Chart 分离问题。

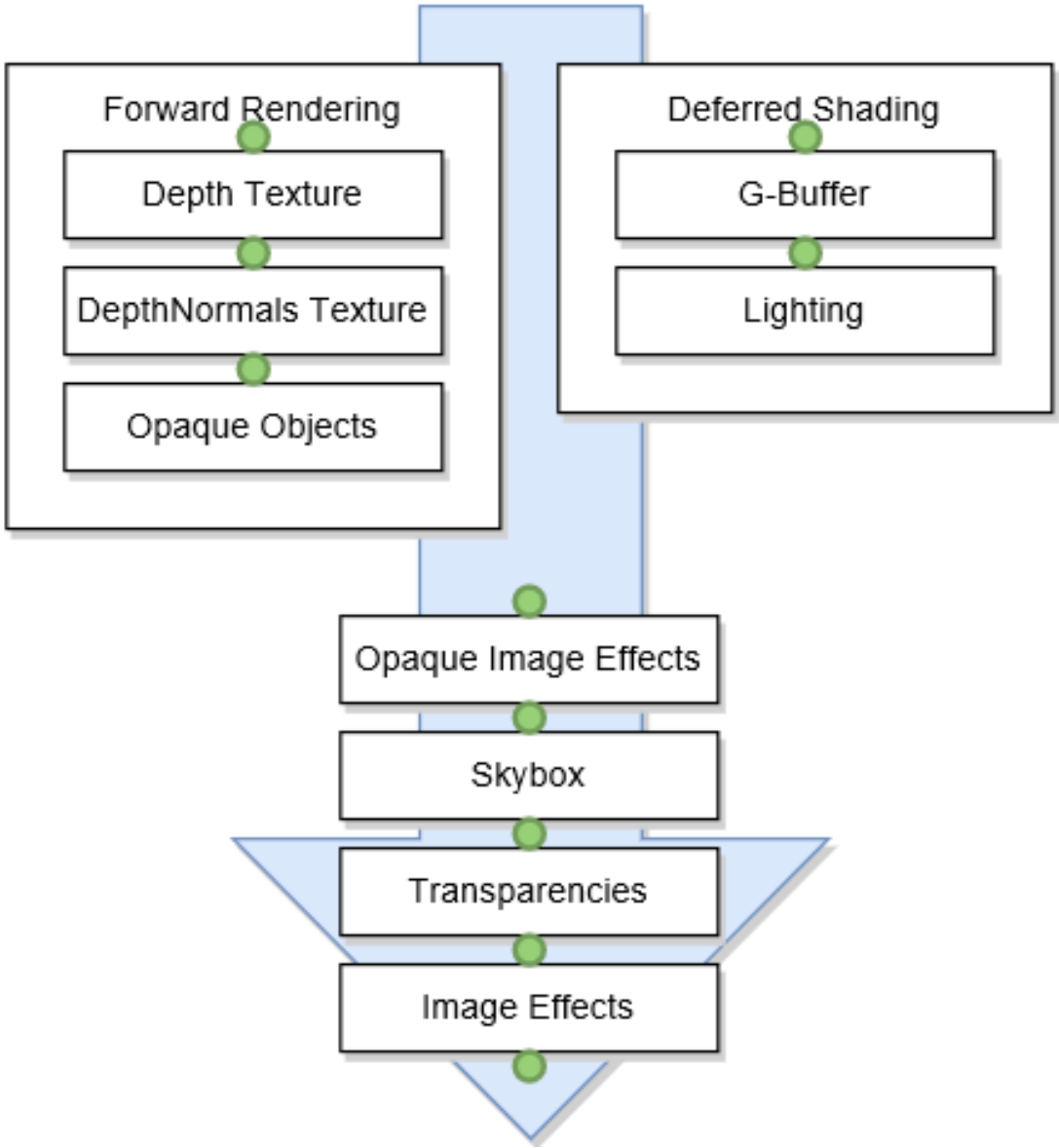
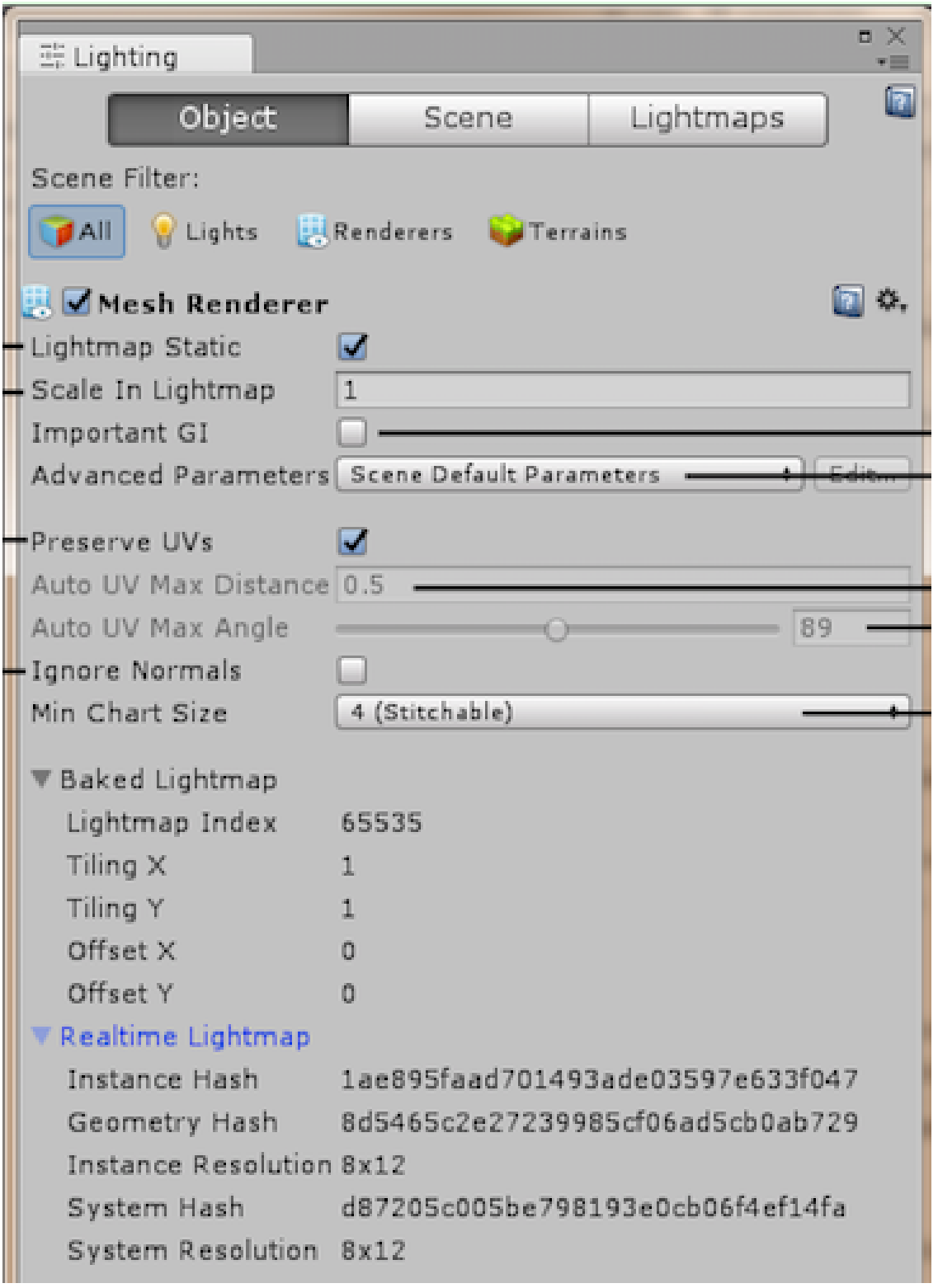
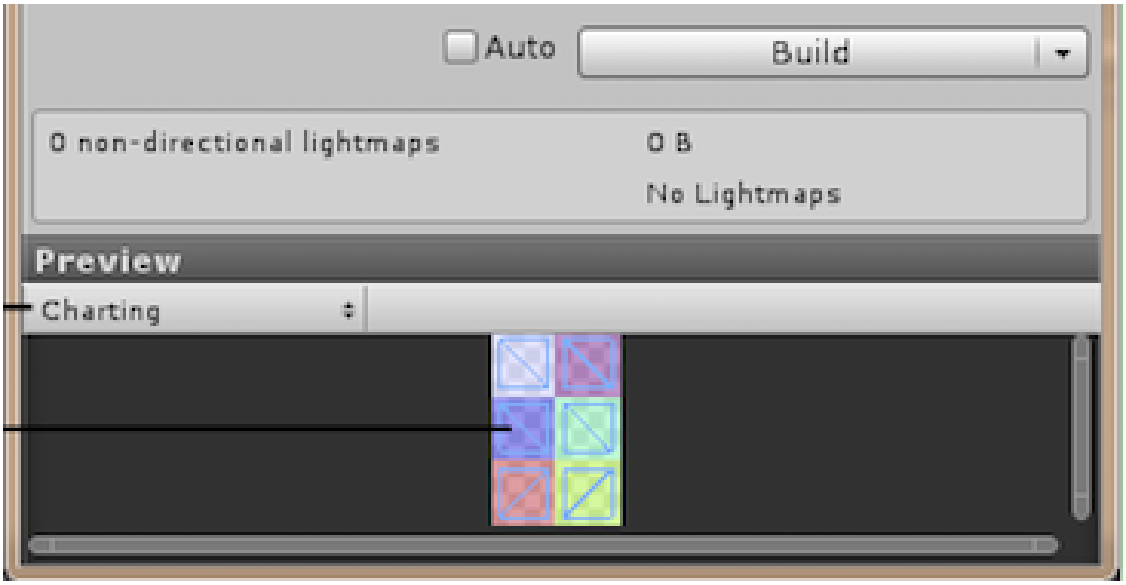


Figure 1: Unity3D 渲染管线图

- Min Chart Size: 这里涉及到 Unity 全新光照系统 Enlighten 的缝合特性，该特性使 Chart 平滑缝合（连接）在一起（比如说球体和圆柱体）。该特性需要每个 Chart 的每条边的方向数据，而方向数据是以块（block）单位存储的，则每个可缝合 Chart 最少需要 2*2 块。如果不需要精确缝合效果（比如说台阶模型），可选择 2（Minimum）。

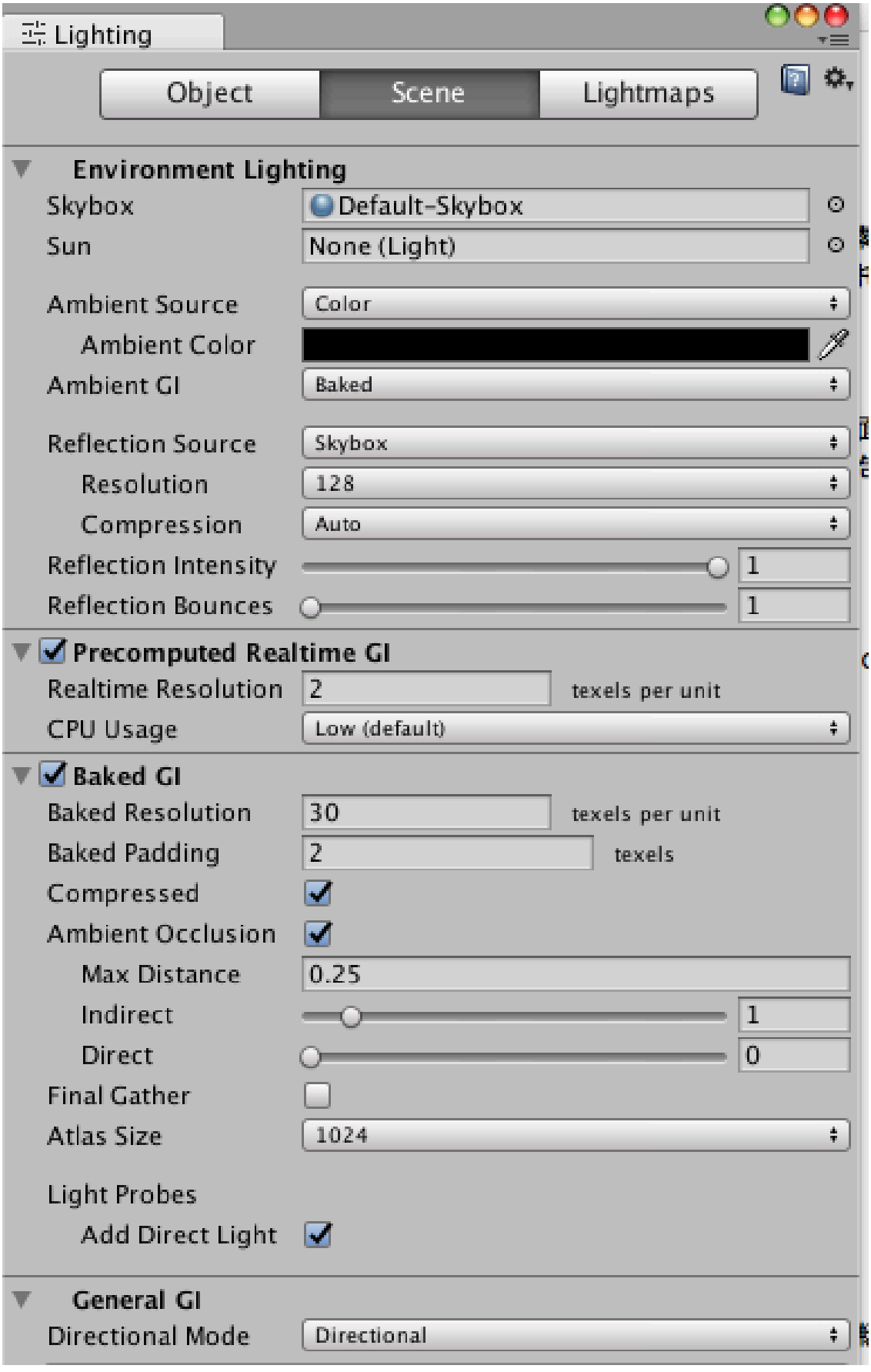

Lighting
Object
Scene
Lightmaps
Scene Filter:
All
Lights
Renderers
Terrains
Mesh Renderer
Lightmap Static
Scale In Lightmap
1
Important GI
Advanced Parameters
Scene Default Parameters
Edit...
Preserve UVs
Auto UV Max Distance
0.5
Auto UV Max Angle
89
Ignore Normals
Min Chart Size
4 (Stitchable)
Baked Lightmap
Lightmap Index
65535
Tiling X
1
Tiling Y
1
Offset X
0
Offset Y
0
Realtime Lightmap
Instance Hash
1ae895faad701493ade03597e633f047
Geometry Hash
8d5465c2e27239985cf06ad5cb0ab729
Instance Resolution
8x12
System Hash
d87205c005be798193e0cb06f4ef14fa
System Resolution
8x12



- Charting: 这个彩图就是 Chart，里面的蓝色线框就是当前选择模型的 UV（也可在 Scene 视图的 UV Charts 模式下浏览）。

1.8.2 Scene 面板

- Environment Lighting: 环境光照
  - Sun: 当使用天空盒时，可在此设置代表太阳的平行光（或者其它够大够远的照亮场景的光源）。如果设置为 None，则假定场景中最亮的平行光代表太阳。
  - Ambient Source: 环境光源
  - Ambient Intensity: 环境光亮度
  - Ambient GI: 指定环境光的全局光照模式
  - Reflection Source: 反向源，使用天空盒（天空盒也是 Cubemap）或其它自定义 Cubemap。如果不需要反射，请选自定义 Cubemap 但不赋值，这样就不会生成反射探针（Reflection Probe）。
  - Resolution: Cubemap 分辨率
  - Compression: 是否需要压缩反射探针（Reflection Probe）
  - Reflection Intensity: 在反射物上反射源（天空盒或自定义 Cubemap）的可见程度
  - Reflection Bounces: 反弹反射。在场景中用反射探针（Reflection Probe）捕捉该反射。该值决定了反射探针所检测物体之间的来回反射反弹次数。如果该值为 1，则只有初始反射（即来自上述 Reflection Source 的 Skybox/自定义 Cubemap 的反射）。

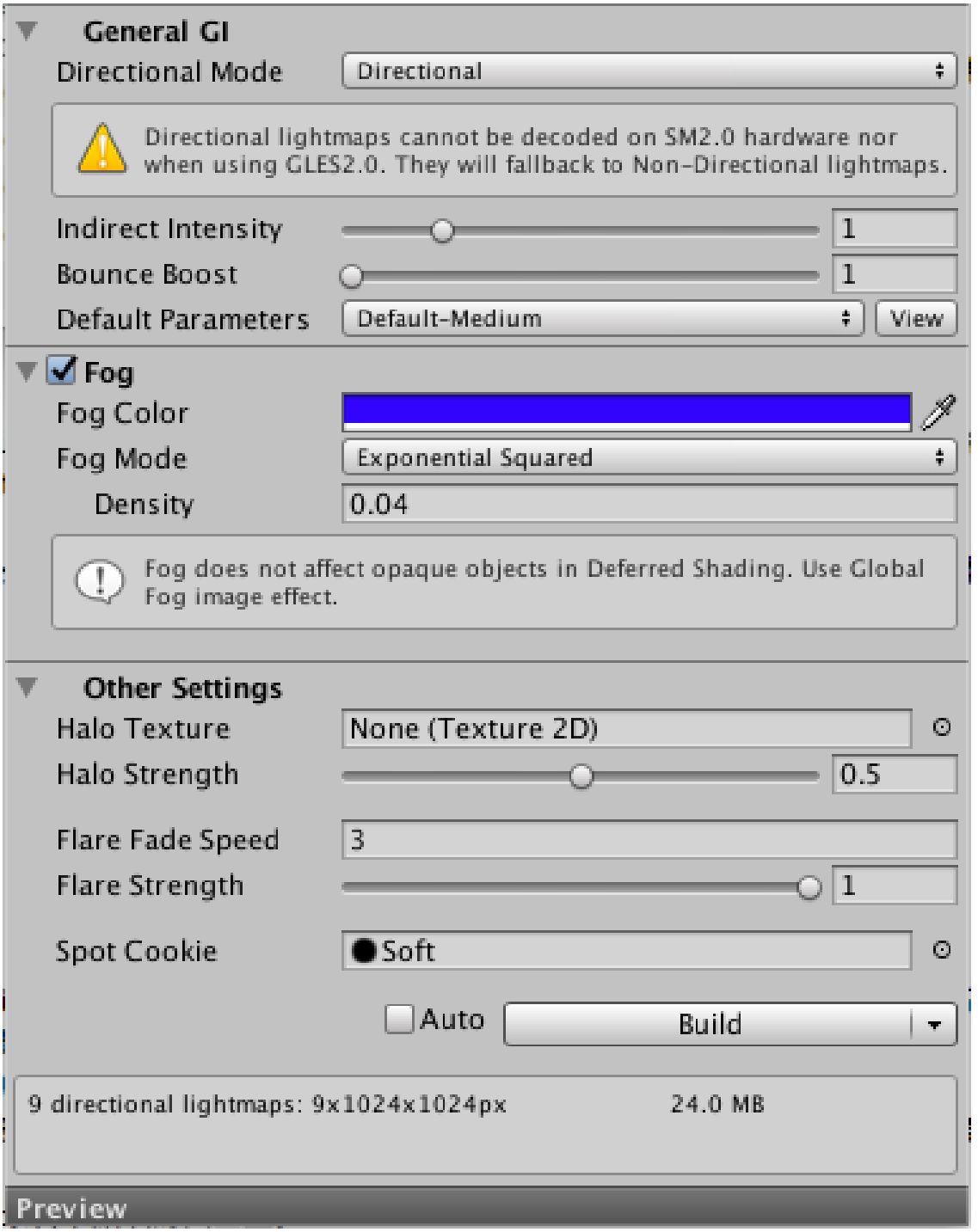


- Precomputed Realtime GI: 预处理实时全局光照
  - Realtime Resolution: 实时 lightmap 分辨率，每世界坐标单位多少纹素，通常每单位一纹素就已经有不

错的效果，如果是地形或大型物体，请适量调低该值。
  - CPU Usage: 运行时最终光照计算的 CPU 占用

- Baked GI: 烘培式全局光照
  - Baked Resolution: 烘培式 lightmap 分辨率，每世界坐标多少纹素
  - Baked Padding: 在 lightmap 上图形之间相距有多少纹素
  - Compressed: 是否压缩烘培式 lightmap（压缩后的 lightmap 可能会产生伪影）
  - Indirect Resolution: （只在 Precomputed Realtime GI 没有勾时才出现）计算间接照明分辨率。该数值等同 Precomputed Realtime GI 中的 Realtime Resolution。
  - Ambient Occlusion: 环境光遮蔽区表面的相对亮度值。该值越大，遮蔽区和无遮蔽区对比越鲜明
  - Final Gather: ” 最终收集”。当勾上时，在同分辨率已烘培 lightmap 中计算最终光反弹量。能提高 lightmap 可视性，但需要耗费额外的烘培时间。
  - Atlas Size: 整张 lightmap 贴图的纹素大小
  - Light Probes: 光线探针
  - Add Direct Light: 在光线探针中加入平行光。如果整个场景都用烘培式照明，但又需要有动态物体照明，请勾上该选项。如果场景实时照明，则光线探针只能放间接光

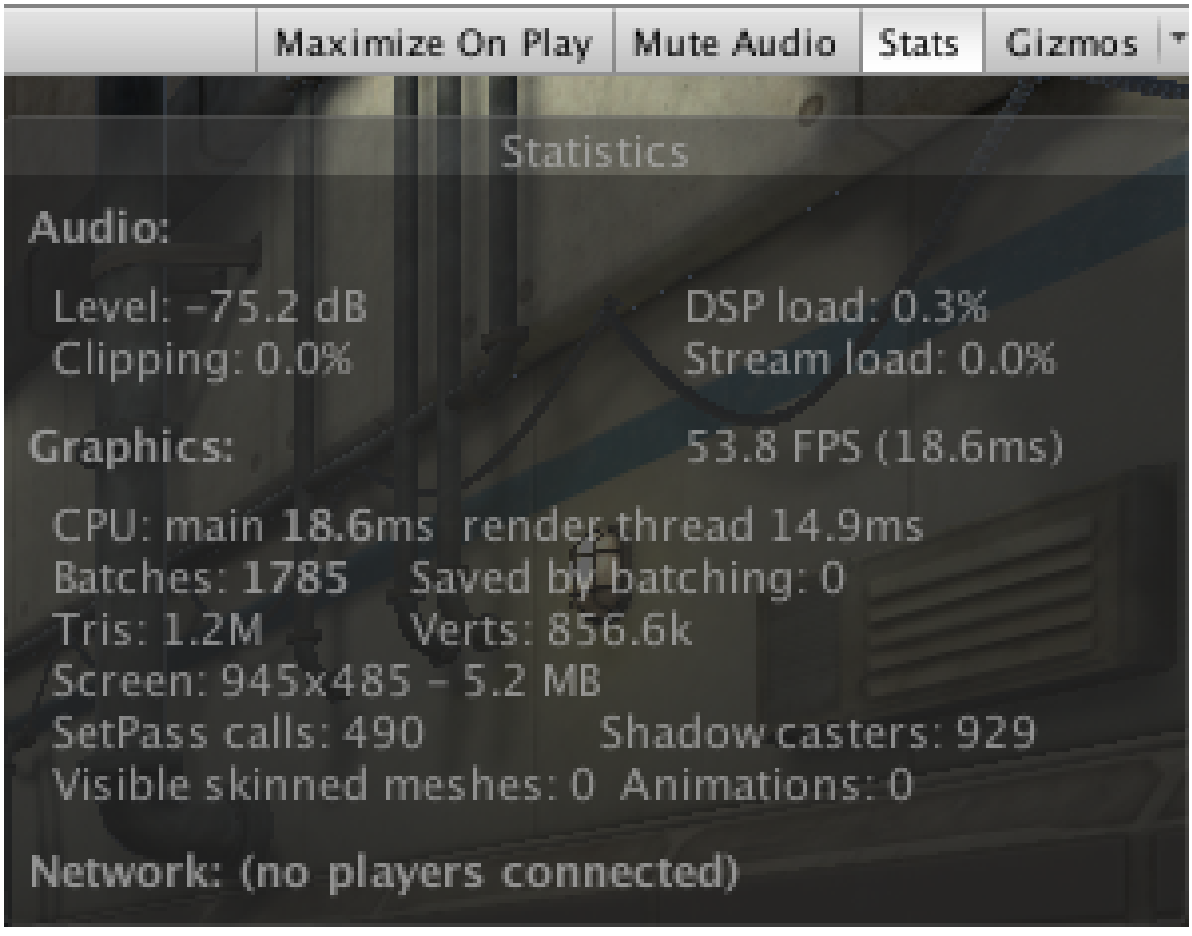
- General GI: 一般全局光照
  - Directional Mode: 通过设置 lightmap 的 directional 模式，可以存储物体表面每一点的主导入射光信息。在 directional 模式下，会生成第二张 lightmap 来存储主导入射光信息。Directional Specular 模式下，主要是配合漫反射法线贴图，甚至是高光反射法线贴图工作。Directional 模式需要两倍空间存储额外的 lightmap 数据，而 Directional Specular 需要四倍存储空间和两倍纹理内存
  - Indirect Intensity: 间接照明亮度值
  - Bounce Boost: 反弹（物体表面之间光反弹）增量
  - Default Parameters: 当前场景的默认 Lightmap 设置（可选择或创建）



- Fog: 雾效
  - Fog Color: 雾的颜色（如果渲染路径选择延迟渲染，则雾效不起作用）
  - Fog Mode: 雾效累积模式（线性累积、指数累积、指数平方累积）
  - Density: 雾效密度
- Other Settings: 其它设置
  - Halo Texture: 光晕贴图
  - Halo Strength: 光晕能见度
  - Flare Fade Speed: 镜头光晕淡出时间
  - Flare Strength: 镜头光晕能见度
  - Spot Cookie: 聚光灯（Spot Light）所用剪影（Cookie）

1.9 Stats 面板

- http://www.voidcn.com/article/p-kegvverb-ha.html
- Unity5 的 Statistics 上的统计信息和 Unity4 有一些区别，
- Statistics 窗口，全称叫做 Rendering Statistics Window，即渲染统计窗口（或渲染数据统计窗口），
- 窗口中罗列出关于渲染、声音、网络状况等多种统计信息，下面详细的解释一下这些项的意义。



- FPS
- FPS(Time per frame andFPS):frames per seconds 表示引擎处理和渲染一个游戏帧所花费的时间, 该数字主要受到场景中渲染物体数量和 GPU 性能的影响，FPS 数值越高，游戏场景的动画显示会更加平滑和流畅。一般来说，超过 30FPS 的画面人眼不会感觉到卡，由于视觉残留的特性，光在视网膜上停止总用后人眼还会保持 1/24 秒左右的时间，因此游戏画面每秒帧数至少要保证在 30 以上。
- 另外，Unity 中的 FPS 数值仅包括此游戏 Scene 里更新和渲染的帧，编辑器中绘制的 Scene 和其它监视窗口的进程不包括在内。
- CPU
  - CPU: 获取到当前占用 CPU 进行计算的时间绝对值，或时间点，如果 Unity 主进程处于挂断或休眠状态时，CPU time 将会保持不变。
- Render thread
  - Render thread:GPU 渲染线程处理图像所花费的时间，具体数值由 GPU 性能来决定
- Batches
  - Batches: 即 Batched Draw Calls, 是 Unity 内置的 Draw Call Batching 技术。

  - 首先解释下什么叫做“Draw call”，CPU 每次通知 GPU 发出一个 glDrawElements（OpenGl 中的图元渲染函数）或者 DrawIndexedPrimitive（DirectX 中的顶点绘制方法）的过程称为一次 Draw call, 一般来说，引擎每对一个物体进行一次 DrawCall，就会产生一个 Batch, 这个 Batch 里包含着该物体所有的网格和顶点数据，当渲染另一个相同的物体时，引擎会直接调用 Batch 里的信息，将相关顶点数据直接送到 GPU, 从而让渲染过程更加高效，即 Batching 技术是将所有材质相近的物体进行合并渲染。
  - 对于含有多个不同 Shader 和 Material 的物体，渲染的过程比较耗时，因为会产生多个 Batches。每次对物体的材质或者贴图进行修改，都会影响 Batches 里数据集的构成。因此，如果场景中有大量材质不同的物体，会很明显的影响到 GPU 的渲染效率。这里说几点关于 Batches 优化相关的方案
  - 虽然 Unity 引擎自带 Draw Call Batching 技术，我们也可以通过手动的方式合并材质接近的物体;
  - 尽量不要修改 Batches 里物体的 Scale，因为这样会生成新的 Batch。
  - 为了提升 GPU 的渲染效率，应当尽可能的在一个物体上使用较少的材质，减少 Batches 过多的开销;
  - 对于场景中不会运动的物体，考虑设置 Static 属性,Static 声明的物体会自动进行内部批处理优化。

- Verts 和 Tris
  - Verts：摄像机视野 (field of view) 内渲染的顶点总数。
  - Tris: 摄像机视野 (field of view) 内渲染的的三角面总数量。- 关于 Tris 和 Verts, 突然想到一些问题，这里需要多嘴说几句：
  - Camera 的渲染性能受到 Draw calls 的影响。之前说过，对一个物体进行渲染，会生成相应的 Draw call，处理一个 Draw Call 的时间是由它上边的 Tris 和 Verts 数目决定。尽可能得合并物体，会很大程度的提高性能。举个很简单例子，比如场景一种有 1000 个不同的物体，每个物体都有 10 个 Tris；场景二中有 10 个不同的物体，每个物体有 1000 个 Tris。在渲染处理中，场景一中会产生 1000 个 Draw Calls，它的渲染时间明显比场景二慢。
  - Unity stats 视图中的 Tris 和 Verts 并不仅仅是视锥中的梯形内的 Tris 和 Verts，而是 Camera 中 field of view 所有取值下的 tris 和 verts，换句话说，哪怕你在当前 game 视图中看不到这个 cube，如果当你把 field of view 调大到 179 过程中都看不到这个 cube，stats 面板才不会统计，GPU 才不会渲染，否则都会渲染，而且 unity 不会把模型拆分，这个模型哪怕只有 1 个顶点需要渲染，unity 也会把整个模型都渲出来。（参考自 Mess 的《Unity Camera 组件部分参数详解》）
  - 之前有童鞋问过我，新建一个空的场景，里边没有添加任何物体，为什么 Status 面板上显示有 1.7k Tris 以及 5.0kVerts。这是因为空的场景自带默认的天空盒。
  - 点击 Windows—Lighting 打开 Lighting 下的 Scene 面板，把 Skybox 里的材质设为空
  - 删掉它，你就会发现 Tris 和 Verts 都变为 0 了（以 Unity 5.5.0 为例）

- Screen
  - Screen: 获当前 Game 屏幕的分辨率大小，后边的 2.1MB 表示总的内存使用数值。

- SetPass calls
  - SetPass calls：在 Unity 4.x 和 3.x 原来的 Stats 面板的第一项是“Draw calls”，然而到了 Unity5.X 版本，Stats 上没有了“Draw calls”，却多出来一项”SetPass calls“。
  - 比如说场景中有 100 个 gameobject, 它们拥有完全一样的 Material, 那么这 100 个物体很可能会被 Unity 里的 Batching 机制结合成一个 Batch。所以用“Batches”来描述 Unity 的渲染性能是不太合适的，它只能反映出场景中需要批处理物体的数量。那么可否用“Draw calls”来描述呢？答案同样是不适合。每一个“Draw calls”是 CPU 发送个 GPU 的一个渲染请求，请求中包括渲染对象所有的顶点参数、三角面、索引值、图元个数等，这个请求并不会占用过多的消耗，真正消耗渲染资源的是在 GPU 得到请求指令后，把指令发送给对应物体的 Shader, 让 Shader 读取指令并通知相应的渲染通道（Pass）进行渲染操作。
  - 假设场景中有 1 个 gameobject，希望能显示很酷炫的效果，它的 Material 上带有许多特定的 Shader。为了实现相应的效果，Shader 里或许会包含很多的 Pass, 每当 GPU 即将去运行一个 Pass 之前，就会产生一个“SetPass call”，因此在描述渲染性能开销上，“SetPass calls”更加有说服力。

- Shadow casters
  - Shadow casters：表示场景中有多少个可以投射阴影的物体，一般这些物体都作为场景中的光源。

- visible skinned meshed

  - visible skinned meshed：渲染皮肤网格的数量。

- Animations
  - Animations: 正在播放动画的数量。

- 其它
  - 目前渲染统计窗口的参数就只有这些，如果你想了解更多的渲染信息，可以打开 Unity 的 Profiler 窗口（右键–AddTab—Profiler）, 这儿你可以获取到更多的渲染数据，比如“Draw Calls”、”VBO Totals”、”VB Uploads” 等等，还能实时观察 CPU、内存和音频的使用情况。

1.10

-