# **理解前向渲染与延迟渲染**

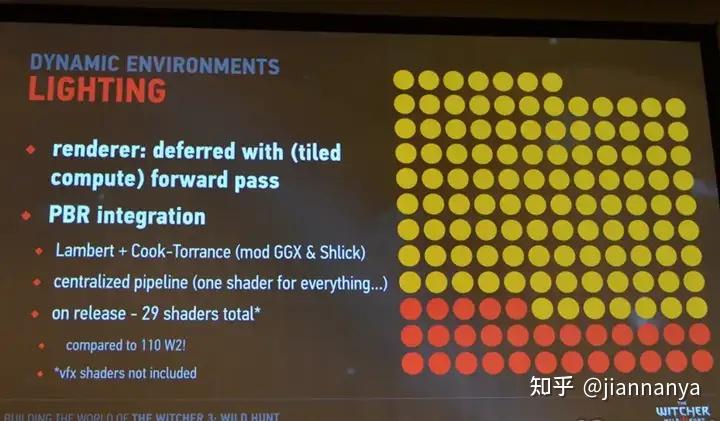
## **一、序**

2015年5月，3A巨作《[巫师3：狂猎](https://zhida.zhihu.com/search?content_id=250572703&content_type=Article&match_order=1&q=%E5%B7%AB%E5%B8%883%EF%BC%9A%E7%8B%82%E7%8C%8E&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)（The Witcher 3：Wild Hunt）》横空出世，该游戏作为一款开放世界的角色扮演游戏，其巨大的开放世界、绵长的剧情故事、画面精美的表现力深深吸引着大量玩家，并在当年一举拿下**年度最佳游戏、最佳RPG、金操纵杆奖、最佳故事讲述、最佳视觉设计等，直到现在，共计获得超过800个奖项**。用如今2024年的眼光回看巫师3的画面，也仍是相当不错的水平，你要知道这可是一款快10年的老游戏了。《巫师3:狂猎》的画面在当年可是游戏圈顶尖的存在，帧帧美如画的美誉可不是凭空吹出来的，正是靠着过于“硬核”的质量在2018年仍能拿下steam的最佳环境奖，直到现在仍有着97%的好评率。



来源：巫师3:狂猎游戏原图

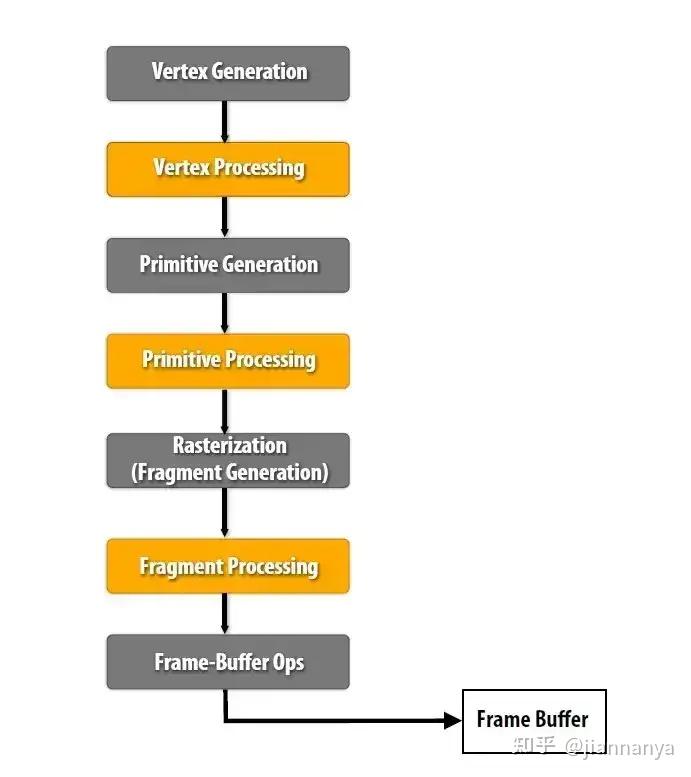
同年8月，被誉为计算机图形学顶会的SIGGRAPH 2015召开，报告的其中一个，就是巫师三图形引擎的详细解说。值得注意的是，游戏精美的画面主要来自于以**延迟渲染（Deferred Rendering）为主，部分前向渲染（Forward Rendering）**的渲染方式。熟悉图形渲染的小伙伴应该了解，在过去，前向渲染一直是主流技术，随着计算机图形学和处理能力的不断进步，渲染引擎也在不断演化和改进，渲染方式慢慢从前向渲染过度到延迟渲染为主，部分前向渲染的方式。在本篇中，小编将探讨渲染引擎的演化历程，**剖析前向渲染和延迟渲染的原理与特点，深入分析它们的优缺点以及适用场景**，旨在为读者呈现一个全面的视角，更好地理解渲染技术的发展趋势和未来方向。



SIGGRAPH 2015 Killing Monsters: Behind the Scenes of the Witcher 3: Wild Hunt 图形引擎解说现场 ，来源：日本媒体4gamer

## **二、前向渲染（Forward Rendering）**

前向渲染是早期游戏中常用的一种渲染技术。其基本原理是在渲染管线的几何处理阶段将场景中的物体转换为屏幕空间的顶点，并进行光照计算。在光照阶段，**每个像素都需要遍历场景中的所有光源**，并计算其对该像素的影响，然后将所有光照计算结果合并以得到最终的像素颜色。



前向渲染管线流程，来源：毛星云，【《Real-Time Rendering 3rd》 提炼总结】(七) 第七章续 · 延迟渲染(Deferred Rendering)的前生今世

前向渲染的核心伪代码可以表示为：

For each light:

For each object affected by the light:

framebuffer **+=** object **\*** light

从流程图可以看到，运用前向渲染方法渲染N个物体，在M个光源下，需要执行N\*M次的渲染次数，**复杂度为O(N\*M)**。所以前向渲染中场景光源数量对计算复杂度影像巨大，比较适合单一光源，或者光源较少且固定的场景。在当时硬件条件还没有那么强的时候，前向渲染还是各大游戏产品的主流渲染方式，因为这些游戏场景复杂度并不高，可能只有一个定向光或者几个局部光源。例如，在2001年的合金装备2（Metal Gear Solid 2）中，由于环境场景光源有限，因此采用了正向渲染，游戏内灯光是动态的，玩家和npc都可以利用阴影区域进行掩护和隐藏。



正向渲染的固定光源场景，来源：合金装备2游戏截图

而在2015年，随着计算机硬件能力的提升，考虑到世界场景的复杂度，Konami（科乐美）工作室采用了基于延迟渲染的Fox引擎，推出了合金装备5：幻痛（Metal Gear Solid V: The Phantom Pain）。



延迟渲染适合复杂场景，来源：合金装备5：幻痛游戏截图

由此，我们可以总结出前向渲染的优缺点：

### **前向渲染优势：**

1. 由于场景固定，物体渲染时需要逐个考虑光源，此时，物体材质和光源属性耦合，即光源信息和光照计算都放在了物体渲染的片元着色器中，此时**模型的各种材质都能使用不同的光照模型和渲染技术**，如卡通渲染、各向异性材质等。
2. **透明材质**的物体也能很好的进行光照计算。

### **前向渲染劣势：**

1. 每个物体都需要进行光照计算，无论这个物体是否可见，由此会带来**过多overDraw**（无意义的渲染）问题。
2. 显而易见，对**多光源的支持不好**。
3. 在每个物体材质着色器中都需要增加光源计算。

## **三、延迟渲染（Deferred Rendering）**

**“评价一个画面好坏的标准，一个简单的标准是说这个画面是否够亮”**——闫令琪（2019年获顶级计算图形学机构 ACM SIGGRAPH颁发的最佳博士论文奖得主）



光带来的画面张力，来源：PlayStation 独占游戏：对马岛之魂（Ghost of Tsushima）

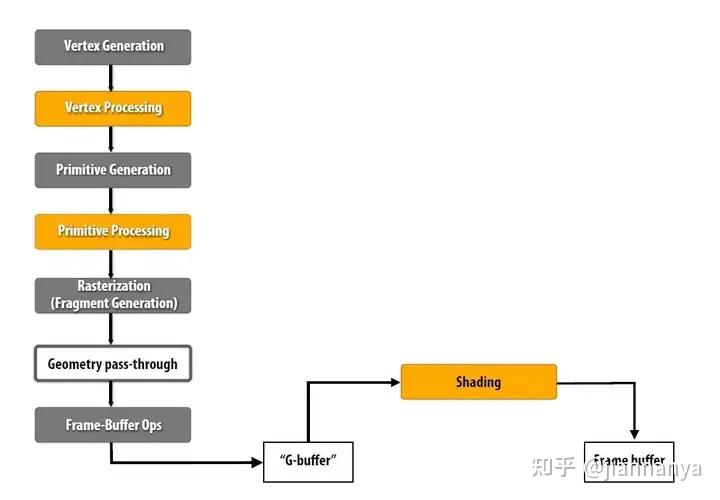
现代游戏画质的飞速提升，**最重要的一点就是光源的数目增加**，如今看起来非常真实的3A游戏场景中往往部署着环境光、局部光等上百个光源，如此前向渲染的性能难以支撑，无法达到实时渲染的要求。怎么样减少绘制次数，**将物体材质与光照计算解耦开来呢**？那就得请出我们的延迟渲染。

**延迟渲染( Deferred Rendering)，即延迟着色（Deferred Shading）**，顾名思义，是将着色计算延后进行处理的一种渲染方法。延迟着色给我们优化拥有大量光源的场景提供了很多可能性，因为它能够在渲染拥有成百上千光源的场景的同时还能够保持能让人接受的帧率。下面这张图展示了一个基于延迟着色渲染出的场景，这个场景中包含了1000个点光源，对于目前的硬件设备而言，用传统的正向渲染来实现几乎是不可能的



基于Deferred Rendering 渲染的含1000个点光源的场景，来源：J. Andersson, SIGGRAPH 2009 Beyond Programmable shading course talk @ Frostbite2（寒霜）引擎

延迟渲染管线流程：



延迟渲染管线流程，来源：毛星云，【《Real-Time Rendering 3rd》 提炼总结】(七) 第七章续 · 延迟渲染(Deferred Rendering)的前生今世

延迟渲染的核心伪代码可以表示为：

For each object:

Render to multiple targets For each light:

Apply light as a 2D postProcess

从上面的伪代码以及流程图中我们可以看出延迟渲染主要是分为了两个阶段，**几何处理阶段(Geometry Pass)**：渲染所有的几何/Color数据到G-Buffer（Geometry Buffer，几何缓冲区），**光照处理阶段(Lighting Pass)**：使用G-Buffer计算场景的光照。得益于硬件对于MRT（Multiple Render Target，多重渲染目标）的支持。让延迟渲染成为了可能。延迟渲染的核心在于G-Buffer，也就是若干张贴图，存的是计算光照需要的所有参数，通常包含了Depth（深度图），Normal（法线贴图），Albedo（反射率贴图），Roughness（粗糙度），Specular（高光）和Metallic（金属度）。

还是以合金装备5：幻痛为例，该游戏中的一帧画面的G-Buffer如下：



合金装备5：幻痛中的一帧G-Buffer，来源：Adrian Courrèges，Metal Gear Solid V - Graphics Study

Deferred Render在G-Buffer生成之后，会根据场景中的光源，逐个计算场景的Diffuse（漫反射）和Specular（高光）：



合金装备5：幻痛的Diffuse和Specular，来源：Adrian Courrèges，Metal Gear Solid V - Graphics Study

最后，将上述几张贴图进行相关计算混合，最后生成了游戏中的一帧：



最终场景，来源：合金装备5：幻痛游戏截图

由此，我们可以总结出延迟渲染的优缺点：

### **延迟渲染优势：**

1. 对**多光源支持良好**，保证物体和光源只绘制一次，避免重复渲染，渲染n个物体在m个光源下的着色，复杂度从O(n\*m)减少到了**O(n+m)次**。
2. 运用G-Buffer，先执行物体的深度检测，再执行着色计算，**避免了overDraw**的现象。
3. **解耦了材质和光源**，使得每个着色器都专注于几何参数提取或者照明。这种分离使着色器的功能进行拆分，简化了着色器系统管理。

### **延迟渲染劣势：**

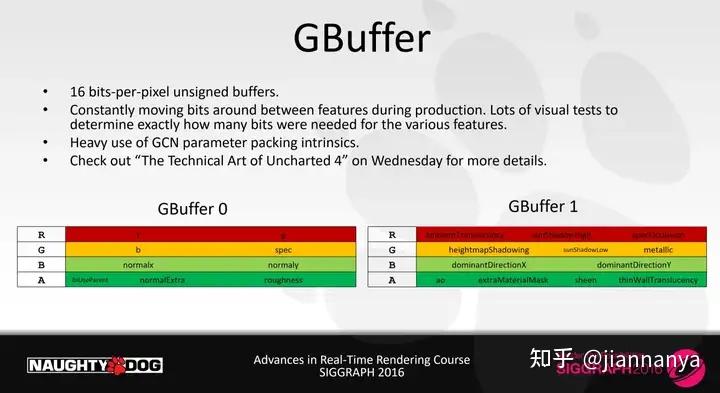
1. G-Buffer的存在导致了**内存开销较大**，需要存储如深度、法线等贴图。贴图数量增多**会额外消耗带宽**。
2. 需要**MRT支持**。
3. 由于透明材质的物体无需记录深度，故延迟渲染**很难处理透明材质**物体。

## **尾**

可以看到，延迟渲染虽然弥补了前向渲染的一些弊端，成为越来越多的现代游戏引擎的首选方法。但延迟渲染这把“双刃剑”用的好的同时也在不断把压力给到游戏研发人员，**怎样以极低的代价压缩G-Buffer空间？怎样单独处理透明模型？**等等这些问题是如今研发人员需要不断思考的。  
  
游戏行业的老大哥顽皮狗（Naughty Dog）在《神秘海域4：盗贼末路（Uncharted 4: A Thief’s End）》给出了良好的示范，其在2016年的SIGGRAPH上以《Deferred Lighting in Uncharted 4》为题做了技术汇报，针对延迟渲染管线中材质多样性导致G-Buffer开销大的问题，顽皮狗极力**压榨G-buffer中每个通道**，最后G-buffer压缩到了只有三张。



顽皮狗经典IP：神秘海域，来源：《神秘海域4：盗贼末路》游戏截图



神秘海域4中的G-Buffer设计，来源：顽皮狗在SIGGRAPH 2016主题演讲

除了压缩G-buffer，如开头所说，巫师三采用了延迟渲染为主，部分前向渲染的方式，最终目的是解决透明材质物体无法写入深度到G-Buffer中的问题。除此以外，**分块延迟渲染，又称之为forward+（Tile-Based Deferred Rendering，TBDR）**作为传统延迟渲染的另一种改进，也受到了越来越多实时渲染引擎的青睐。