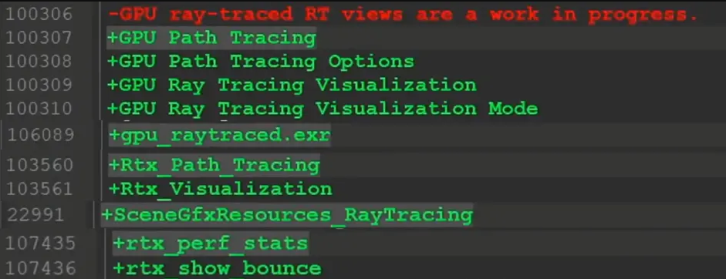
注意！此文档是残缺的（缺失部分图片，视频专栏链接，无数学公式表述），可能影响阅读。

进入 bilibili.com/read/cv23384330 获得最完整的阅读体验。作者：B1ueMicr0

1. 起源 2 引擎的图形管线

目前起源 2 的图形管线是纯光栅化渲染管线。为什么要提这个呢？因为 V 社正在筹备大动作，也就是对起源 2 的实时光线追踪管线的支持。



这对于起源 2 游戏玩家以及地图制作者们来说是非常重大的一件事，这意味着一些极速快感。附录 A 会对光线追踪管线进行简单介绍。

我们知道，目前主流渲染模式分为两大类：前向渲染（Forward Rendering）和延迟渲染（Deferred Rendering），并且两者是可以共存的。（例如使用延迟渲染的游戏可以多开一个 Pass 走前向管线处理半透明物体）

起源 1 作为一个将要有 20 年历史的引擎，它使用了符合它那个年代定位的前向渲染模式。前向渲染提供了十分优异的多重采样抗锯齿（MSAA）的支持。对于此反走样模式的更多技术细节，推荐阅读《Real-Time Rendering 4th》的 Chapter 5 部分。

起源 2 也仍在使用这种基本思路，但是需要注意的一点是：起源 2 并不是传统的前向渲染。

我们知道，由于场景光源的增加，传统前向渲染的劣势被放大。对于每个需要逐像素计算的光源，渲染一个几何体的时候需要逐个做一次光照计算，时间复杂度为 ，其中 m 为场景几何体数量，n 为光源数量。举个例子，若场景内有20个几何体，其中放置了 5 个光源，那么整个场景的渲染过程就需要进行 100 次光照计算。

并且这里大量的计算属于是 Overdraw，也就是管线的最终阶段会抛弃掉大量的在这之前已经辛辛苦苦算好的光照结果。

这里给出起源 1 引擎的 Garry‘s Mod 作为示例，可以看到，生成了多个光源后，帧率从 300 骤降到 100 .（测试 GPU 为 Nvidia RTX 3070 Ti）

bilibili.com/video/BV1zg4y1L73T

当然，优化方法肯定是有的。这里我们可以使用 Culling，例如 Early-Z, Z-prepass 等。它们本质上都是在计算光照的阶段之前把受遮挡的图元剔除，以此减少计算量。还有一种就是大家喜闻乐见的延迟渲染了。延迟渲染把场景先绘制在 G-buffer 里，再逐光源对缓冲区进行着色。因其分离了 Mesh Draw 和 Light Draw，让物体只被绘制一次，光源也只绘制一次，时间复杂度随之大大降低到了 O (m + n)

起源 2 引擎同时提供了对前向渲染和延迟渲染的支持（见 Valve 开发者社区网站）。不过遗憾的是，由于起源 2 SDK 目前仍开放不全，我无法测试延迟渲染在起源 2 中的具体效果。

前向渲染在像素着色器中可以获取到所有的光照和几何信息，每种材质可以使用不同的光照模型和渲染。这也是 Garry 在接受 s&box 访谈的时候鼓吹 s&box 支持可无限拓展的着色器的原因。而作为延迟渲染，为了优化光照计算而做出了许多着色器功能的分离，使着色器系统管理得到简化，虽然提高了效率，但也因此降低了着色器的灵活性，使其支持的着色器数量很少。

这里让我们回到刚刚那个“起源 2 并不是传统的前向渲染”的话题，起源 2 引擎提供了一种新的模式，叫做 Tile-Based Forward Rendering（Forward+）的渲染模式。

布满屏幕的 Tile

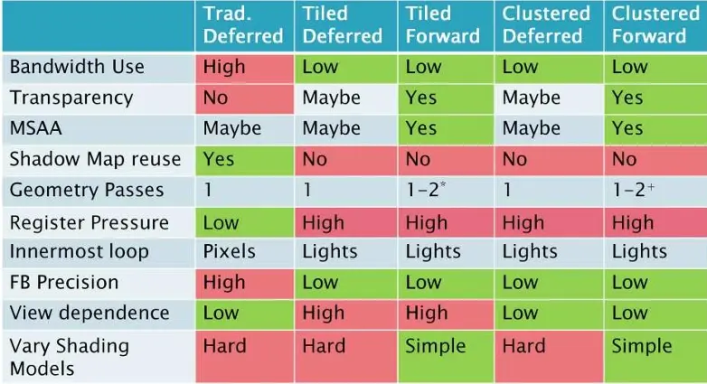
Forward+ 保留了作为前向渲染优秀的多着色器支持，这使得卡通渲染，各向异性材质等很轻松地在游戏里实现。例如 s&box 社区的 MToon Shader，这里放出一个 s&box 游戏内的演示。

  (@AperturePenguin) bilibili.com/video/BV1na4y1P7m8

起源 2 引擎的 Forward+ 模式，相较于起源 1 的传统前向渲染提供了更优秀的多光源 Culling，具体是在着色之前就利用 depth 把 Tile 内的无关光源都剔除掉，只对 Tile 有影响的光源才做光照计算，这样大幅降低了多光源的计算压力。起源 2 使用 Forward+ 渲染模式，既可以从前向渲染天生具有的优势中受益，又可以很好地支持多光源。

bilibili.com/video/BV1ts4y1e79r

综上，我们也放出下列表格来对比几种渲染的优缺点：（起源 2 同时支持表格中的前三项技术，最后两项有待证实）如果某一天起源 2 SDK 开放了渲染模式的选择，那么社区开发者们可以通过这个思考怎么最大化最终帧率。



可以看出，起源 2 的 Forward+ 模式保留了优秀的 MSAA 抗锯齿，这对像 CS 2 玩家这类群体是非常利好的。延迟渲染因为做 MSAA 撑大了整个 G-buffer 数倍之多，造成的带宽瓶颈非常严重，性能开销极大。

这里需要详细说明：延迟渲染能做 MSAA. 早在十多年前的 DirectX 10 时期，就已经有了硬件支持 MSAA 的 MRT。现在流行的一种 “延迟渲染不支持 MSAA” 的说法是完全错误的。

如果延迟渲染学前向渲染一样，光照计算结束之后才做 sampling，到那会儿 G-buffer 里各种原始信息都被覆盖了，插完值得到的只会是错误结果。不仅开销大效果还拉，而并不是延迟渲染不能做 MSAA.

不过就算是 Forward+，V 社同时还集成了 AMD FSR 2.0 的支持，这是一种使用 temporal filtering 的抗锯齿（鉴于 FSR 的功能，准确来说是一种 upscaling），类似于 TAA，各档位可为低端硬件提供高质量的画质。最近 Facepunch 还为起源 2 的渲染管线中添加了另一些 FidelityFX 套件的东西— FFX 随机屏幕空间反射 以及 AMD 降噪器。

您可能会有疑问：既然是电脑游戏了，PC 显卡不都是 IMR 渲染架构? 怎么还用 Tile-Based 渲染？参见 附录 B ，起源 2 的这个特色不是为了省带宽。

2. 基于物理渲染的材质—PBR Texturing

起源 2 图形最大的改进之一当属基于物理渲染（PBR）

提到 PBR，我们就不得回避 BRDF（双向反射分布函数）这个概念了。（详细的定理介绍在 附录 C ）这里简述：

反射光亮度 = BRDF \* 入射辐照度

（此关系有一个具体的称谓，详情见附录 D）BRDF 确定了辐照度和光亮度之间的关系，并且不是经验方程。它是一种应用在计算机图形学中非常理想化的物理模型。虽然是 Physically-Based ，但它却不需要光线追踪来完成。可以广泛应用在传统光栅化管线的游戏当中。起源 2 引擎使用了 PBR 的流程来处理 Material. 而起源 1 的 Phong 模型则是对 BRDF 进行快速估计的经验方程，不满足物理定律但是效率高。

这里注意：Apex, Titanfall 2 也是起源 1 引擎，但是它们已经和原版起源大相径庭，魔改出了 PBR 渲染系统。并且有消息可知，《猎杀弗里曼》也有了 PBR 系统。（CV21018884）

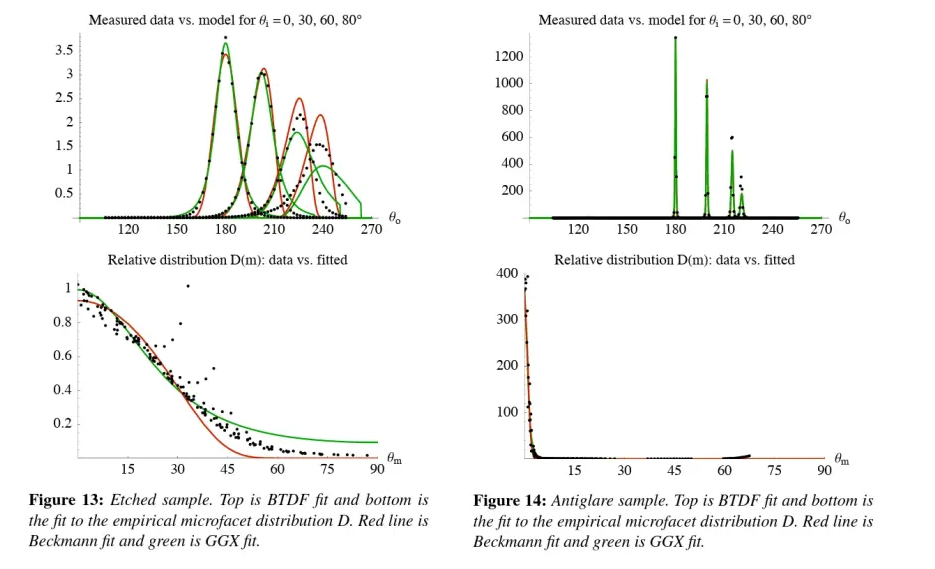
起源 2 的质感更贴合现实。这是 BRDF 物理准确性（其实 BRDF 也不是绝对正确的，这里只是相对于起源 1 的传统模型）的体现。具体细节接下来将讲述。

最初的 Half-Life 2 (2004) 起源 1 使用的是 Phong 着色模型。在起源 1 发展的这接近 20 年内，出现了很多更高级的修改版模型。我们这里以 Specluar 为例。

对起源 1 而言，只要是有 Specular 的地方，都是使用了下述的一项技术：

1977 年，Blinn 提出了一种叫 Blinn-Phong Normal Distribution Function (NDF) 的法线分布函数。（NDF 用于确定反射高光的具体表现）

Blinn-Phong NDF 只是对 Phong 模型 进行的修改，这意味着它并非是 Physically-Based 的. 对此，您一定想明白了在材质真实性上，起源 1 被起源 2 薄纱的原因。起源 2 的 PBR 系统使用的 Trowbridge-Reitz NDF (GGX Distribution)，具体原理可参见 [11]，对于其它的 例如拉姆达函数等对这文章目的没有用处的内容将不在此展开。我在这里只进行了不严谨的简单研究。



其中材质粗糙度由  控制。为了体现起源 2 的优势，下面给出的几组数据 Walter et al. 显示了作为 GGX 法线分布函数（绿色）和 真实采集的数据（离散的黑点）的比较：

根据以上数据，我们不难发现 GGX 分布非常贴合真实。这只是起源 2 材质系统的冰山一角，毕竟就举了一个 BRDF 表面反射模型的例子。这里我们也不详细展开解法，其实前面也已经给了您一些提示。（关系 I）

起源 2 引擎在材质渲染上相比起源 1 是降维打击的。

GGX 分布函数原先广泛用于电影工业领域，随着个人计算机算力的大幅提升，游戏领域也开始逐渐引入。但这个模型实际上在今天的实时渲染领域非常常见，虚幻 4 / 5，寒霜等耳熟能详的游戏引擎都在使用。

您可能会产生疑问：为什么我要这么大篇幅地分析起源 2 使用的这些“普通的”技术呢？我觉得，这可能就是 V 社的魅力所在吧...（说难听点就是起源 1 太落后了）

说到这里，您一定能体会到起源 2 引擎相较于起源 1 的强大了——它真正地步入了现代。

. 起源 2 图形的案例分析（Case Study）

这里使用 NVIDIA Nsight GPU Trace 图形分析工具进行数据采集和分析。此案例分析主要集中于对起源 2（2023）的底层优化方面。

我分别选取了 Counter-Strike 2 几个比较具有代表性的场景：1. 有光源 2. 开阔场景 3. 高帧率的简单场景，由于发出 Nsight 截图太花了不太美观，这里我提炼了最有价值的信息并用文字表述。需要 Nsight 格式源文件的可以在我的网盘 bluemicro.ys168.com - 计算机图形资料 -Desktop.7z

我们分析出 CS2 计算一帧需要 1500 - 2000 个 Drawcall. 作为对比，GTA 5 每绘制一帧平均 1900 个 Drawcall. 不过这和 CS2 处于内测阶段也有关系，游戏优化一般是放在最后做的。

对于起源 2 的游戏来讲，我们发现整个 GPU 的瓶颈实际上是内存子系统，并不是计算单元。实际上现在很多游戏都是如此，游戏建模和材质都比较庞大，对内存带宽的要求非常高。

起源 2 本身不是特别注意实时渲染，通过开发机离线算好了的数据很多（比如全局光照），所以计算单元吞吐较少是正常的，捕获到的几个计算单元的空闲时刻也情有可原。

对于管道延迟, L1 的问题，这和起源 2 的 Forward+ 渲染模式有关。因为 Tile-Based 模式的渲染需要等待生成 Tile 表才能进行进一步光栅化操作，这会造成整个管线的一些延迟；并且 Tile 表存放在 SRAM 里，对内存吞吐量也有明显的推动，这可能也是几组数据显示内存子系统吞吐量占比很高的原因。

目前，PC GPU 架构设计走向正在趋向于双发射 FP32 的设计，实际上就是为实时光线追踪铺路。因为在目前的光线追踪管线（见 附录 A）的大量计算操作仍然在 Compute Shader 上（也就是“传统单元”）完成。

为什么提双发射 FP32 GPU？因为光线追踪 shader 以及 denoise 就严重依赖于 FP32 算力。所以以后 V 社为 Hammer 集成了硬件加速光线追踪的地图编译功能时，我们使用这一类 GPU 将有效提高速度。例如 AMD RDNA 3, NVIDIA Ampere / Ada 架构的 GPU，它们都拥有包含 2x FP32 执行的 ALU.

4. 起源 2 的虚拟现实

起源 2 引擎在 VR 上的强大表现，相信您已经在 半条命：爱莉克斯 中目睹过了。

对于 VR 性能的优化，最直接的做法就是降渲染分辨率。NVIDIA 有 multi-resolution shading，AMD 有 variable-rate shading. V 社的核心思路是将屏幕分割成 3x3 的区域，对于聚焦区域可以进行全分辨率渲染，其它部分适当降低分辨率。

V 社顶级图形工程师 Alex Vlachos 在 GDC 2016 的 "Advanced VR Rendering Performance" 会议上会为您提供最详细准确的 VR 优化方案。

5. Source 2 SDK 的一些改进技术

·方便快速的包围盒生成

起源 2 的 ModelDoc 集成了一种叫 QuickHull [15] 的算法，可以根据模型信息来自动创建包围体积，这对于快速生成碰撞箱是很有用的。值得注意的一点是，虽然它的名字带 “Quick”，但它仍然是离线处理的。

·VRAD 改进

我们知道，起源 1 的光照烘焙器 VRAD 是基于辐射度（Radiosity）算法的。这是一种和光线追踪对立的，实现全局光照的一种有限元法，它将场景分成若干个面元 (Patch) 进行计算。

来自 VRAD - Valve Developer Community

事实上，两者都是以解渲染方程（The Rendering Equation）作为指导思想。何为渲染方程？见 附录 D

对于起源 1 VRAD 来讲，辐射度算法可以高效地算出无限次的漫反射反弹。

这个表示辐射度的方程是由渲染方程得来的，起源 1 VRAD 通过将此方程转化为线性方程组进行求解。其中， 代表辐射出射度（自发光）， 代表面元  向外辐射的能量， 代表形状因子， 代表反射系数，是一个常量。（其实就是 Reflectivity, 天天被 Hammer 雷普的人应该很熟悉这个词）

不仅是全局光照，起源 1 基于法线贴图的自阴影和 AO 效果也是通过辐射度算法 [18] 实现的。被称为 “Half-Life 2 Basis”.

然而，随着时代的发展，游戏场景和光源的复杂度提高，该方法的缺点也被逐渐放大。首先，形状因子 本身就是很难计算的；VRAD 就是要解非常大一坨积分，使用半立方体算法结合 Z-buffer 求解。（速度最快）

需要注意的一点是，形状因子也可以使用蒙特卡洛 (Monte Carlo) 方法 (附录 E) 求解。

如果场景复杂了起来，那么需要求解的方程组会变得极其庞大，很难求解。体现在起源 1 的 Hammer 地图制作上，则是超低效率的最终编译。

起源 2 的 VRAD 3 更换了 Intel Embree Raytracing Library，算法上直接从 Radiosity 跳到了 Raytracing. 由于实现原理上的不同，所以上述的问题就都消失了。不过，目前 VRAD 3 使用的 "Raytracing" 是一种类似于光子映射 (Photon Mapping) 的方法，因为光线是从光源射出而不是从特定视角。

目前 VRAD 3 最大的缺点是仍不支持 GPU 加速的光线追踪。但请别感到悲哀，现在 V 社和 Facepunch 都在开发 VRAD 的硬件加速功能。目前 Intel Embree 最新版本身已经实现了对 Intel Xe-HPG GPU 架构的硬件加速支持。

值得注意的是，Intel Embree 的光线追踪管线和附录里的差别很大，借用了很多光栅化管线的思路。被 Imagination 定义为 "Level 0 Legacy Ray Tracing" —— 0 级史前时代光追。

·Hammer 实时光照预览

起源 2 的 Hammer 2 支持了实时的光照预览（类似起源 1 Hammer++ 的效果），但是只有直接光照，目前来说，要看全局光照的效果一样得等待光照烘焙。不过对于常年被起源 1 Hammer 折磨的地图制作者们来说，Hammer 2 绝对是非常人性化的。

Conclusion

这篇文章我们深入讨论了起源 2 以及起源 1 部分的图形底层，突出了起源 2 的图形优势。看到这里，我相信您已经对起源 2 的图形方面了解的比较透彻了。V 社是久负盛名的，以高技术力著称的顶级游戏公司，相信将来完全体的起源 2 ，会在那款游戏上展现出真正次世代游戏引擎的水准。

附录

Appendix A: 光线追踪管线简述

光线追踪管线的行为逻辑很像计算 (compute) 管线，即使是计算卡（没有图形输出功能的 GPU） 也可以运行。它通过 Ray Generation Shader 使用 TraceRay() 生成光线来启动整个管线。在光线和场景内三角形判断求交（通过 Intersection Shader 判断）后即可计算颜色（击中调用 Closest Hit Shader, 未击中则调用 Miss Shader 来计算颜色），最终的颜色便是我们将要输出的结果。

光线追踪管线内还有一个可选的 Shader 叫 "Anyhit"，主要用于具有 Alpha 材质的场景。系统根据调用 Anyhit Shader 判断半透明场景是否继续跟踪光线。

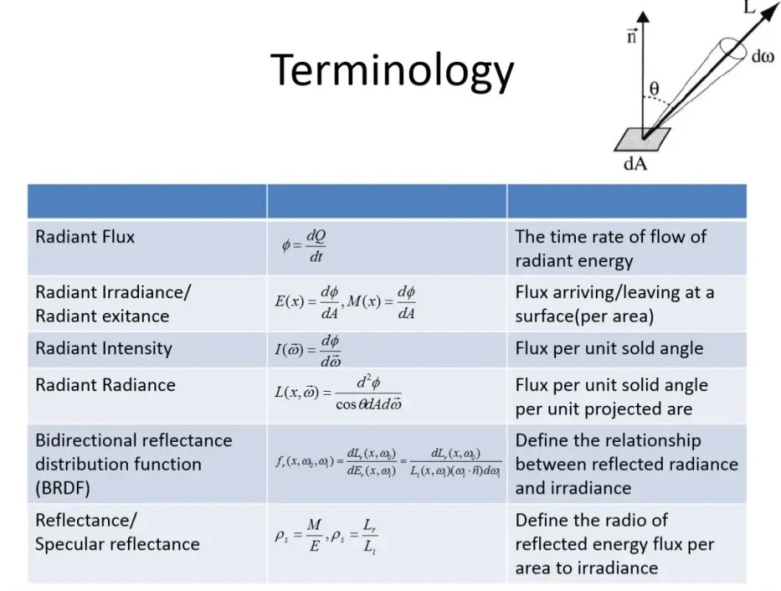
光线追踪是一个集合，例如递归光线追踪 (Whitted-style Ray-Tracing), 路径跟踪 (Path-Tracing) 等都属于光线追踪。Whitted-Style 光线追踪只能生成锐利的阴影，反射信息，无法做 Diffuse, Glossy texture，很难达到 Photo-Realistic 水准。所以现在能见到的光线追踪绝大多数都是路径跟踪。

Appendix B：现代 PC GPU 基本渲染结构进化

之前————NVIDIA Maxwell (GTX 9x0) 架构————现在

传统 IMR ———— 首次在桌面 GPU 上引入 Tile-based 模式 ———— 硬件上的 "TBIMR" (Tile-based immediate mode)，和移动端的 TBR 是有区别的，详情见 [4].

Appendix C：双向反射分布函数（BRDF）



Appendix D: The Rendering Equation (渲染方程)

渲染方程是计算机图形学中真实渲染的数学表达，Kajiya 在 1986 年的图形学论文 "The Rendering Equation" 提出了这个概念。与它以前的诸多经验模型不同的是，渲染方程是物理正确的，让计算机图形正确地模拟世界成为了可能。

具体细节可搜索此论文 [16] ，受限于附录篇幅我们不详细展开。

Appendix E: Monte Carlo method

蒙特卡洛方法是一种通过随机抽样（或模拟随机变量）解决数学问题的数值方法，由冯·诺依曼 (John von Neumann) 等人于第二次世界大战时期提出，用于研制原子弹的曼哈顿计划。图形渲染中需要大量快速的积分运算，所以蒙特卡洛方法是一种被广泛被图形学界使用的求解积分的数值方法。例如实时光线追踪求解渲染方程就使用了蒙特卡洛方法。

蒙特卡洛方法具有严格的大数定律数学背景，并不是瞎几把猜值。根据伯努利 (Bernoulli) 大数定律可知，随着样本数增加，频率会收敛于概率，重复的试验越多误差就会越低。这也一定程度上解释了为什么做全局光照采样率越高，画面越准确。

References

[1] NVIDIA Corp. "The GeForce 6 Series GPU Architecture", GPU Gems 2 , 2005.

[2] NVIDIA Corp. "NVIDIA Ada GPU Architecture" , white paper, 2022.

[3] NVIDIA Corp. "NVIDIA Turing GPU Architecture", white paper, 2018.

[4] NVIDIA Corp. "NVIDIA GeForce GTX 980 - Featuring Maxwell, The Most Advanced GPU Ever Made.", white paper, 2014.

[5] Velho, L. ,  V. Silva , and  T. Brito . "Immersive Visualization of the Classical Non-Euclidean Spaces using Real-Time Ray Tracing in VR." Graphics Interface 2020, 2020.

[6] Valve Developer Community, "Source 2", developer.valvesoftware.com, 2023.

[7] Jeremiah, "Forward vs Deferred vs Forward+ Rendering with DirectX 11", 3D Game Engine Programming, 3dgep.com, 2015.

[8] Olsson, O. ,  M. Billeter , and  E. Persson . "Efficient Real-Time Shading with Many Lights." International Conference on Computer Graphics and Interactive Techniques, ACM Press, 2014.

[9] Markus Billeter, Ola Olsson, and Ulf Assarsson. Tiled forward shading. GPU Pro 4: Advanced Rendering Techniques, 2013.

[10] Tile Base Render (Forward+) , zhuanlan.zhihu.com/p/553907076, 2022.

[11] Bruce Walter, et al. "Microfacet models for refraction through rough surfaces." Proceedings of the 18th Eurographics conference on Rendering Techniques. Eurographics Association, 2007.

[12] Blinn, J. F. . "Models of light reflection for computer synthesized pictures." , ACM Computer Graphics (SIGGRAPH '77 Proceedings), vol. 11, no. 2, pp. 192-198, July 1977.

[13] Phong, B. T. . "Illumination for Computer Generated Pictures." Communications of the ACM 18.6(1998):311-317.

[14] Whitted, and Turner. "An improved illumination model for shaded display." ACM, ACM 1979:14.

[15] Gregorius, Dirk, "Implementing QuickHull", Game Developers Conference, Mar. 2014.

[16] Kajiya, and T. James . "The rendering equation." ACM Computer Graphics 20.4(1986):143-150.

[17] Goral, C. M. , et al. "Modeling the interaction of light between diffuse surfaces." Acm Siggraph Computer Graphics 18.3(1984).

[18] Green, C. . "Efficient self-shadowed radiosity normal mapping." ACM, 2007.

[19] Tomas A-M., et al. "Real Time Rendering Fourth Edition", CRC Press, 2018.

[20] Hammersley, J. M. , and  D. C. Handscomb . Monte Carlo Methods. Methuen & Co Ltd, 1964. 作者：B1ueMicr0 https://www.bilibili.com/read/cv23384330?spm\_id\_from=333.999.list.card\_opus.click 出处：bilibili