

**研 究 生 课 程 论 文**

****

|  |  |
| --- | --- |
| **题目：** | **光线追踪发展概述** |
| **学号：** | **22251037** |
| **姓名：** | **李龙** |
| **专业：** | **软件工程** |
| **学院：** | **软件学院** |
| **课程：** | **三维动画与交互技术** |

**摘要：**光线追踪是一种产生逼真图像的方法;它在像素级上确定图像中的可见表面而与z-buffer和BSP树不同的是，光线跟踪是逐像素操作，而不是逐原语操作。这使得在处理屏幕空间中有大型物体的场景时，光线追踪的处理作用相对缓慢。但是，它有许多不错的特性，这些特性通常使它成为离线渲染呈现甚至某些交互式应用程序的正确选择。本文将概述光线追踪的发展历史，并简述其的一些加速算法。

1. 引言

可见性算法是渲染技术的重要组成部分，也是计算机图形学方面一直在研究的热点问题。例如，为了生成一张图片，必须确定这些图片上的哪些点在阴影里，哪些点在光照下，而它们又各自是什么颜色。而如果要考虑全局光照的话，我们又要确定哪些表面可以因为接受了其它表面的光使它变得可见。解决可见性问题最常用的两种方法是基于光栅化的方法和基于光线跟踪的方法。这种可见性算法的性能对于交互式应用程序是至关重要的。

光线追踪是计算机图形学中用来解决可见性问题的最重要的技术之一。许多用其他技术很难或不可能的现象用光线跟踪技术就很简单了，包括阴影、反射和折射光。但是其也因为计算量大往往只被应用于离线渲染上。近几年，GPU发展迅速，光线追踪算法也在不断地改进，使得有些显卡和交互式游戏可以支持光线追踪算法进行实时的渲染，这是光线追踪算法应用的一大突破，本文将简要介绍一下光线追踪算法的发展和一些关于光线追踪的加速算法。

1. 光线追踪算法发展
   1. 光线投射算法

最早的光线追踪算法要啊追溯到Arthur Appel于1968年提出的光线投射算法。这种算法有些类似z-buffer和BSP-tree算法，都是逐像素查询其对应的颜色，并将其显示在屏幕上。但是它的方法是从眼睛处向每一个像素投射出一道光线，然后看这个光线与画面中的哪个物体相接触，然后将那个点的颜色显示在对应的像素点上。这时的光线投射算法并没有考虑到光的反射、折射，本质上也是逐像素直接取用对应点的颜色值，这其实是跟光栅化很相似的。

* 1. 光线追踪算法

光线投射算法之后有很多人对其进行了完善，如布林冯模型，它在之前光线投射算法的基础上更加考虑了光源的位置和法线的方向，综合了漫反射光、环境光和镜面反射光，建立了一个更加真实的光照模型。

而我们更为熟知的光线追踪算法，是由Whitted于1979年提出，这对于业界是一次革新性的技术进步。该算法在结合了布林冯模型的基础上发现了光线追踪的递归性质，并提出了一个递归的算法，即当光线打到一个物体上时，那个物体就成为了一个新的摄像机，并由折射和反射发出新的光线进行光线追踪，如此递归，直到最大深度。这种递归的方法很好的模拟了真实场景中多次反射的情况，解决了传统布林冯模型在镜面反射上的糟糕表现，并且可以帮助我们实现全局光照，这是业界的一次巨大进步。

同时，光线追踪算法很难模拟出那种软阴影，因为它的光线是通过几何严格计算，这其实并不符合自然界漫反射的现象，这导致了其只能呈现出并不具备真实感的硬阴影。要想实现软阴影，必须得模拟漫反射现象，每次递归时要多反射几道邻近的光线，但是这又会导致开销的指数爆照。对此，Cook等人于1984年提出了一种分布式光线追踪方法解决了这个问题，这种方法采用了蒙特卡洛的思想，对每一个像素投射多道光线，每道光线打到物体后产生的新的反射光线并不是一致的，而是从某种分布中采集，最后取平均值，这种方法很好地避免了指数爆炸的情况，在不多的开销下解决了金属光泽、软阴影、景深、运动模糊等问题。

* 1. 路径追踪算法

在前人的研究基础上，Kajiya于1986年进一步建立了渲染方程的理论，并使用它来解释光能传输的产生的各种现象。这一方程描述了场景中光能传输达到稳定状态以后，物体表面某个点在某个方向上的辐射率与入射辐射亮度等的关系。这个用物理学来模拟光的传播的理论方程成为了未来全局光照模型发展的理论基础。

同时，他也在论文中提出了路径追踪算法，这开创了基于蒙特卡洛的全局光照这一方向领域。根据渲染方程，Kajiya 提出的路径追踪方法是第一个无偏的渲染方法。路径追踪的基本思想是从视点发出一条光线，光线与物体表面相交时根据表面的材质属性继续采样一个方向，发出另一条光线，如此迭代，直到光线打到光源上（或逃逸出场景），然后用蒙特卡洛的方法，计算其贡献，作为像素的颜色值。

但是这种方法有个问题就是，因为光源很小，所以很多光线其实打不到光源上，这就很难产生一条有效路径，于是Veach等人于1995年提出采用双向路径追踪的方式，同时从视点、光源打出射线，经过若干次反弹后，将视点子路径和光源子路径上的顶点连接起来（连接时需要测试可见性），以快速产生很多路径的方法。这种方法能够产生一些传统路径追踪难以采样到的光路，所以能够很有效地降低噪声。

虽然双向路径追踪算法相比最原始的路径追踪算法已经有了很强的效率提升，但是对于复杂场景的表现力仍显不足。而想要提升路径追踪算法效率的关键在于找到有效路径的效率。于是Veach 等人于1997年提出了一种新的蒙特卡洛采样方法——MLT方法，其核心思想在于，一旦发现对结果影响更大的有效路径后，MLT采样方法就会在路径空间中该路径临近的区域继续采样，使得得到的路径对结果的影响较大的可能性更大。即可以自适应的生成贡献大的路径，简单来说它会避开贡献小的路径，而在贡献大的路径附近做更多局部的探索，通过特殊的变异方法，生成一些新的路径，这些局部的路径的贡献往往也很高。与双向路径追踪相比MLT更加鲁棒，能处理各种复杂的场景。比如说整个场景只通过门缝透进来的间接光照亮，此时传统的路径追踪方法因为难以采样到透过门缝的这样的特殊路径而产生非常大的噪声，而MLT采样方法就能很好地解决这一问题。

1. 光线追踪加速算法
   1. 参数化加速

Arthur Appel的光线投射算法，根据他的说法，光线投射算法有大约一半的时间消耗用在确定投影与场景的点对点对应，也就是说大约一半时间都消耗在求交上。而到后面whitted的光线追踪算法里，这个问题并没有被解决，反而扩大了光线追踪算法开销巨大的问题。其在论文里的简单场景实现里，有 13% 的时间占比属于额外开销，求交时间占比是 75%，着色时间占比是 12%。而根据作者的说法，对于更复杂的场景，射线与表面求交计算所需的计算时间占比会超过 95%。因此，很多人对于如何加速光线追踪进行很多的优化。

如Smyrl等人于1989年提出了一种参数化光线追踪的算法，即当场景内的几何形体没有改变，仅仅改变了光源的位置和角度时，参数化表示每个像素像素值的参数表达式，并将其存储起来，当光源的颜色和位置发生改变时我们并不需要重新进行渲染，只需要改变一下像素值的参数就行。

* 1. 简化运算加速

上述参数化加速方法没有本质上解决光线追踪的巨大开销，我们可以发现，光线追踪的开销主要是用于光线与物体的求交上，一条光线要跟每个物体都进行一次求交运算，而其实大部分物体光线都是打不到的，这就导致了巨大的浪费，于是Whitted引进了包围盒技术来加速这个求交过程。

包围盒技术（BVH）是指场景中的物体可以被一个规则的盒子包围住，这个盒子在光线追踪的过程中，用于代替物体本身来与光线进行求交测试，如果光线与包围盒都不相交，那么光线就更加不可能与包围盒里的物体有交点；如果光线与包围盒相交，再对光线与包围盒里的物体的三角形面片进一步进行求交计算。这种技术减少进行相交测试的数目，降低复杂度，进一步得到效率上的提升。在包围盒的选取中，一方面应该选择形状简单的包围盒，降低光线与包围盒相交测试的代价；另一方面应该选取能够紧密包围物体面片的包围盒，提高光线与包围盒进行相交测试的有效性，如使用KD树等存储结构。

通过 BVH 结构，可以将三角形遍历的次数显著减少。例如哈佛大学的弥勒佛，它的三角形规模大约是 100 万，使用 BVH 的话，BVH 4 下的平均节点深度可能也就是 10 级，也就是大约遍历 10 次 BVH 后，就能抓出被射线击中的三角形。相对于原来的 100 万三角形而言，相当于降低了 5 个数量级或者说 1/100000。

* 1. 并行化加速

另外还有利用并行运算来加速光线追踪的算法。光线追踪算法对每个像素的每个样本执行相同的算法，然而，在非相干射线的跟踪阶段找到最近的交集会导致线程和数据访问的分歧，这大大降低了吞吐量（Wald et al 2003）。

处理这个问题的一种可能是显式地生成相干射线集。为此，Szirmay-Kalos和Purgathofer于1998年提出了全局射线束跟踪;以及最近的Nimier-David等人（2019）采用了相干MCMC采样。这些方法可以以最小的开销生成高度一致的工作负载，这就使得光线追踪的并行效率更高。

Pharr 等人于1997年提出了一种算法，，即将每次产生的光线进行排序，令所有光线只针对场景的一个子集进行测试，以提高缓存一致性。虽然这对于非常大的场景是可行的，但重复加载射线数据会显著增加内存带宽。

Gunther等人于2007年提出了一种包遍历的方法，通过强制对一组射线进行SIMD处理来利用存在相干性的射线，这增加了线程间的通信和同步，同时它假设高相干性射线显著慢于深度优先遍历非相干射线。Bikker于2012年提出了一种另包遍历算法，该算法使用批处理来改善数据的局域性。

目前最先进的基于gpu的全局照明框架是基于将工作拆分为计算光线与场景(跟踪内核)的交集和阴影（Laine et al 2013）。事实上，在跟踪内核中处理光线的顺序可以独立于阴影，这使得光线排序或重新排序方法可行。

在GPU上发生线程发散的情况下，整个线程被阻塞，直到它的所有射线完成遍历。Aila和Laine于2009年提出通过用全局队列中的新射线替换已经完成的射线来提高SIMD效率。推测遍历等技术稍微增加了射线相交测试的冗余，因为它们工作在可能终止的射线上。

但这种冗余只是虚拟的，否则核心将处于空闲状态。Boulos等人于2008年提出了一种CPU上SIMD射线跟踪的包重排序方法。类似地，线程压缩算法（van Antwerpen 2011;Wald 2011），去除空二次射线，被提出用于GPU射线跟踪。虽然经处理的翘曲数减少了5倍，但由于发散增加，性能仅增加了16%。

Garanzha和Loop（2010）在射线排序步骤后使用宽度优先包遍历。他们提出了对射线进行排序的想法，使用基于哈希的方法将射线排序为连贯的数据包，以减少计算中的分歧。

在分析gpu上光线追踪的效率时，Aila和Laine（2009）也评估了基于交错光线原点和归一化光线方向的哈希排序准则。当时，排序开销太大，无法改善整体呈现时间。Moon等人使用了另一种针对核外渲染的寻址缓存相干存储器访问的光线重排序方案（2010）。他们建议使用一个估计的终止点来对光线进行排序，这个终止点是通过光线追踪一个适合主存储器的简化场景来计算的。然而，由于昂贵的命中点估计，该方法只适用于核外射线追踪。

最近，Costa等人于2015年提出了一种用于环境遮挡的光线排序方法。然而，它们的排序准则主要基于射线方向，适用于方向取决于原点的短射线或阴影射线。

1. 实时光线追踪

实时渲染最大的困难是，要渲染的场景是动态的，渲染器通常不知道场景将如何变化，因为这可能会受到用户交互的影响。所以在过去几十年里，光线追踪主要被应用于离线渲染上，因为其开销实在过于巨大，渲染速度无法跟上实时渲染的要求，直到最近引入硬件光线追踪(HWRT)，实时渲染才能够解决每帧的许多光线查询，允许在GPU上遍历光线追踪加速结构（Keller et al 2019），这才使实时光线追踪变为了可能，之后大家在硬件跟算法的基础上不断改善革新实时光线追踪技术，使得实时光线追踪被应用到了更多的交互式场景当中。

但是GPU只能以实时帧速率跟踪每个像素的几条光线，因此有必要将采样集中在重要的光源上。Moreau等人于2019年提出了一种实时光线跟踪的动态多光采样算法来维护以实时渲染为目标的分层光采样数据结构。他们的方法基于两级BVH层次结构，降低了部分层次结构更新的成本。通过重新安装更新较低级别的BVH，保持其原始拓扑结构，进一步提高了性能。结果表明，这种方法在每帧从头开始重新创建整个层次结构时的误差在6%以内，同时速度快了两个数量级，对于现代GPU上有数千个移动光源的场景，每帧需要不到1ms的层次结构更新时间。此外，作者还证明，通过时空过滤，他们的方法允许使用每帧16.1ms的光线跟踪阴影渲染具有数千个灯光的复杂场景。

Bitterli等人则（2020）介绍了一种新的蒙特卡罗方法，基于重新采样的重要性抽样的推广直接照明。它允许无偏的空间和时间重用附近的样本，并导致更有效的偏置变体。与以前的方法相比，他们的算法在误差上降低了一到两个数量级，同时还只需要简单的图像空间数据结构。已经证明了这个算法适用于高性能GPU实现，可以实时渲染具有成千上万个动态光源的场景。

由于路径跟踪渲染需要大量时间，而动态投影映射需要高帧率渲染，例如500-1000 fps。为了克服这一差距，Nomoto等人（2020）利用了视觉持久性的概念。在动态场景中，通过以947fps的速度投影路径跟踪的噪声图像，对观察者进行感知上的整合。再现的图像可以在低延迟下与物理运动和观察者视点一致，同时高度逼真。

在虚拟现实的光线追踪上也有着这样的简化算法，如Yang 等人于2020年提出了一种可变速率射线跟踪的算法，通过控制每个像素的跟踪率，在图像质量下降最小的情况下，有效地提高了VR应用中的实时路径跟踪性能。基于此评价，VRRT能够在相同的性能预算下提供更有效的样本射线计数，从而提高渲染场景的图形质量，呈现逼真的虚拟环境。基于基准测试，该方法在当前主流图形硬件和虚拟现实设备上可以提高30%以上的帧/秒(FPS)。

1. 结语

光线追踪是计算机图形学的基石之一，它的优点在于能够很好地模拟真实的物理情况，对于处理反射、折射、阴影和光线的多次弹射都有着很好的表现。但是它最大的缺点就是性能，因为其递归的性质，每次打到一个物体就要发射出新的光线进行独立的运算，这就使得它在生成高质量图像的同时带来了巨大的开销，这往往令它的计算速度跟不上实时渲染的要求。

由于近几年GPU的快速发展，越来越多的显卡以及游戏都开始支持用光线追踪来进行实时渲染，同时VR领域里光线追踪的应用也是如火如荼的发展，我相信在未来光线追踪的发展趋势仍然是着重于加速二字，如何在避免失真的情况下尽可能地减少开销仍然是光线追踪研究的主流方向。因为光线追踪的物理模型目前来看已经足够完善，它也确实能渲染出比光栅化更加真实的现实模型，随着显卡和算法的发展，光线追踪代替光栅化成为实时渲染的主要方法也是一个可以想象的未来。

未来实时光线追踪的一个大方向大概率是对纹理空间技术和可变速率光线追踪的探索。如缓存材质和局部解，以及BRDF拆分。而在高效采样和积分策略以及 重建方面，业界也还有很多事情需要去完成。

**参考文献：**

[1] Turner Whitted. An improved illumination model for shaded display[J]. ACM SIGGRAPH Computer Graphics,1979,13(2).

[2] Appel, A. Some techniques for shading machine renderings of solids. AFIPS 1968 Spring Joint Comptr. Conf., pp. 37~15.

[3] Bui-Tuong Phong. Illumination for computer generated images.Comm. ACM 18, 6 (June 1975), 311-317.

[4] Robert L. Cook,Thomas Porter,Loren Carpenter. Distributed ray tracing[J]. ACM SIGGRAPH Computer Graphics,1984,18(3).

[5] Cook, Robert L., "A Reflection Model for Computer Graphics," ACM Transactions on Graphics, vol. 1, no. 1, pp. January 1982.

[6] James T. Kajiya. The rendering equation[J]. ACM SIGGRAPH Computer Graphics,1986,20(4).

[7] C. H. Séquin,E. K. Smyrl. Parameterized Ray-tracing[J]. ACM SIGGRAPH Computer Graphics,1989,23(3).

[8] Veach and L. J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In SIGGRAPH ’95. 419–428.

[9] Veach and L. J. Guibas. Metropolis light transport[P]. Computer graphics and interactive techniques,1997.

[10] Ingo Wald, Timothy J. Purcell, Joerg Schmittler, Carsten Benthin, and Philipp Slusallek.

2003. Realtime Ray Tracing and Its Use for Interactive Global Illumination. Eurographics State of the Art Reports.

[11] Laszlo Szirmay-Kalos and Werner Purgathofer. 1998. Global Ray-Bundle Tracing with Infinite Number of Rays. Computers & Graphics 23, 2 (1998), 193–202.

[12] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. ACM Transactions on Graphics 38, 6 (2019), 203.

[13] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. 1997. Rendering Complex Scenes with Memory-coherent Ray Tracing. In Proceedings of International Conference on Computer Graphics and Interactive Techniques. 101–108.

[14] Ingo Wald, Christiaan P. Gribble, Solomon Boulos, and Andrew Kensler. 2007. SIMD Ray Stream Tracing - SIMD Ray Traversal with Generalized Ray Packets and On-the-fly Re-Ordering. In Technical Report UUSCI-2007-012.

[15] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. 2007. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. In Proceedings of IEEE Symposium on Interactive Ray Tracing. 113–118.

[16] Timo Aila and Samuli Laine. 2009. Understanding the Efficiency of Ray Traversal on GPUs. In Proceedings of High Performance Graphics. 145–149.

[17] Singh, Jag Mohan and Wasnik, Pankaj and Ramachandra. Hessian-Based Robust Ray-Tracing of Implicit Surfaces on GPU. SIGGRAPH Asia 2018 Technical Briefs.2018,16（4）.

[18] Yang, Jinyuan and Li, Xiaoli and Campbell, Abraham G. Variable Rate Ray Tracing for Virtual Reality. SIGGRAPH Asia 2020 Posters.2020,19(2).

[19] Gruen, Holger and Benthin, Carsten and Woop, Sven. Sub-Triangle Opacity Masks for Faster Ray Tracing of Transparent Objects. Proceedings of the ACM on Computer Graphics and Interactive Techniques.2020,3(2).

[20] Moreau, P. and Pharr, M. and Clarberg, P.. Dynamic Many-Light Sampling for Real-Time Ray Tracing. Proceedings of the Conference on High-Performance Graphics.2022,16-22.

[21] Nomoto, Takashi and Koishihara, Ryo and Watanabe, Yoshihiro. Realistic Dynamic Projection Mapping Using Real-Time Ray Tracing. ACM SIGGRAPH 2020 Emerging Technologies.2020,13(2).

[22] Bartels, Pieterjan and Harada, Takahiro. Combining GPU Tracing Methods within a Single Ray Query. SIGGRAPH Asia 2022 Technical Communications.2022,17(4).

[23] Bitterli, Benedikt and Wyman, Chris and Pharr, Matt and Shirley, Peter and Lefohn, Aaron and Jarosz, Wojciech. Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting. ACM Transactions on Graphics.2020,148（17）.