基于流式路径追踪的实时真实感渲染技术

Streaming Path Tracing for Real-Time Realistic Rendering Technology

王宸/WANG Chen，过洁/GUO Jie，郭延文/GUO Yanwen
（南京大学，中国 南京 210023）
(Nanjing University, Nanjing 210023, China)

DOI：10.12142/ZTETJ.2024S1008
网络出版地址：http://kns.cnki.net/kcms/detail/34.1228.tn.20240724.1131.016.html
网络出版日期：2024-07-24
收稿日期：2023-11-20

摘要：从图形处理器（GPU）的线程调度和内存访问两个角度出发，提出了一种基于流式路径追踪的实时真实感渲染方案。它将传统的路径追踪算法分解成多个独立的逻辑模块，使得算法的实现更加贴合GPU硬件的调度模式，并且使用数组结构体（SoA）风格的内存布局重新排列数据，使得算法在GPU中运行可以减少对动态随机存取存储器（DRAM）的访问次数，从而提升算法的运行性能。该方案对GPU友好，相比于无优化的路径追踪算法，在GPU中的运行时间降低了83%~88%。

关键词：实时真实感渲染；路径追踪；GPU友好；流式；SoA

Abstract: A streaming path tracing for real-time realistic rendering by examining the graphics processing unit (GPU) thread scheduling and memory access is presented. Traditional path tracing algorithms are decomposed into several independent logic modules, making the implementation more in line with GPU hardware´s scheduling pattern. Furthermore, using a structure of arrays (SoA) style memory layout rearranges data to reduce the number of accesses to dynamic random access memory (DRAM) when algorithms run on the GPU, thereby improving performance. The proposed scheme is GPU-friendly, showing a decrease in the runtime by approximately 83% to 88% on the GPU compared to the unoptimized path tracing algorithm.

Keywords: real-time realistic rendering; path tracing; GPU-friendly; streaming; SoA

引用格式：王宸，过洁，郭延文．基于流式路径追踪的实时真实感渲染技术 [J]. 中兴通讯技术，2024，30(S1)：54-59. DOI：10.12142/ZTETJ.2024S1008

Citation：WANG C, GUO J, GUO Y W. Streaming path tracing for real-time realistic rendering technology [J]. ZTE technology journal, 2024, 30(S1): 54-59. DOI: 10.12142/ZTETJ.2024S1008

光线追踪技术是一种用来产生真实感光照效果的图形渲染技术。这种技术由于需要极高的计算代价，通常被用于离线的图形渲染任务中，如：高端的视觉特效或三维动画制作。在传统的实时渲染任务中，三维场景通常使用光栅化方法进行渲染，将三维场景转换为二维图像，但这种方法很难准确模拟光线在真实世界中复杂的反射、折射以及散射现象。而光线追踪是一类用于追踪光线在三维场景中传播的方法，由于其直接模拟物理世界中光线传播的现象，因此能够捕捉到更为真实的光照效果。

近年来，随着显卡技术的更新迭代，通用图形处理器（GPGPU）[1]发展迅速，并且诞生了针对光线追踪加速优化的硬件单元RT Core。得益于此，部分光线追踪技术已经可以在游戏和其他实时应用中落地使用，如：基于光线追踪的环境光遮蔽、基于光线追踪的阴影等，一定程度上提高了实时渲染结果的真实感。尽管如此，离线渲染中的基于物理的无偏光线追踪渲染算法，例如：路径追踪[2]、路径指引[3]等算法依然难以在图形处理器（GPU）实现中达到实时的要求。

本文中，我们将结合GPGPU编程的特点，提出一种GPU友好型的路径追踪实现方法。通过重新调度算法子任务，以减少GPU线程的等待时间，加快线程的内存访问速度，从而达到在实时渲染的要求下仍能保证高真实感的目的。相比于无优化的路径追踪算法，该方案在GPU中的运行时间降低了83%~88%。

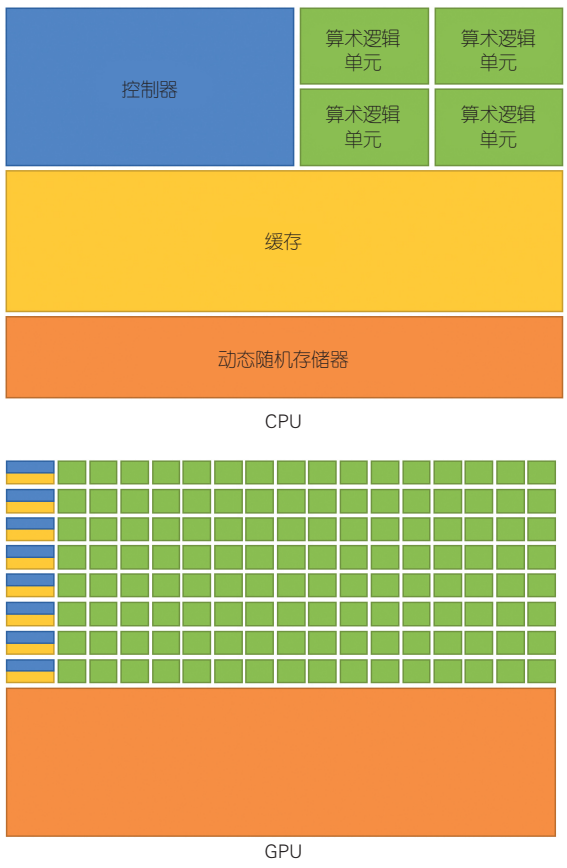
基金项目：国家自然科学基金项目（61972194、62032011）；江苏省自然科学基金项目（BK20211147）

1 GPGPU 编程特点分析

现代显卡编程接口的设计，如统一计算架构（CUDA）、开放计算语言（OpenCL）以及图形管线中的 Computer Shader，便于用户对显卡进行通用编程，而不是局限于传统的光栅化图形管线。与面向中央处理器（CPU）的编程类似，这些编程接口提供了对显存的读写操作，并为并行编程提供了原子操作与同步功能，使得通用的计算任务可以相对方便地移植到 GPU 中实现。同时，GPGPU 编程需要考虑到 GPU 硬件架构与 CPU 的区别。如图 1 所示，GPU 比 CPU 拥有更多的算术逻辑单元和更弱的控制器和缓存机制，导致 GPU 中的线程执行模型和内存访问模式与 CPU 有本质的区别。

1.1 GPU 的线程执行模型

GPU 采用单指令多线程（SIMT）的执行模型，用于组织调度成千上万个线程。由于图形渲染任务中，不同像素的绘制过程通常可以独立执行，很容易地按照像素将渲染任务分割成很多独立的任务并发执行，即可以为每个像素分配独立的线程，同时使用不同的数据执行相同的绘制指令。



▲图1　CPU 和 GPU 硬件架构对比图

在 SIMT 执行模型中，线程会以一定数量（通常，英伟达公司发布的显卡数量为 32，AMD 公司发布的显卡数量为 64）被打包成一组线程束（Warp）。同一组线程束的线程执行相同的指令。当同一组线程束内的线程需要执行不同的分支时，同一分支的线程会继续执行，而不同的分支线程会进入等待状态，从而造成系统性能损失。

与传统基于光栅化的图形渲染算法不同的是，基于物理的路径追踪算法控制分支较多，例如：由于场景复杂的遮挡关系，不同像素的光线弹射次数并不一致，并且不同的材质模型需要选择不同的分支计算。这使得传统的算法实现形式在 GPU 线程中会存在大量的分支等待时间，大大影响了算法的整体性能。

1.2 GPU 的内存访问模式

现代 GPU 具有高带宽的内存系统，而这种高内存带宽通常是以产生相对较长的延迟为代价的[4]。内存访问延迟一般是指指令访问内存到获取返回结果之间的时间间隔。而在 GPU 中，线程执行一次全局内存访问需要消耗几百个时钟周期。因此，考虑线程束的内存访问延迟是十分重要的。

另一方面，GPU 在设计中允许同一时钟周期内，容纳比可并发执行线程数更多的线程，以便一组线程在等待内存访问时，可以调度另一组线程执行，从而隐藏了内存访问时存在的延时情况。延迟隐藏的能力与线程的资源使用情况有关，如寄存器的使用数量。当计算内核使用的寄存器数量越多时，GPU 中可以同时执行的线程数就越少，延迟隐藏的能力就越差。

2 GPU 友好的实时真实感渲染方案

基于上述 GPGPU 编程的特点，本文针对蒙特卡洛路径追踪算法使用流式结构，设计了一套 GPU 友好的算法实现方案，使得算法的运行更贴合单指令多线程的执行模型，提高了线程资源的利用率，同时优化了内存布局以减少全局内存访问的次数，从而减少内存访问带来的延时问题。

2.1 传统路径追踪

传统路径追踪算法通过追踪光线在场景中传播的路径来计算全局光照效果。算法规定每一个像素从相机出发向场景内发射光线，经过若干次反射、折射或者散射，若最终击中光源，则评估该路径对像素颜色值的贡献。物理世界中认为光线可以弹射无数次，直到能量被完全吸收，而算法中通常光线的弹射次数（路径深度）由参数指定，作为算法终止的

条件之一。

在图2中，场景由康奈尔盒子、斯坦福兔子和一个实心小球组成。其中，地面是导体材质，小球为玻璃材质，其余均为漫反射材质。不难看出，随着光线弹射次数的增加，由路径追踪算法渲染的结果中，复杂材质的反射、透射现象逐渐接近真实世界的效果。

传统路径追踪算法在GPU中实现时，如PURCELL等[5]在早期可编程图形硬件上实现的光线追踪算法，将算法的条件分支和循环整体在一个内核程序里实现。这种方法通常被称为大内核方法（Mega-kernel）。在这种形式下，GPU线程的调度是以一个像素的路径追踪流程为单位的。然而，复杂的算法结构会导致内核存在大量的控制分支和内存访问需求。同时，由于渲染场景的复杂性，不同像素发射的光线可能会击中不同类型的材质，并且有些像素发出的光线在采样的过程中可能会提前终止，而另一些像素发出的光线会在场景中经过多次反射或折射，因此光线的路径通常是完全不同的。这使得大内核路径追踪方法中，由SIMT方式调度的GPU线程之间存在大量的互相等待时间。同时，高分辨率的渲染任务通常需要绘制百万甚至千万数量级别的像素，而消费级GPU的计算单元数通常不到一万。这使得同时调度的线程束中，先执行完成的线程束仍在占用线程资源，而等待的任务无法第一时间获得线程的使用权，使得算法效率低下。

2.2 流式路径追踪

为了减少线程资源的浪费，我们选择将整体算法按流程划分为多个指令数少且独立的小内核程序，使得数据可以按照上一个内核程序的执行结果分流到不同的内核程序中继续执行，从数据流的角度减少了算法程序在GPU中执行的条件分支。根据蒙特卡洛路径追踪算法的流程，我们将其分解成以下7个小内核程序：

1）路径初始化：根据相机参数为每个像素生成一条初始光线。

2）求交测试：该内核需要调用光追硬件的求交指令进行光线求交步骤，记录当前光线是否与场景有交点，以及交点的局部信息，如：几何信息和材质信息。场景光线的求交测试结果是路径追踪流程中产生大量分支的原因之一。当光线与场景有交点时，光线会继续追踪下去；而当光线与场景无交点时，该条路径会被终止追踪。流式方法会将产生这两种结果的路径分流到不同的内核中，从而提前释放将要终止路径的线程资源。

3）光线相交处理：与场景有交点的光路会被分流到该内核继续进行计算。若当前光线的方向由出射点的双向散射分布函数（BSDF）采样所得，需要根据BSDF采样的权重计算能量贡献。同时，需要对光源进行采样，得到光路可能的发射方向，并且记录光源的采样信息，同时生成一条阴影测试光线，用于判断光源是否被遮挡。

4）阴影测试：该内核用于在采样光源时，判断被采样的光源光线是否被物体遮挡。该操作同样需要调用光追硬件的求交指令。虽然硬件光追的求交速度很快，但相对来说仍然是算法整体耗时的主要来源，因此尽量避免不必要的求交步骤是很重要的。由于光源采样的概率可能为0，即可能采样到失效样本，并且不需要对该样本计算贡献，因此，采样到失效样本的路径不需要进行阴影测试，可以在光源采样步骤中先筛选出有效样本，再将其统一调度到该内核中调用求交指令。

5）光源贡献计算：通过阴影测试的路径会被分流到该内核，进而继续计算光源的贡献值。

6）材质BSDF采样：需要进行材质BSDF采样步骤的路径与需要进行光线相交处理的路径相同，经过阴影测试和光源贡献计算后的数据流会在该内核汇合。根据当前交点的材质BSDF采样出当前路径的下一次光线弹射方向，同时，由于采样存在失效情况，如：BSDF值为0或样本概率为0，可以丢弃样本失效的路径，以减少无效的线程资源占用。

▲图2 不同光线弹射次数的路径追踪渲染结果图

7）光线丢失处理：求交测试失败的路径会被分流到该内核，即当前光路与场景中没有交点，需要中断该条路径。在该内核中，通常需要计算环境光照对光路能量的贡献。

各个内核的执行流程如图3所示。初始化阶段以像素为单位生成初始光线。当初始光线进入光线追踪循环后，在CPU端，调度程序按照前一个内核的数据分流执行计算内核。分流的计算内核可以并发执行，而合流的内核需要等待所有上一步内核的计算完成。在光源贡献计算结束且光线追踪的弹射次数达到规定的数值后，即可停止光线追踪，将结果记录到内存后进行后处理并输出。此外，当计算资源足够多时，即可以调度的GPU线程数远大于像素数时，可以流水线式执行图3所示流程，进一步提高并发度。

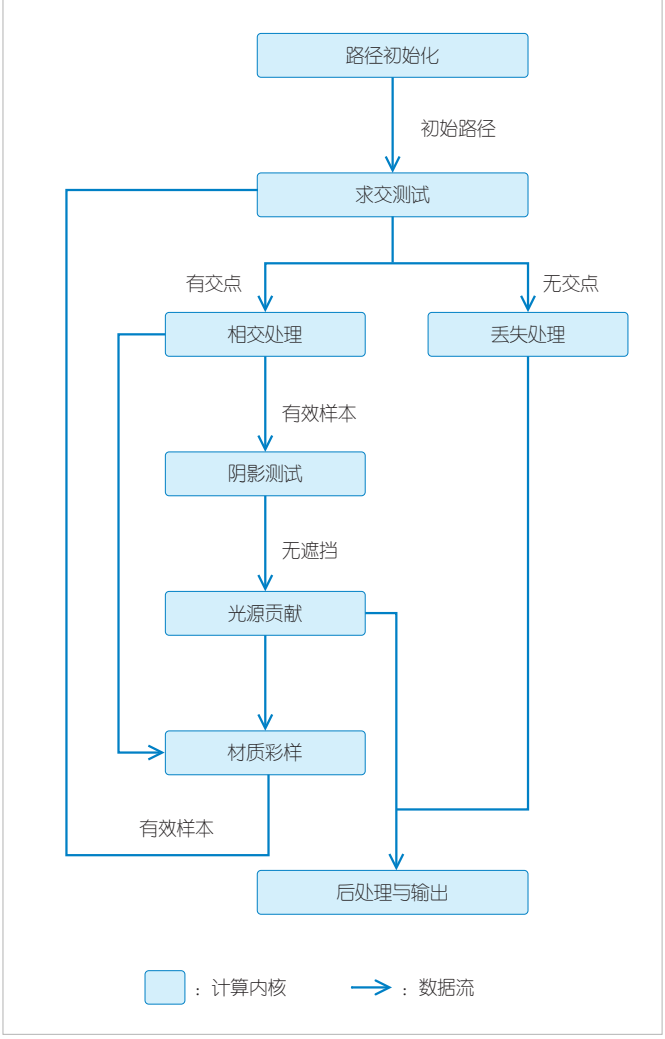
2.3 SoA的数据内存布局

路径追踪算法过程具有大量的全局内存访问操作，而数据在内存中的布局直接影响了GPU中线程的内存访问速度。这是因为在GPU硬件的设计中，全局内存是以二进制存储在动态随机存取存储器（DRAM）单元中的。相对计算单元的时钟周期来说，DRAM的数据访问速度非常慢。

为了优化对DRAM中数据的访问速度，GPU引入了内存合并技术。若一个线程束内所有线程访问的全局内存是连续的，硬件会将这些访问指令合并成对一个连续内存的访问，通过减少内存业务来提高访问效率。

基于内存合并的规则，我们在实现路径追踪算法时，使用数组结构体[6]（SoA）的内存布局组织数据。如图4所示，若一个对象类型含有3个成员变量X、Y、Z，在面向对象风格的编程模式中，该类型的数组会按照结构体数组（AoS）的形式存储，数据会按照成员的声明顺序排列。这导致在线程束中线程访问不同对象的同一个成员时，需要查询不连续的内存，进而无法使用内存合并技术。反之，使用SoA的内存布局使得同一线程束内的线程访问的是连续内存。

此外，SoA的内存布局还能更好地利用GPU的带宽，因为线程中执行的指令未必需要访问对象的所有成员。SoA的内存布局使得其在访问对象的部分成员时，另一部分成员不需要载入共享内存或者寄存器，从而避免了无效的带宽浪费。

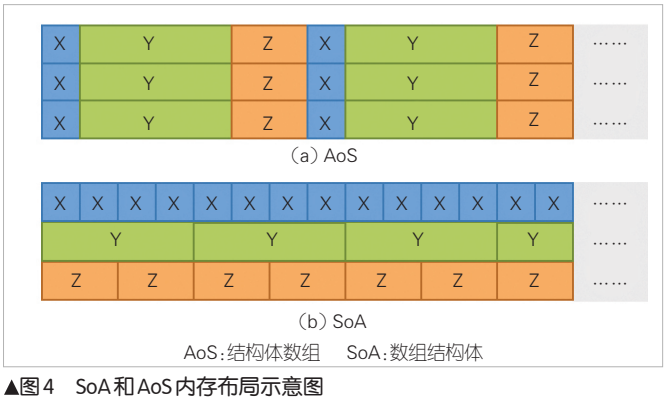


▲图3　流式路径追踪流程图

3 实验分析和效果对比

我们基于硬件光追API OptiX[7]分别实现了传统大内核的路径追踪方法、流式路径追踪方法，以及结合SoA优化内存布局的流式路径追踪方法，并且在英伟达RTX-3090显卡（24 GB）上完成性能的对比测试。

我们在Rendering Resources网站[8]上选取了11个测试场景，分别是：bathroom、bathroom2、bedroom、classroom、kitchen、living-room、living-room2、living-room3、staircase、staircase2和veach-ajar。其中，部分场景渲染结果如图5所示。



▲图4　SoA和AoS内存布局示意图

▲图5 部分测试场景渲染结果图

为了验证流式路径追踪方法和SoA内存布局在GPU上运行的优势，我们固定光线的弹射次数（深度）和质量（单像素的采样数量），在不同场景中分别使用3种方法进行渲染测试。在这11个场景上，为保证渲染效果的真实感，同时平衡渲染耗时，我们设置光线的最大弹射次数为3、单帧的采样数为1。测试得到渲染的平均帧率如表1所示。其中，流式路径追踪算法的实时帧率是大内核方式帧率的2～3倍，使用SoA内存布局的方法使系统性能有一定的提升。

若不考虑实时帧率，设置光线深度为64、单像素采样数为1 024的测试结果如图6所示。不难发现，SoA内存布局在光线弹射次数更多时，对算法性能的优化更明显，能够得到相比于无内存优化方法1.5～3.5倍的性能提升，总的优化提升达到了5.8～8.6倍。

▼表1 测试场景的平均渲染帧数

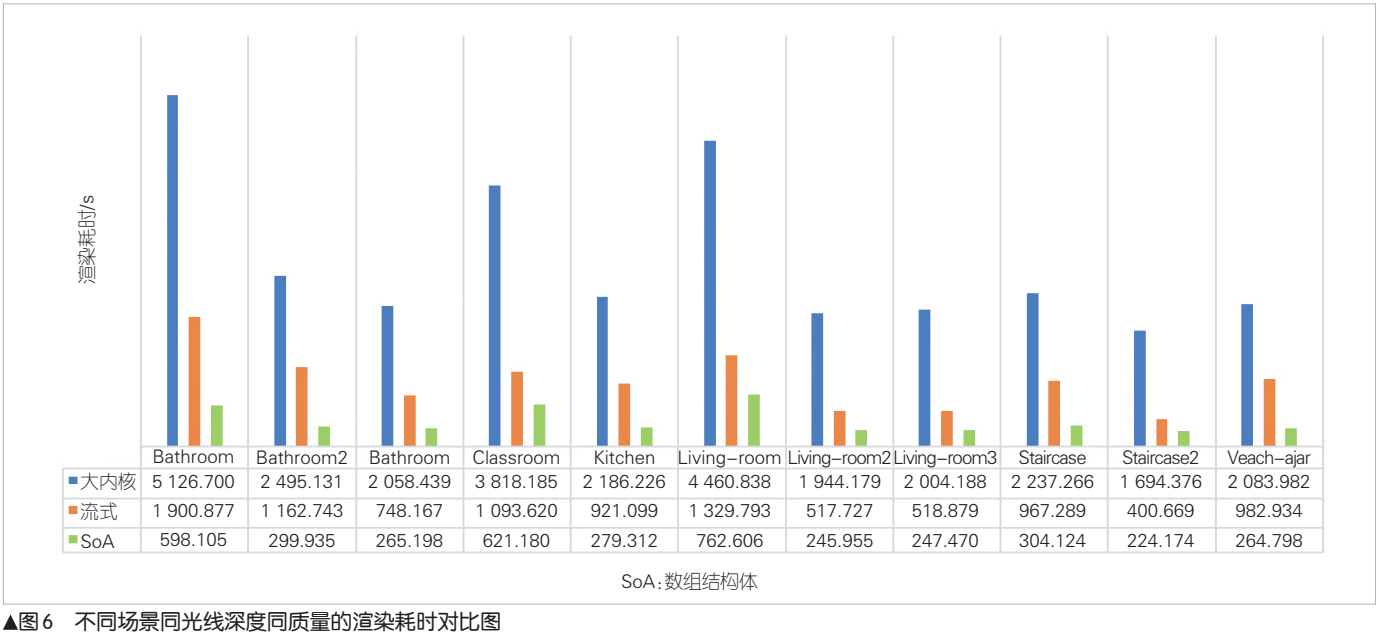
场景	面片数	分辨率	平均帧率(FPS)		
			大内核	流式	流式SoA
Bathroom	1 551 922	1 920×1 080	9	30	38
Bathroom2	3 731 807	1 280×720	22	60	83
Bedroom	4 475 258	1 280×720	24	63	84
Classroom	311 496	1 280×720	9	30	33
Kitchen	4 324 591	1 280×720	23	65	95
Living-room	419 697	1 280×720	8	24	26
Living-room2	1 779 233	1 280×720	23	59	90
Living-room3	2 358 624	1 280×720	21	45	73
Staircase	787 987	720×1 280	25	60	85
Staircase2	92 765	1 024×1 024	20	59	90
Veach-ajar	1 148 068	1 280×720	30	70	110

FPS：每秒帧数 SoA：数组结构体

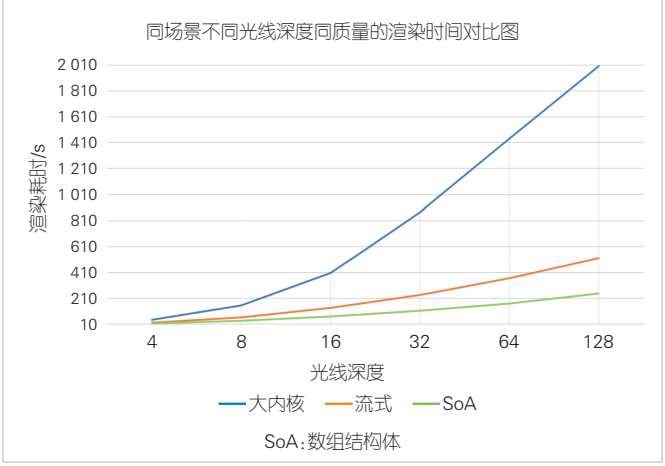
另外，我们改变光线的弹射次数，比较了3种方法的渲染性能。图7展示了在living-room3场景下，3种方法在不同光线深度时渲染所需的时间。可以看出，深度越深流式方法的性能提升越明显。这在加速离线渲染的场景中具有很大的应用价值。

4 总结与展望

本文中，我们从现代GPU硬件的线程调度和内存访问两个角度出发，探索了GPU友好的路径追踪实现方案，提出了流式路径追踪方法，并使用SoA的内存布局优化了算法运行时的内存访问。通过控制光线的弹射次数和采样数，该方法在保证了渲染真实感的同时也满足了实时渲染的性能要



▲图6 不同场景同光线深度同质量的渲染耗时对比图



▲图7 同场景不同光线深度同质量的渲染时间折线图

求，在测试场景中相比于未优化的大内核方法减少了超过80%的渲染耗时。

随着现代GPU硬件的快速发展，使用硬件光追已然成为未来的发展方向，而传统的离线渲染方法，如路径追踪、路径指引等，势必会逐步迁移到实时渲染领域，以提高实时渲染的真实感。同时，这种GPGPU的编程思想可以扩展到其他算法中，有助于现有算法在实际生产中落地应用。

参考文献

[1] GHORPADE J, PARANDE J, KULKARNI M, et al. GPGPU processing in CUDA architecture [EB/OL]. [2024-02-25]. http://arxiv.org/abs/1202.4347
[2] KAJIYA J T. The rendering equation [C]//Proceedings of the 13th annual conference on Computer graphics and interactive techniques. ACM, 1986: 143-150.. DOI: 10.1145/15922.15902
[3] MÜLLER T, GROSS M, NOVÁK J. Practical path guiding for efficient light-transport simulation [J]. Computer graphics forum, 2017, 36(4): 91-100. DOI: 10.1111/cgf.13227
[4] LAINE S, KARRAS T, AILA T M. Megakernels considered harmful: wavefront path tracing on GPUs [C]//Proceedings of the 5th High-Performance Graphics Conference. ACM, 2013: 137 - 143. DOI: 10.1145/2492045.2492060
[5] PURCELL T J, BUCK I, MARK W R, et al. Ray tracing on programmable graphics hardware [J]. ACM transactions on graphics, 21(3): 703-712. DOI: 10.1145/566654.566640
[6] MICIKEVICIUS P. GPU performance analysis and optimization [EB/OL]. [2024-02-25]. https://on-demand. gputechconf. com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf
[7] PARKER S G, BIGLER J, DIETRICH A, et al. OptiX: a general purpose ray tracing engine [J]. ACM transactions on graphics, 29 (4): 66. DOI: 10.1145/1778765.1778803
[8] BITTERLI B. Rendering resources [EB/OL]. [2024-02-25]. https://benedikt-bitterli.me/resources/

作 者 简 介

王宸，南京大学在读硕士研究生；主要研究领域为计算机图形学。

过洁（通信作者），南京大学副研究员；主要研究领域为计算机图形学和虚拟现实技术；先后主持和参加基金项目20余项；已发表论文70余篇。

郭延文，南京大学教授；主要研究领域为计算机图形学和三维视觉；先后主持和参加基金项目20余项；已发表论文100余篇。