The Research on the ray tracing algorithms Base on GPU

Huaqing Mao, Li Zhu
Department of Information Science & Technology Ou Jiang College Wen Zhou University, Wen Zhou, China, 325027
mr.maohuaqing@yahoo.com, yeah_1397118@hotmail.com

Abstract: Scientific visualization, computer 3-D (Three dimension) animation and virtual reality are the three major research directions in computer graphics, and the core technology is 3D realistic graphics. Propelled by the game industry; GPU (Graphics Processing Unit) performance rate is doubled per year. In order to achieve more realistic graphics effect, more and more complex operations are supported by CPU. GPU computing architecture beats the CPU in the cost and power consumption. In this paper, Current status of GPU and ray tracing algorithms are introduced, constructing 3-D scene on the division algorithm by KD-Tree is presented. Eventually, deficiencies are summarized; further research and improvement are pointed out.

Key word: KD-Tree, GPU, Ray tracing

基于 GPU 的光线追踪算法的研究

毛华庆，朱　丽
温州大学瓯江学院信息系，温州，中国，325027
mr.maohuaqing@yahoo.com, yeah_1397118@hotmail.com

摘　要：科学计算可视化、计算机动画及虚拟现实是最近几年来在计算机图形学领域内三大发展方向，其技术核心均为三维真实感图形。在游戏产业的推动下，计算机图形处理器（Graphics Processing Unit）GPU 的性能正以每年翻倍的速度发展，为了实现更逼真的图形效果，GPU 支持越来越复杂的运算，而且在性能上，采用基于 GPU 计算的构架，所需要的成本和功耗都要优于 CPU。本文简要介绍了 GPU 及目前光线追踪发展现状，并给出了利用 KD-Tree 对三维场景进行划分方法和渲染过程，最后对研究中的不足进行了总结并指出了进一步的研究内容和改进方法。

关键字：KD-Tree；计算机图形处理器；光线追踪

1 引言

根据当前实际应用的需求，采用包含丰富信息的直观的三维场景代替简单的二维图像已成为计算机图形学领域的发展趋势。目前，在非工作站电脑中进行高质量的三维场景渲染仍是一个非常耗时的计算过程，不能满足实际应用中的实时性要求，虽然 CPU 主频不断攀升，但因受功耗和发热量的影响，CPU 的频率遇到了瓶颈，例如：Intel 公司的 Pentium IV 发展到现在，其频率一直都在 2GHz-~3GHz 左右，在频率上的提升潜力已经不大，于是，CPU 也开始向双核乃至多核方向发展，因此国内外的学者纷纷将研究重点转移到多核处理器上，利用多核进行并行计算，但基于 CPU 这种架构的性能已很难再次提升，超线程、多流水线、复杂的分支预测、大缓存等技术占用了 CPU 芯片上绝大多数晶体管和面积，目的是让芯片上极少数的执行单元能够满负荷工作，效果不尽人意，在通用计算中的指令级并行仍然偏低。GPU 在进行图形渲染时，整个渲染过程高度并行，因此 GPU 的硬件设计上就是高度并行的，GPU 芯片中拥有大量的执行和运算单元，而控制单元只占很少一部分，而且 GPU 的显存是固化在印刷电路板（PCB，Printed Circuit Board）上

的，显存的运行的频率和带宽都要高于主板内存。作为性能强劲的数据并行处理器，GPU 所具有的高强度的数据并行计算能力为图形渲染以外的计算带来了性能上的新突破，并在物理模拟、信号处理、全局光照、生物计算、几何计算和数据挖掘等领域的应用都取得了较好的研究成果[1-3]。

综上所述，如何利用目前 GPU 的可编程能力以及高强度并行计算能力，追求更高效性能，更低的功耗是目前的研究热点之一。本文在不影响三维渲染质量和算法复杂程度的前提下，研究了如何利用基于 GPU 的高效场景划分算法与渲染方法，以提高图形绘制速度，解决实时三维场景渲染在速度、质量及场景复杂度之间越来越突出的矛盾。

2 GPU 与光线追踪

2.1 GPU 简介

GPU 的运算速度如此之快，主要是因为图形渲染的高度并行性，使得 GPU 具有两点主要特征：超长流水线技术、高速缓存。

流水线技术是一种将每条指令分解为多步，并让各步操作重叠，从而实现多条指令并行处理的技术，超长流水线则是指指令分解成多步的级数较多。在流水线中的程序仍是一条条顺序执行指令，但可以预先取若干条指令，并在当前指令尚未执行完时，提前启动后续指令的另一些操作步骤，这样可加速程序的运行过程。

CPU 中的缓存主要用于减小访存延迟和节约带宽。但缓存在多线程环境下会失效，原因是在每次线程上下文切换后，都需要重建缓存上下文，每次缓存失效会造成几十甚至上百个时钟周期的的损失。与此同时，为了实现缓存与内存中的数据保持一致，还需要额外的复杂逻辑进行控制。在 GPU 中则没有复杂的缓存体系和替换机制。GPU 中的缓存是只读的，从而解决了存贮器访问的读写冲突，不需要考虑缓存一致性的问题，其主要功能是用于过滤对存储控制器的请求，减少系统对显存的访问。因此 GPU 中的缓存主要作用就是节约显存带宽，而不是减小访存延迟。

2.2 光线追踪研究现状

在目前所有实时的三维场景渲染的程序中，基本上都是利用光栅化进行渲染的。虽然光栅化非常适合硬件计算，速度非常快，很成熟。但其缺点在于光栅化技术很难实现全局光照、折射、反射、阴影等效果。为了实现更加真实进行三维实时渲染，Turnerr Wllitted[4]在 1980 年提出了一种基于物理原理的绘制过程光线跟踪的绘制方法，它能够真实模拟反射，折射，硬阴影等效果。Reshetov 等人[5]利用基于 CPU 的光线包跟踪进行渲染，但该方法仅在计算主光线上效率较高，一旦光线经过反射等步骤，方向开始分散，导致效率低下。Purcell[6]等和 Carr[7]等同时在 2002 年提出来第一个由 GPU 实现的光线跟踪算法的框架。Purcell 等人提出的光线跟踪算法将所有的光线跟踪内核都建立在 GPU 之上，包括视线产生器，遍历器，求交器和着色器。Carr 等人仅仅把 GPU 作为一个光线—三角形求交器，将其它的光线跟踪的工作留给 CPU。这两种方法的不同之处在于，前者将所有的光线跟踪处理过程都放在 GPU 上完成，而后者将光线跟踪的不同工作分别交由 CPU 和 GPU 处理。虽然前者能够消除 CPU 与 GPU 之间的频繁的数据传输而带来的损失，但这种方法会由于数据准备使 GPU 大量处于空闲阶段。而后者虽然能够获得更高的 GPU 使用率，因为光线在被发送到 GPU 之前，已经提前在 CPU 上被处理成束，所以在 GPU 光线求交前就已经获得了光线的一致性，但是这种方法会由于 CPU 和 GPU 之间大量的数据传输而带来性能上的损失。Sven Woop 等人[8]通过重建一个光线跟踪算法的无组织内存存取结构模式，以及重建一个用于渲染场景的比显存要大的虚拟内存结构，提出了一个完全的硬件体系结构，将整个场景采用均匀网格来表示，这样将整个光线跟踪的算法放到 GPU 上来执行。并在 2005 年的 SIGGRAPH 上展示了第一个实时光线追踪加速硬件：光线处理单元 RPU（Ray Processing Unit）。但由于该硬件在其他方式渲染上性能仍有待提高，而且成本较高，普及仍需要时日。英伟达（NVIDIA）公司在 2007 年 6 月推出了基于采用该公司 GPU 核心计算的全新基础构架 CUDA（Compute Unified Device Architecture）[9]，该构架提供了硬件的直接访问接口，而不必像传统方式一样必须依赖图形 API 接口来实现 GPU 的访问，从而给大规模的数据计算应用提供了一种更加强大的计算能力。

3 基于 GPU 优化的光线追踪

3.1 光线追踪原理

光线追踪是通过视点向屏幕上的像素发射光线，当光线与三维场景中的物体发生碰撞时就会启动一条

978-1-935068-18-1 © 2010 SciRes.

着色程序，该程序会生成额外的光线，又时会生成大量的光线往各个方向发射。生成的光线越多，渲染的细节就越细腻，但显然运算量就会越大。随着光线数量的不断的反射和折射，可以想象运算量的庞大程度，再加上 SIMD 架构在面对随机性运算时效率低下，也正是这两个原因，实时光线追踪一直停留在实验阶段。

光线追踪步骤如图 1 所示，其步骤如下：

1.设定一个视点，由这个视点射出的光线会穿过显示屏幕上的每一个像素点；

2.构造视点光线的交汇点，在上图中视线焦点为一个球模型；

3.在构造完球模型后，调出球模型的光照程序；

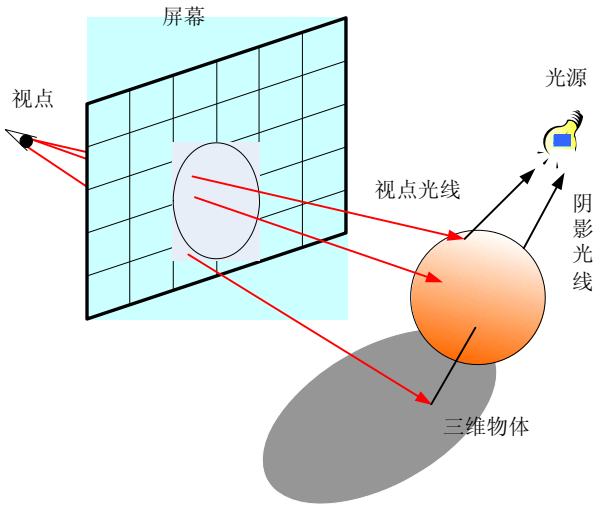

Figure 1. Ray Tracing Principle

图 1 光线追踪原理

4.假设球模型的材质是可以反射光线的，所以在上图中球模型的光照程序被调用后，球模型表面的光线会被完全反射，这些反射光线以球为原点折射出去；

5.这些反射光线会映射在某一个区域中，由于有了法线，入射光线，如果相交物体具有反射属性，可以非常容易的生成反射光线。那么这些由反射以及折射生成的一些光线就被称为次级光线。可以用同样的方法对其进行求交测试，然后进行着色，贡献到相应的像素上。这样就可以很轻松的实现了折射反射的效果；

6.反射光照程序被调用后，依照现实环境就会产生出阴影光线，并且这个阴影光线将是透过反射的成像区域进入光源方向；

7.在反射成像区的点与光源之间存在遮挡物—球模型，由于球模型存在，所以不会有任何颜色被添加到这个多边形区域的点上，因此这个点就会被我们描绘成物件的阴影。不过这个阴影区域一旦回归到帧缓存中后将会恢复其本身的颜色。

3.2 基于 KD-Tree 的场景划分

为了加速光线跟踪，利用空间划分结构，把空间合理的组织到树型结构中，从而把线性的复杂度降低到了对数级别，可大大缩短光线跟踪的计算时间。KD-Tree 是一种在 K 维空间内对图元进行轴平齐的空间二分剖分方法。

KD-Tree 通过不断把当前节点的场景划分成两部分，即两个轴平齐的包围盒而结构化无序场景，从而极大的加速光线求交过程（如图 2a 中所示）。

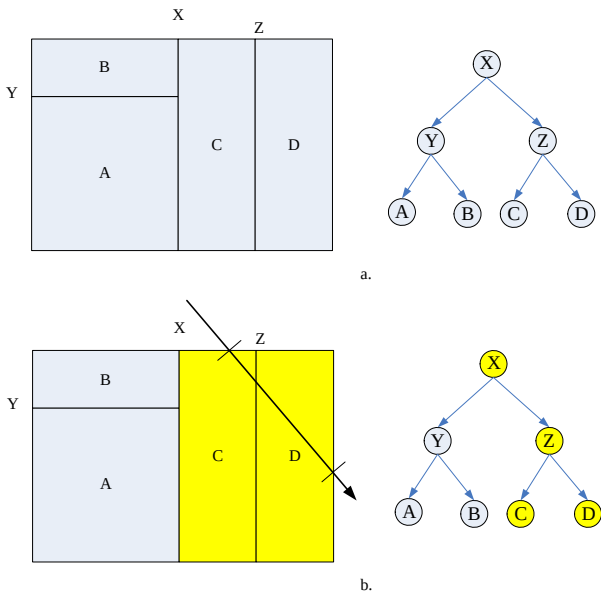

Figure 2. KD-Tree Scene Division and Intersection Test

图 2 KD-Tree 场景划分与相交检测

场景划分结束后，应对光线与图元进行相交检测，其过程如图 2b 中所示，黑色的射线和 A、B 两个图元不可能有交点。利用 KD-Tree 划分结构，我们看到黑色射线经过的叶节点一共有两个。而与射线相交的图元一定在这两个节点里面，所以我们根本不用把这条射线和所有的图元求交。只测试在它经过节点里面的图元就可以了，而且一旦有一个图元相交了，那么下一个节点就没有任何必要再测试了。

利用 CUDA 来实现基于 GPU 的光线跟踪算法，

因为每个像素的颜色值是高度独立的，这也非常适合 CUDA 的架构。虽然 GPU 没有栈的硬件实现，遍历树的时候可能会有一点困难。但是利用数组或者位操作模拟一个栈的行为时完全可行的。

整个渲染过程是根据每一个像素来逐一实现，整个渲染过程分为几个部分：生成视点光线，KD 树的遍历，三角形交叉检测，和阴影显示。以下是渲染模板的伪代码。

```
// 生成视点光线;
// KD-Tree 遍历
for (循环判断是否有叶结点需要检查)
{
    // 查找 KD-tree 的下一个叶结点;
    for (对每个叶结点对应的三角图元进行操作)
    {
        // 三角形相交测试
        if (光线与图元相交)
        {
            设置图元信息;
        }
    }
}
if (图元为空)
    用背景色填充该像素;
else
    渲染;
```

在渲染的时候，每一帧视点和灯光的位置都有可能实时变化，但相对于渲染中的 GPU 来说，每一帧中的视点和灯光的位置都是固定不变的，因此可将视点和灯光信息存储在 CUDA 的常量内存中。

在预先知道三维场景中物体或视点的运动轨迹的情况下，那么可以把不同帧的绘制任务交给不同的线程，从而提高单位时间的绘制效率，不同线程之间只需要共享一些场景的静态信息，而不需要协同和相互等待，因此最大化并行效率。这个方法直接，简单易实现，能极大的提高单位时间的绘制效率，对一些离线动画的绘制应用来说十分实用。但是，如果事先只能知道当前帧的相机参数，或者对于真实感图像生成的应用来说，我们需要利用单帧图像中的计算并行性。

4 结论

目前光线追踪技术的问题在于不能预先渲染动态、随机的三维场景，它只能实时处理的弊端显然不能满足性能上的需求，同时光线追踪技术本身也需要进一步完善自身。本文的研究均基于 NVIDIA 公司旗下支持 CUDA 的 GPU 进行研究，未来还应采用 OpenCL 通用解决方案以推广到其他平台上。另外，由于硬件环境的限制，还未对多 GPU 以集群的方式进行实验研究。

另外，要想用三维技术真实逼真的显示现实场景，还应考虑到光线的二次处理，当光线遇到任何一个表面后，都会产生反射光，而当光线遇到透明物体时，则产生折射光。如果反射或折射光再次遇到另外的物体表面或透明物体后，则再次产生反射与折射，从而产生光线多次反射和折射，光线处理的次数越多，显示的效果越逼真。但由于光线追踪的计算量过大，因光线的二次甚至多次处理问题还有待进一步进行深入研究。

References (参考文献)

[1] Owens J.D., Luebke D., Govindaraju, et al. A survey of general—purpose comutation on graphics hardware [C]. Computer Graphics Forum, 80-113, 2007.

[2] Wu enhua. State of the Art and Future Challenge on General Purpose Computation by Graphics Processing Unit [J]. JOURNAL OF SOFTWARE. 2004, 15, 10

[3] Wu enhua, Liu youquan. General Purpose Computation on GPU [J]. JOURNAL OF COMPUTER-AIDED DESIGN & COMPUTER GRAPHICS. 2004, 16, 5

[4] Turner Whitted. An Improved Illumination Model for Shaded Display [J]. Graphics and Image Processing. 1980

[5] Alexander Reshetov, Alexei Soupikov, Jim Hurley. Multi-level ray tracing algorithm[C]. ACM SIGGRAPH. 2005

[6] Timothy J. Purcell, Ian Buck, William R. Mark, Pat Hanrahan. Ray Tracing on Programmable Graphics Hardware [C]. ACM Transactions on Graphics. 2002

[7] Nathan A. Carr, Jesse D. Hall, John C. Hart. The ray engine [C]. SIGGRAPH/EUROGRAPHICS Conference On Graphics Hardware. Saarbrucken, Germany. 37 – 46. 2002

[8] Sven Woop, Jörg Schmittler, Philipp Slusallek. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing [C]. ACM Transactions on Graphics (TOG). 434 – 444. 2005

[9] http://www.nvidia.com/object/cuda_home_new.html [OL]

978-1-935068-18-1 © 2010 SciRes.