

基于 CUDA 的并行碰撞检测算法研究

田园, 万毅

(兰州大学 信息科学与工程学院, 甘肃 兰州 730000)

摘要:碰撞检测是计算机图形仿真中的关键问题之一。尽管研究人员提出了许多优秀的碰撞检测算法,但是随着仿真场景规模的增大,在单处理器上实现的碰撞检测算法已经难以达到实时性的要求。因此,当前研究的核心问题是如何提高碰撞检测的速度。在对已有算法研究分析的基础上,提出了一种基于层次包围盒的并行碰撞检测算法。该算法的核心思想是用多处理器并行遍历层次树以避免单处理器需要两棵树相互遍历的情况,并提出以并行的方式生成层次包围盒树来进一步提高算法效率。结合 CUDA 平台提供的并行计算解决方案,整个算法在图形处理器上得以实现。结果表明,该算法显著地提高了碰撞检测的速度,满足实时性的需求。

关键词:计算机图形;碰撞检测;层次包围盒;并行;CUDA

中图分类号:TP3-0

碰撞检测问题在计算机图形学、虚拟现实、计算机游戏以及 CAD/CAM 等领域有着非常重要的研究意义^[1]。在计算机生成的虚拟场景中,必须遵循一个现实生活中普遍存在的事实:两个不可穿透的物体不能同时占据空间中相同的区域。因此,碰撞检测问题对于能否产生具有真实感的图形场景至关重要。近 20 年来,国内外研究人员已经在碰撞检测领域中做了相当多有意义的工作,针对不同的应用、采用不同的方法提出了很多实用的碰撞检测算法,为碰撞检测技术的快速发展做出了重要贡献^[1]。现有的碰撞检测算法大致可分为两类:空间分解法和层次包围盒法。这两种方法的目的是减少不必要的几何元素相交测试,提高碰撞检测的效率。然而,随着人们对虚拟场景真实度要求的不断提高,场景规模越来越大,模型也越来越复杂,现有的碰撞检测算法已无法满足实时碰撞检测需要。因此,如何减少碰撞检测所带来的系统消耗问题已经成为重要的研究内容之一。

近年来,计算行业正在从只使用 CPU 的“中央处理”向 CPU 与 GPU 并用的“协同处理”发展。为了打造这一全新的计算模式,NVIDIA 公司开发了一种称为 CUDA(Compute Unified Device Architecture,统一计算设备架构)的通用并行计算架构,该架构使得 GPU 能够解决复杂的计算问题。得益于 CUDA 所带来的并行计算解决方案,并行碰撞检测算法的研究迎来了新的机遇。并行计算是一种能够有效提高碰撞检测速度的方法,经过分析研究,提出了一种基于 CUDA 的并行碰撞检测算法。

1 并行碰撞检测算法的基本思想

我们所提出的是一种基于层次包围盒的并行碰撞检测算法。层次包围盒的碰撞检测算法由于具有较好的性能,适用于复杂环境中的碰撞检测,因而受到了广泛的研究。层次包围盒方法是利用体积略大而形状简单的包围盒把复杂的几何对象包裹起来,在进行碰撞检测时首先进行包围盒之间的相交测试;如果包围盒相交,再进行几何对象之间精确的碰撞检测^[2]。在单处理器的情况下,首先必须对两个需要进行碰撞检测的对象分别生成层次包围盒树(简称包围树),再通过双重递归遍历两棵包围树来确定需要进行检测的部分。如果某个对象在模拟过程中发生位移或形变,则必须在下一次进行碰撞检测前对该对象的包围树进行更新,而更新过程依然是一个遍历树的过程。由此可见,当一个对象含有的基本几何元素越多时,生成包围树的时间越长,并且包围树的深度越大,遍历所需要的时间也更长。为了减少碰撞检测的系统消耗,则利用并行的方法来生成包围树和避免双重递归遍历。根据分析,层次包围盒的碰撞检测是一种典型的单指令多数据流^[3](Single Instruction Multiple Data,简称 SIMD)问题,即处理器多次执行重复的代码,每次处理不同的数据。对于这类问题,可以使用多个处理器,每个处理器同一时刻执行相同的代码但处理不同的数据,从而实现并行化。

1.1 并行生成包围树

生成一棵包围树有自顶向下和自底向上两种方

法。自底向上生成包围树的方法与霍夫曼编码相似。单个处理器完成这项工作的方法是:

1) 根据给定的 n 个基本几何元素的包围盒构成 n 棵二叉树的集合 $F = \{T_0, T_1, \dots, T_n\}$, 其中每棵二叉树 T_i 中只有一个根结点, 其左右子树均为空。

2) 从 F 中取出 T_i , 用贪婪算法在其余的二叉树中找到与 T_i 距离最近的 T_j , 将 T_i 与 T_j 作为左右子树构造一棵新的二叉树, 并且新二叉树根结点包围盒大小由其左右子树决定。重复该过程直到 F 中所有的二叉树被处理完。该过程的时间复杂度为 $O(n^2)$ 。

3) 删除 F 中原有的二叉树, 将新生成的树全部加入 F 中。

4) 重复(2)和(3), 直到 F 只含有一棵树为止。

上述算法的时间复杂度为 $O(n^2 \log_2 n)$ 。从生成树的过程来看, 处理器重复做相同的工作来处理不同的数据, 属于 SIMD 问题, 因此, 可以使用并行的方式。

提出的并行生成包围树算法的基本思想是:

1) 根据 n 个基本几何元素的包围盒构成 n 棵二叉树的集合 $F = \{T_0, T_1, \dots, T_n\}$, 其中每棵二叉树中只有一个根结点, 其左右子树均为空。

2) 从 F 中取出 T_i , 并行计算其余根结点到 T_i 的距离, 再用并行的方式找到与 T_i 距离最小的根结点 T_j , 将 T_i 和 T_j 加入到集合 $H = \{S_0, S_1, \dots, S_n\}$ 中。重复该过程直到 F 中所有的根结点全部加入 H 中, 其中 S_{2i} 与 S_{2i+1} 之间的距离最小。

3) 用多个处理器 P_0, P_1, \dots, P_m ($m = \frac{n}{2}$) 对有序集合 H 中的根结点进行处理, 由处理器 P_i 将 H 中 S_{2i}, S_{2i+1} 作为左右子树构造一棵新的二叉树。

4) 删除 F 中原有的二叉树, 将新生成的树全部加入 F 中。

5) 重复(2)、(3)和(4), 直到 F 只含有一棵树为止。

并行生成树的过程如图1所示。对于有 n 个根结点和 p 个处理器的情况, 该算法在计算根结点之间距离的时间复杂度是 $O(\frac{n}{p})$; 在上述算法(2)中,

总共需要进行 $\frac{n}{2}$ 次查找, 并行查找最小值的时间复杂度为 $O(\frac{n}{p} \log_2 n)$, 那么得到集合 H 所需要的时间复杂度为 $O(\frac{n^2}{p} \log_2 n)$; 在生成树时, 并行算法的时

间复杂度为 $O(\frac{n}{p} \log_2 n)$ 。整个算法的时间复杂度为 $O(\frac{n^2}{p} \log_2 n)$, 与单处理器所采用的算法相比较, 当 p 越接近 n 时, 并行算法的效率越好。

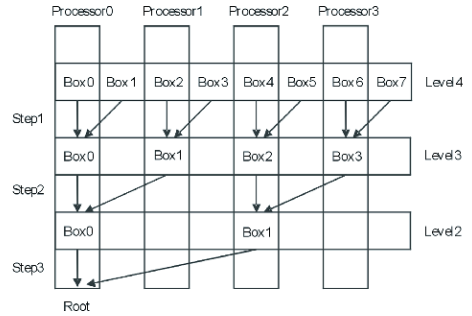


图1 并行生成包围树的基本思想

1.2 并行遍历包围树

单处理器进行碰撞检测时需要两棵包围树进行双重递归遍历。假设对象 A 与 B 进行碰撞检测, 若树 A 根结点与树 B 根结点的包围盒相交, 则树 A 向下遍历, 当到达树 A 的某个叶结点时, 用该叶结点遍历树 B, 如果能到达树 B 的叶结点, 再进行进一步的几何元素相交测试。假设对象 A 的包围树含有 n 个结点, 对象 B 的包围树含有 m 个结点, 那么完成碰撞检测的时间复杂度为 $O(n \times m)$ 。

为了避免双重递归遍历所带来的系统消耗, 减少遍历的次数, 我们的想法是只生成对象 A 的包围树, 在碰撞检测时, 只为对象 B 的所有基本几何元素生成包围盒, 然后由多个处理器分别处理每个包围盒与树 A 的遍历问题。该方法的时间复杂度是 $O(n \times \frac{m}{p})$ (p 是处理器个数), 与并行生成包围树一样, 当 p 越接近 n 时, 算法的效率越好。

2 并行碰撞检测算法的实现

自 2006 年 NVIDIA 公司推出统一渲染架构的 G80 核心以来, 可编程流处理器已取代原有的固定管线流程成为当今 GPU 发展的模式。G80 核心拥有 128 个 Streaming Processor (流处理器, 简称 SP), 每 8 个 SP 被划分为 1 个 Streaming Multiprocessor (流多处理器, 简称 SM), 每个 SM 可容纳 768 个活动线程, 那么一个 G80 核心便可以同时建立 12280 个活动线程。而 NVIDIA 之后推出的 G200 核心, 拥有 240 个 SP, 每个 SM 可容纳 1024 个线程, 能够同时建立的活动线程数更是高达 30720 个^[4]。由此可见, GPU 为并行计算提供了一个很好的解决方案。

CUDA 技术是一种基于 NVIDIA 图形处理器的并行计算体系架构,并使用标准 C 语言作为其编程语言。CUDA 程序分为在 host 执行和在 device 执行两个部分,host 就是传统的 CPU,而 device 就是支持 CUDA 的 GPU。在执行 CUDA 程序时,host 将需要并行计算的数据在内存中准备好,传入显存中由 device 进行处理,数据处理完成后从显存传回内存中供用户使用。

根据所提出的算法,首先是生成对象的包围树。定义一个基本几何元素的包围盒为一个含有包围盒顶点坐标、几何元素顶点坐标的结构体。由 host 为对象所有基本几何元素生成包围盒并存储在内存中的一维结构体数组中,根据几何元素个数计算出包围树深度 depth 及每层结点的个数。假设包围树每层结点个数分别为 $n_i, n_{i-1}, \dots, 2, 1 (i = \text{depth} - 1)$, 在显存中分配大小为 $n_i + n_{i-1} + \dots + 2 + 1$ 的结构体数组用于存储生成的包围树。将包围盒数组从内存中传入显存后,在 device 上为数组中每一个元素建立一个对应的线程,由这些线程完成以下工作:

1) 以数组中指定元素对应的包围盒作为目标,计算其余包围盒与它之间的距离。该过程可以由多个线程并行完成,每个线程负责与其对应的包围盒的处理。

2) 将包围盒数组调整成为有规则的序列,使得数组下标为 $2i$ 和 $2i + 1 (i \in N)$ 的两个包围盒距离最近。这一过程的方法如图 2 所示。在 Step 1 中,指定数组中第一个元素 T_0 为目标包围盒,计算其余包围盒到 T_0 的距离并用并行查找最小值的方法找到与 T_0 距离最近的包围盒。查找结束后,与 T_0 距离最近的包围盒将被移至数组中 T_1 的位置。再指定 T_2 作为目标,用相同的方法找到与 T_2 距离最近的包围盒。这一过程将被重复执行 $\frac{n}{2}$ 次,直到数组被调整完为止。

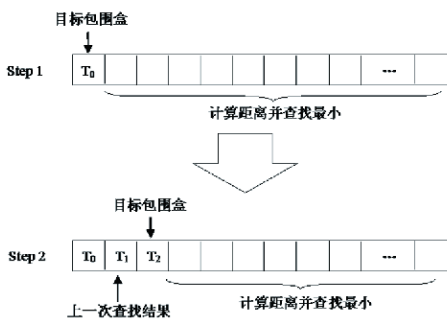


图 2 调整包围盒序列

这里使用的并行查找最小值的方法如图 3 所示。将含有 n 个元素的数组分成 n 个只有一个元素的子序列,使用 $\frac{n}{2}$ 个处理器对两个相邻子序列进行合并,得到 $\frac{n}{2}$ 个新的子序列,在合并时将对比两个子序列中第一个元素的值,将较小的值作为合并后子序列中第一个元素的值,再对新得到的 $\frac{n}{2}$ 个子序列两两合并,……,如此反复直到合并成一个序列为止。整个过程结束后,数组的最小值将位于数组第一个元素中。

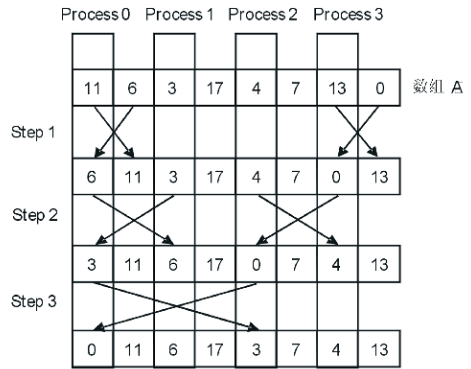


图 3 并行查找最小值

3) 由多个线程生成包围树每层结点,并存储在事先分配好的数组中。该过程如图 4 所示。

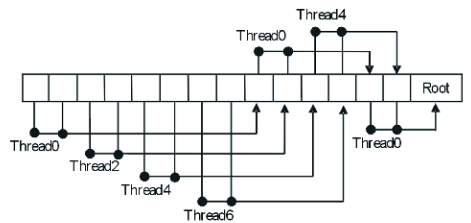


图 4 多线程生成包围树

假设对象 A 与 B 进行碰撞检测,并且 device 已为对象 A 生成包围树,那么只需要为对象 B 所有基本几何元素生成包围盒,用每个包围盒去遍历对象 A 的包围树。当 host 将对象 B 的包围盒数组从内存中传入显存后,在 device 上建立与包围盒数目相同的线程,由每个线程独立完成一个包围盒与包围树的遍历问题。由于 CUDA 不支持堆栈结构,因此,CUDA 线程深度遍历二叉树无法用递归或堆栈的方式实现。文献[5]中给出了一种模拟堆栈^[5]的方法,使得 CUDA 线程深度遍历二叉树得以实现。

3 实验结果

我们在中国科学院近代物理研究所的超级计

算中心上对并行算法进行实验。该超算中心采用联想深腾 7000G 高性能服务器,每个计算结点配置 2 个 Intel Xeon 2.33GHZ 处理器,8G 内存和 4 个 G200 核心,能够获得 50 ~ 150 倍于通用 CPU 的加速计算效果。实验所用的碰撞场景如图 5 所示。在场景中水平放置一块布,并固定布的两个角,使其在受到重力作用自然下垂时呈悬挂状态。在布的下方放置一个球体,使布在下垂过程中受到球体的阻挡而部分覆盖在球体上。当布覆盖在球体上并达到静止状态时,开始统计并行算法和串行算法在生成包围树和遍历树所需要的平均时间。

表 1 是 GPU 并行算法和 CPU 串行算法执行时间的对比。从实验结果来看,当问题规模较小时, GPU 在生成包围树所需要的时间比 CPU 要长,其主要原因在于 CUDA 在组织线程、启动内核函数和

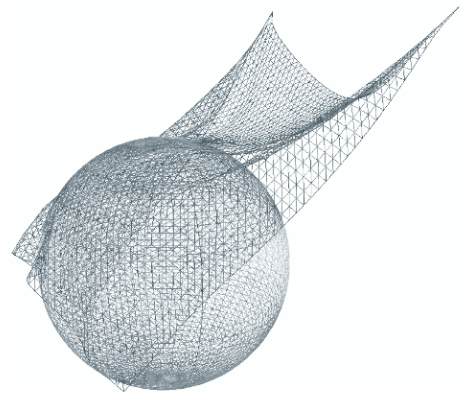


图 5 碰撞检测实验场景

分配任务所消耗的时间掩盖了实际的执行时间。在完成一次碰撞检测的时间方面, CPU 随着问题规模的增大,需要的时间也越来越长,而 GPU 则只需要遍历一棵树的时间便能够完成碰撞检测,较 CPU 有了明显的提高。

表 1 碰撞检测串行算法与并行算法时间对比

基本几何元素个数				执行时间(ms)						
球体		布		生成包围树			完成碰撞检测			
顶点	三角形	顶点	三角形	CPU	GPU	加速比	CPU	GPU	加速比	
1024	1922	576	1058	62	140	1: 0.44	17.6	11.6	1: 2.04	
4096	7938	2304	4418	1052	786	1: 1.34	137.5	48.2	1: 2.85	
16384	32258	4096	7938	25725	5740	1: 4.48	518.8	187.3	1: 2.76	

4 总结

在分析基于层次包围盒的碰撞检测算法的可并行性后,提出了基于 CUDA 的并行碰撞检测算法。通过并行生成包围树和避免双重递归遍历包围树使得碰撞检测的效率得到了很大的提高。就此提出的并行生成包围树的思想,同样可以用于更新包围树。实验证明,在 GPU 上实现的并行碰撞检测算法能更好地满足实时性的需要。

参考文献:

- [1] 石其. 基于 GPU 的碰撞检测算法研究[D]. 长沙:湖南大学,2009.
- [2] 马登武,叶文,李瑛. 基于包围盒的碰撞检测算法综述[J]. 系统仿真学报,2006,18(4):1058-1061.
- [3] Michael J. Quinn. PARALLEL PROGRAMMING IN C WITH MPI AND OPENMP[M]. 北京:清华大学出版社,2005.
- [4] David B. Kirk, Wen - mei W. Hwu. Programming Mas -

sively Parallel Processors[M]. 北京:清华大学出版社,2009.

- [5] 曹家音. 基于 KD - Tree 遍历的并行光线跟踪加速算法[J]. 科技传播,2010(17):233-233,224.
- [6] 熊一梅,曾究文,陈一民. 基于并行的快速碰撞检测算法的研究[J]. 计算机应用与软件,2008,25(4):48-50.
- [7] Xavier Provot. Deformation Constraints in a Mass - Spring Model to Describe Rigid Cloth Behavior[J]. In Graphics Interface 1995,18(5):147-154.
- [8] 陈晔,徐乃平. 真实感布仿真中布与刚体的碰撞检测及修正[J]. 软件学报,2001,12(12):1874-1880.
- [9] 王晓荣,王萌,李春贵. 基于 AABB 包围盒的碰撞检测算法的研究[J]. 计算机工程与科学,2010,32(4):59-61.
- [10] 沈照功,潘振宽. 基于弹簧质点模型的布料仿真及碰撞处理方法[J]. 计算机仿真,2006,23(3):284-287.