图形硬件加速的柔性物体连续碰撞检测

唐 敏 林 江 童若锋
(浙江大学计算机科学与技术学院 杭州 310027)

摘 要 给出了一种图形硬件加速的柔性物体连续碰撞检测算法,可以实时检测复杂柔性物体场景中所有物体间碰撞和自碰撞.算法将柔性物体的碰撞检测过程进行流式分解,映射到图形硬件上并行执行,同时使用了并行流式登记算法,在图形硬件上高效实现了变长数据结构.该算法已经使用 OpenCL 在 AMD Radeon HD 5870 图形硬件上实现.针对一组各具特色的柔性物体仿真场景进行测试,对比 CPU(Intel Q6600@2.4GHz)上的单线程优化实现,可以获得 9.2～11.4 倍的计算加速.

关键词 柔性物体;连续碰撞检测;流式映射;图形硬件;OpenCL
中图法分类号 TP391 **DOI 号**: 10.3724/SP.J.1016.2010.02022

Graphics Hardware Accelerated Continuous Collision Detection Between Deformable Objects

TANG Min LIN Jiang TONG Ruo-Feng
(*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310027)

Abstract This paper presents a graphics hardware accelerated continuous collision detection algorithm for deformable objects. It can detect all inter-object and intra-object collisions for complex scenes which consist of deformable objects in real time. The algorithm executes collision detection tasks fully in parallel by mapping them with streaming decomposition. With parallel streaming registration algorithm, variable length data structures are efficiently supported on graphics hardware. The algorithm has been implemented on AMD graphics hardware platform with OpenCL. A serial of benchmarks with different characteristics are used for testing. Comparing to an optimized single-threaded CPU implementation on Intel Q6600@2.4 GHz, the algorithm achieves 9.2x～11.4x speedups.

Keywords deformable objects; continuous collision detection; streaming maping; graphics hardware; OpenCL

1 引 言

真实世界中的物体并非完全是刚体,它们在运动过程中常会产生一定的形变,即所谓柔性物体.对于柔性物体的交互式仿真一直是基于物理模型仿真的重要研究内容,并广泛应用于视频游戏、CAD/CAM、机器人仿真、计算机动画等领域.为了模拟物理真实或近似真实的物体变形过程,应用系统需要使用非穿透约束监测所有几何碰撞情况,并对碰撞做出快速反应.在视频游戏中,赛车与场地或其它赛车的碰撞、橄榄球运动员间的冲撞、战场中的弹片飞

收稿日期:2010-08-22.本课题得到国家自然科学基金(60803054)、浙江省自然科学基金(Y1100069)和教育部-Intel 信息技术专项科研基金(MOE-INTEL-09-05)资助.**唐 敏**,男,1974 年生,博士,副教授,主要研究方向为碰撞检测、图形硬件算法加速.E-mail: tang_m@zju.edu.cn.**林 江**,男,1987 年生,硕士研究生,主要研究方向为图形硬件算法加速.**童若锋**,男,1969 年生,博士,教授,主要研究领域为图像与视频处理.

溅都需要进行碰撞检测完成.在某些游戏中,碰撞检测的 CPU 耗时可达总量的 50%～90%[1].因此,快速高效的碰撞检测对于提升游戏引擎的性能至关重要.

连续碰撞检测方法被用于精确计算柔性物体间发生的碰撞.连续碰撞检测方法插值物体的运动路径,将碰撞检测问题转化为非线性多项式方程求根问题,可以计算出物体在离散位置间发生的碰撞情况及返回离散实例间的第一接触时间[2].连续碰撞检测方法确保了不会漏检变形过程中物体间碰撞或物体自碰撞,但其计算量较大.例如,一对变形三角形需要进行 15 个 Edge-Edge/Vertex-Face 元素测试,而每个元素测试则需要求解一个单变量多项式方程.对于较复杂的仿真场景,实时连续碰撞检测依然是一个挑战.

图形硬件在近期性能不断大幅度提升,其浮点处理能力已经超越了市场主流多核 CPU.本文的目标是使用最新的图形硬件对柔性物体的连续碰撞检测进行加速,使其达到实时处理速度.

本文的主要贡献如下:

(1)对柔性物体的碰撞检测过程进行了流式抽象,提出了一种流式映射方法,将涉及的几何数据表达为多种流数据,同时将碰撞检测任务分解为一系列可以在流式处理器上充分并行执行的核心(kernel),使其在图形硬件上高效执行.

(2)针对碰撞检测结果出现的随机性,设计了一种并行流式登记算法,在图形硬件上高效实现了变长数据结构.该算法克服了现有方法消耗大量显存或使用原子操作的弊端,降低了显存需求并保证了并行执行效率.该算法可应用于其它需要使用变长数据结构的应用,如粒子系统、破碎仿真等.

(3)给出了在图形处理器上柔性物体间碰撞和自碰撞的统一检测算法,克服了常规方法依赖于图像分辨率或需要在图形处理器和 CPU 间传递大量数据的弊端,保证了算法的精确性与高效性.

本文算法已经在许多复杂场景(4K～92K 三角形,包含柔性物体、破裂物体、N-body 仿真等)中得到验证.对比 CPU(Intel Q6600@2.4GHz)上的串行优化实现,算法在 AMD Radeon HD 5870 图形硬件上获得了 9.2～11.4 倍加速.对于所有复杂测试场景,可以检测包含自碰撞的所有碰撞情况,速度达到 6.4～40.5ms/每帧.

本文第 2 节给出相关工作综述;第 3 节详细论述连续碰撞检测算法的流式映射、并行流式登记、图形硬件加速等具体算法;第 4 节给出该算法的实现细节、实验结果和性能分析;第 5 节对比本文算法和前人算法,并分析了本文算法的缺陷;第 6 节为结论与未来工作.

2　相关工作

下面给出柔性物体的连续碰撞检测、包围盒层次结构、图形硬件加速的碰撞检测、流式数据压缩等方面的研究进展综述.

2.1　连续碰撞检测

连续碰撞检测已经成为精确计算仿真过程中第一接触时间的标准工具.它在刚体和铰链模型上的使用已经取得了较好的效果[3-5].对于柔性物体,由于自碰撞情况的大量出现,连续碰撞检测则困难得多[6].对于柔性物体,Hutter 和 Fuhrmann[7] 不仅使用了三角形包围盒,也对其附属特征(顶点和边)使用包围盒以控制伪真率.唐敏[8] 等人使用了一种基于 Hash 表的机制对特征共享造成的冗余元素测试进行剔除.Curtis[9] 等人使用了代理三角形剔除冗余元素测试.唐敏[10] 等人提出了连续法向锥和孤集的概念,剔除相邻三角形和非相邻三角形的冗余元素测试.最近,唐敏[11] 等人提出了一种非穿透性过滤器,可以大幅度降低元素测试的伪真率.

2.2　包围盒层次结构

包围盒层次结构(BVH)是进行高效碰撞检测的重要工具.各种包围盒层次结构,如球树[12-13]、轴对齐包围盒(AABB)树[14]、定向包围盒(OBB)树[15]、离散定向多面体(k-DOP)树[16]等,已经被广泛应用于刚体和柔性物体的碰撞检测过程.对于柔性物体,包围盒层次结构需要伴随着仿真过程动态更新.许多算法都使用了简单的包围盒,并在每帧重新计算包围盒层次结构[5].有些算法使用了重新整理的方法进行更新[17-20].也有部分研究者使用了动态或选择性重构的算法[21-23].Wald[24] 实现了多核 CPU 上的包围盒层次结构并行重建.Lauterbach[25-26] 等人利用了图形处理器的大数目并发线程对包围盒层次结构重建进行加速.

2.3　图形硬件加速的碰撞检测

图形处理器最初专为图形任务设计,使得 CPU 从此类任务中解脱出来.现代图形处理器被设计为一种众核处理器,内部集成了大量并行工作的流式处理单元,支持流数据的大规模并行处理,其浮点处理能力已经大大超出了市场主流多核 CPU.AMD

公司最新推出的 HD5970 图形处理器具有 3200(1600×2)个流式处理单元,核心主频达到 725MHz.使用图形处理器对计算密集型任务进行加速已经被越来越多的研究者重视.

针对多核处理器,唐敏[27]等人使用了 BVTT 前线增量式更新策略,实现了多核体系架构下的碰撞检测任务细粒度分解,进而使用了 SIMD 指令[28]进一步提升性能.但图形处理器与多核 CPU 差异较大:图形处理器中并行处理单元数目庞大,但 Cache 尺寸有限.对于碰撞检测任务,算法需要针对这些特点进行重新设计才能发挥出图形处理器的强大处理能力.

许多研究者曾经使用图形处理器对碰撞检测任务进行加速.早期工作[29]只对粗粒度碰撞检测进行加速.最近,研究者通过使用深度缓存或模板缓存利用图形处理器的光栅处理能力进行碰撞检测加速[30-34]、快速距离场计算[35-36]或分离/穿透距离计算[37].这类方法的处理精度常常受到图像分辨率的限制.Zhang[38]等人使用了 AABB 流对柔性物体的碰撞检测进行加速,然后在 CPU 上串行进行精确测试.Kim[39]等人使用了一种多核 CPU 和图形处理器混合执行的方法进行碰撞检测.这类方法的性能被 CPU 与图形处理器间的数据通信瓶颈所限制.最近,Lauterbach[26]等人在 GPU 上使用了轻量化 BVH 和 OBB 进行离散碰撞检测.

2.4 流数据压缩

图形硬件可以抽象为一台擅长处理流数据的流式处理器.为了提高处理速度,我们需要对流数据进行压缩,去除不需要处理的数据单元.早期的图形硬件不支持 Scatter 操作,因此只能通过 Prefix Sum 算子进行流数据压缩[40].最新的图形硬件直接支持 Scatter 操作,但压缩过程仍然需要使用原子操作.原子操作的互斥性严重降低了算法的并行性,因此并行 Prefix Sum 算子[41]是较为高效的方案.并行 Prefix Sum 算子的一个缺陷为需要较大的显存空间.例如,为了进行分别由 N 个三角形与 M 个三角形构成的两个几何模型间的碰撞检测,需要使用 $N\times M$ 规模的显存空间.而为了处理 N 个三角形间的自碰撞检测,则需要 N^2 规模的显存空间.对于 10K 数量级的仿真场景,这个数目已经超出了图形硬件显存的最大容量.

3 流式碰撞检测

为了充分利用图形硬件的并行处理器能力,我们将图形硬件抽象为一台流式处理器:即支持大规模流式数据的细粒度并行处理,同时提供流数据的 Gather 和 Scatter 操作.针对流式处理器的特点,我们将柔性物体碰撞检测过程中涉及的几何数据抽象为流数据,同时将功能模块映射为可以高效运行在流式处理上的核心.

3.1 流数据

在处理过程中,我们将涉及的几何数据抽象为以下几种流数据(如图 1 所示):

(1)顶点流 S_v.其流数据单元为在每个仿真时间点的模型顶点几何坐标,为了进行连续碰撞检测,我们使用了两组顶点流 $S_v(t_0)$和 $S_v(t_1)$,分别保存前一仿真时间点和当前仿真时间点的顶点几何坐标.

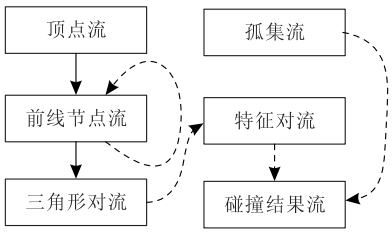

图 1 算法中使用的各种流数据

(2)前线节点流 S_f.其流数据单元为包围盒检测树(BVTT)的前线节点,是 BVTT 中内部节点和叶节点的集合,碰撞查询过程在这些节点处终止.它反映了在两个离散时间点间 BVTT 的遍历情况.

(3)三角形对流 S_t.在对 BVH 进行遍历的过程中,所有通过了包围盒测试的非相邻三角形对被收集为三角形对流 S_t,这些潜在的碰撞三角形对将被进一步分解为特征对进行检测.

(4)特征对流 S_g和孤集流 S_o.所有包围盒发生重叠的非相邻三角形对将被分解为 9 个 Edge-Edge 特征对和 6 个 Vertex-Face 特征对,并使用特征包围盒和非穿透性过滤器进行剔除,潜在的碰撞特征对被保留在特征对流 S_g中.相邻三角形对将被使用孤集流 S_o独立处理,其中保存了所有相邻三角形相关、且未被非相邻三角形对相关测试所覆盖的特征对[10].孤集流 S_o通过分析模型拓扑结构获得.

(5)碰撞结果流 S_{cd}.特征对流 S_g和孤集流 S_o中的所有数据单元将被精确测试,若发生碰撞,碰撞的特征对标示信息和第一接触时间信息将被保留在碰撞结果流 S_{cd}中.

对于任意的柔性物体,其变形过程可以通过顶点流定义.碰撞检测算法以任意时刻的顶点流为输入,对 BVTT 前线进行动态更新,得到更新后的前

线节点流,同时将可能发生碰撞的三角形对,记录在三角形对流中.而对于三角形对流,我们将通过特征包围盒测试和非穿透性过滤器剔除不会发生碰撞的特征对,对需要进行精确元素测试的特征对则记录在特征对流中.最后,特征对流和孤集流将被精确测试,得到发生接触的碰撞时间.

3.2　流数据处理核心

柔性物体的变形过程和碰撞检测过程将被设计为一系列流数据处理核心(图 2).通过在图形硬件上执行这些核心,整个算法完全运行在图形硬件上.

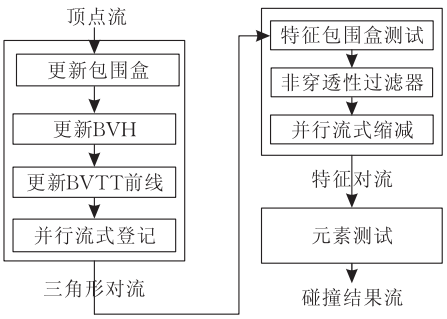

图 2　流数据处理核心

本文算法不依赖于特定的变形算法,即通过提供在各离散仿真时间点的顶点坐标值支持各种变形.对于发生变形的柔性物体,算法在图形硬件上更新几何对象的包围盒、重新整理 BVH.通过进行流式处理,BVTT 前线节点流得到动态更新,使用并行流式登记算法产生紧凑的三角形对数据流(记录所有潜在碰撞的三角形对).对于三角形对数据流经过进一步的处理(特征包围盒测试[7]、非穿透性过滤器[11])同时进行流式缩减,产生了特征对数据流(记录所有潜在碰撞的几何特征对).最后这些几何特征对和记录在孤集流中的几何特征对将被用于元素测试,计算得到碰撞结果数据流(记录所有发生碰撞的几何特征对及其第一接触时间).

3.3　前线节点流的动态更新

为了将碰撞检测任务并行分解,算法使用了基于 BVTT 前线的并行更新策略[27]:即利用了柔性物体仿真过程中的时空相关性,通过对上一个仿真时间点的 BVTT 前线进行动态演进得到当前 BVTT 前线.我们将 BVTT 前线表示为图形硬件上的前线节点流 $S_f(t_0)$,并使用运行在图形硬件上的核心对其进行更新,产生出反映当前仿真时间点碰撞情况的新前线节点流 $S_f(t_1)$.对前线节点流中 $S_f(t_0)$的一个数据单元$\{N_a^i,N_b^i\}$进行更新的算法如算法 1 所示.我们使用了一个临时堆栈保存对$\{N_a^i,N_b^i\}$进行递归遍历(第 13～17 行)的中间结果,将包围盒重叠的节点保存在 $S_f(t_1)$中(第 4 行、第 11 行),同时若 N_a^i和 N_b^i都是叶节点,则它们还将作为潜在碰撞三角形对保存在三角形对流 S_t中(第 6 行).临时堆栈使用显存寄存器实现,避免了频繁访问全局内存,保证了运行的高效性.

算法 1.　Update ($\{N_a^i,N_b^i\}$):对前线节点流 $S_f(t_0)$中单元节点$\{N_a^i,N_b^i\}$进行更新.

```
1.  while TRUE do
2.    if IsLeaf(N_a^i) AND IsLeaf(N_b^i) then
3.      if N_a^i not adjacent to N_b^i then
4.        S_f(t_1)+={N_a^i,N_b^i}//保存在前线节点流中
5.        if BoundingBoxTest(N_a^i,N_b^i)==
                                    NoOverlapping then
6.          S_t+={N_a^i,N_b^i}//保存在三角形对流前线中
7.        end if
8.      end if
9.    else
10.     if BoundingBoxTest(N_a^i,N_b^i)==
                                    NoOverlapping then
11.       S_f(t_1)+={N_a^i,N_b^i} //保存在前线节点流中
12.     else
13.       if IsLeaf(N_a^i) then   //递归检查子节点
14.         Push({N_a^i,N_b^i→left}),
            Push({N_a^i,N_b^i→right})
15.       else
16.         Push({N_a^i→left,N_b^i}),
            Push({N_a^i→right,N_b^i})
17.       end if
18.     end if
19.   end if
20.
21.   if empty() then return end if
22.   Pop(N_a^i,N_b^i)          //得到下一对节点
23. end while
```

3.4　并行流式登记

上一仿真时间点和当前仿真时间点的 BVTT 前线 $S_f(t_0)$和 $S_f(t_1)$分别表示为 $S_f(t_0)=\{N\}$和 $S_f(t_1)=\{M\}$.其中 $N\rightarrow M^*$,即 $S_f(t_0)$中的节点与 $S_f(t_1)$中的节点为一对多的映射关系.在碰撞检测中,BVTT 前线节点的长度变化较大(依赖于仿真场景).在最坏情况下,$\|S_f\|=O(n^2)$,这里 n 是场景中三角形的数目.因此算法需要一种高效的变长数据结构保存 BVTT 前线 $S_f(t_0)$和 $S_f(t_1)$.

与 CPU 不同,在图形硬件上高效实现变长数据结构十分困难,并且图形硬件的显存空间有限(AMD Radeon HD 5870 的显存空间为 1GB).如何

紧凑保存 $S_f(t_0)$ 和 $S_f(t_1)$，且不影响算法的并行性仍是难点. 目前文献中的解决方案包括：

(1) 使用较大的预留空间保存数据，然后使用 Prefix Sum 算子进行流式压缩[40-41]：这个方法需要较大的显存空间($O(n^2)$)，但能较好地支持并行操作，运行效率较高；

(2) 使用原子操作维护紧凑的存储：最新的图形硬件支持原子操作，即能实现多线程对寄存器对象的互斥访问. 通过使用原子操作，可以将数据保存在紧凑的空间中，但该方法将严重影响并行执行效率.

为了克服以上解决方案的缺陷，我们提出一种并行流式登记算法以实现变长数据结构，避免使用原子操作和冗余空间分配，同时保持运行的高效性. 在实际运行中，对比基于原子操作的实现方案，在不同的测试场景中，我们观察到 3.2～4.1 倍的加速.

在图形硬件上，柔性物体的碰撞检测任务以 Block 和线程为单位分两级并行执行. 在前线更新过程中，每个线程处理一个 BVTT 前线流数据单元的更新(参见算法 1)，并产生新的前线流数据单元. 常规方法使用原子操作收集所有新生成的前线流数据单元，而一次对全局内存的原子操作需要上百个时钟周期才能完成，每次原子操作都会造成大量线程被阻塞等待，计算性能受到严重影响. 我们首先提出了一种基于分段原子操作的并行流式登记算法(图 3)：对于属于同一个 Block 的所有线程，使用原子操作修改其分段存放空间的独立索引变量，保存新生成的前线流数据单元，最后使用 Prefix Sum 算子进行数据压缩(如图 3 所示). 假定 Block 数目为 n，每个 Block 中的并发线程数目 m，则算法使得 $n\times m$ 个线程对同一索引变量的竞争访问转化为 n 个独立的 m 个线程对同一索引变量的竞争访问，从而大大降低了线程耦合度，提高了并行执行效率.

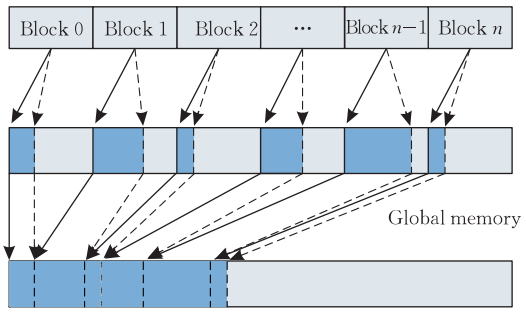

图 3 基于分段原子操作的并行流式登记

该算法可以进一步扩展，完全避免使用原子操作. 如图 4 所示，通过为每个线程分配一段空间，用于临时存放新生成节点，各线程间可以完全独立地并行执行. 由于各线程间无数据共享，线程内的计算流程是串行执行，因此不需要原子操作. 只有当某个线程新生成的节点数超过分配的临时空间的大小时，才使用原子操作放入最终存储空间中. 当各个线程完成新节点的生成后，各段新节点数据零星分布在临时空间中，需要使用 Prefix Sum 算子进行一次数据压缩. 由于每个线程生成的新节点在临时空间中的起止位置、数量已知，只需要为每段新节点计算 Prefix Sum，便能确定其在最终存储空间中的新起始位置，最后将数据并行移动. 对比于基于分段原子操作的方法，通过完全避免使用原子操作，并行流式登记算法的性能可获得 52% 的提升.

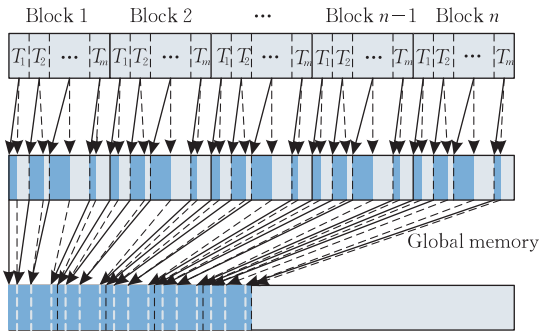

图 4 不使用原子操作的并行流式登记

3.5 生成特征对流

为了执行连续碰撞检测，三角形流中的每对潜在碰撞三角形将被分解为 15 个特征对(9 个 Edge-Edge 特征对和 6 个 Vertex-Face 特征对)，以进一步精确测试. 对于这些特征对，我们将使用特征包围盒[7]和非穿透性过滤器[11]进行剔除. 所有被剔除的特征将使用 Prefix Sum 算子从特征对流中剔除，保持特征对流的紧凑性. 在实践中，通过使用特征包围盒和非穿透性过滤器，特征对流的尺寸将被压缩至原尺寸的 4.2%～9.8%.

3.6 碰撞检测结果

对于压缩后的特征对流，其中的每个流数据单元(即 Edge-Edge 特征对或 Vertex-Face 特征对)将使用三次方程求解器计算其接触时间. 我们使用了牛顿迭代法在图形硬件上进行三次方程求解，若发生碰撞，则将结果记录在碰撞结果流中.

4 实现与结果

本节将描述算法的实现细节，并展示多个测试场景中碰撞检测算法的运行效率.

4.1　实验平台

本文算法已经在一台安装了 AMD Radeon HD 5870 图形硬件的 PC 机(Intel Q6600@2.4GHz,4GB 内存)上实现.开发工具为 Microsoft Visual Studio 2005,OpenCL 被用作图形硬件 API.AMD Radeon HD 5870 基于 40 纳米工艺制造,具有 1600 个流式处理单元,采用 1GB/256bit 规格的 GDDR5 显存,其核心频率和显存频率分别为 850 MHz 和 4800MHz.算法使用的包围盒为 k-DOP(特指 18-DOP).和 AABB 或包围球相比,k-DOP 提供了更高的剔除效率,并支持高效的动态重建.BVH 使用了最长轴中面分割法自顶向下递归构造.当模型发生变形时,使用了重新整理的策略维护紧凑的 BVH.本研究使用的图形硬件 AMD Radeon HD 5870 为 AMD(中国)公司捐赠.

4.2　测试场景

5 个各具特色的测试场景被用于测试算法的运行效率,它们分别来自不同的仿真试验,具体包括:

(1) Flamenco(49K 三角形,图 5).身着色彩绚丽西班牙舞裙的激情舞者.场景中将产生大量的自碰撞.

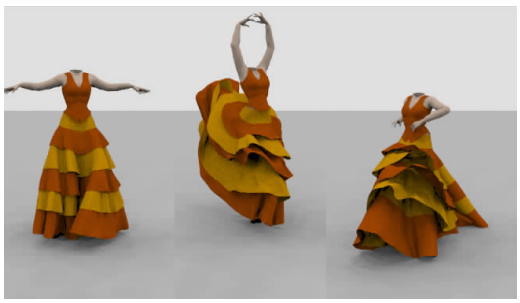

图 5　测试场景 Flamenco

(2) Cloth-ball(92K 三角形,图 6).一块方形布料落在球体上,并不断缠绕产生大量自碰撞.

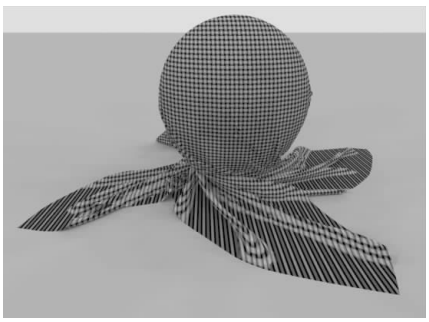

图 6　测试场景 Cloth-ball

(3) N-body(34K 三角形,图 7).由数百个随机运动的球体和圆锥体组成,它们相互撞击,产生大量物体间碰撞.

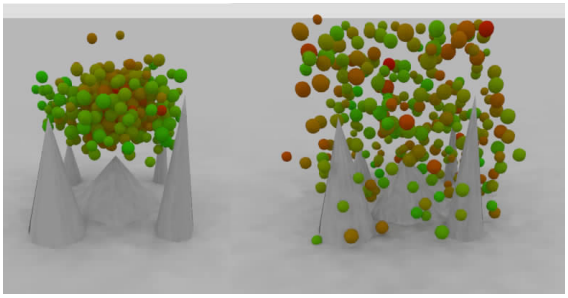

图 7　测试场景 N-body

(4) BART(4K 三角形,图 8).数百个随机运动的三角形相互撞击.这个高度复杂的场景包含了大量重叠的几何元素,它是 BVH 更新和碰撞查询的最坏情况之一.这个测试场景来自 BART 动态光线跟踪数据集[42].

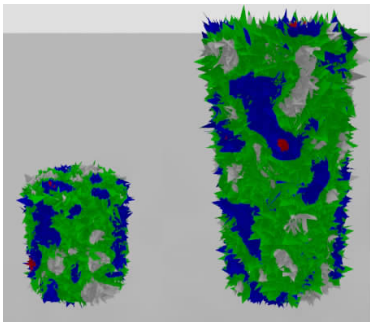

图 8　测试场景 BART

(5)(40K 三角形,图 9).身着柔顺长裙的舞者从站立到缓缓坐下,产生了大量的物体间和物体内碰撞.

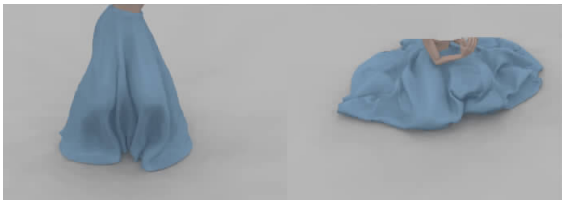

图 9　测试场景 Princess

以上测试场景都保留了一系列仿真时间点的关键帧数据.实验将在每个仿真时间点执行连续碰撞检测,并计算运动过程中的第一碰撞时间.

4.3　运行效率

我们分别采用了两种方案进行柔性物体的连续碰撞检测:图形硬件加速方案(AMD Radeon HD 5870)和基于 CPU 的单线程方案(Intel Q6600@2.4GHz),并比较这两种方案的运行效率.通过使用图形硬件进行加速,所有测试场景中柔性物体间的连续碰撞检测都获得了可观的性能提升.表 1 给出了不同测试场景分别获得的加速情况.实验结果验

证了经过流式映射的碰撞检测算法能够较好地适应图形硬件体系架构，并充分利用了众核处理器的大规模并行计算能力，从而取得了较理想的性能提升(9.2～11.4 倍). 图 10 给出了测试场景 Cloth-ball 中各核心所占 GPU 运行时间比例，其中 Edge-Edge 和 Vertex-Face 特征对的剔除和处理约占总处理时间 68%，而 BVTT 前线的动态更新约占总运行时间 26%，其它的工作，如包围盒更新、BVH 重新整理、孤集和特征对的精确测试等，所占比例约为 6%.

表 1 碰撞检测运行效率

测试场景	加速比	每帧运行时间/ms
Princess	10.8	6.4
Flamenco	9.6	33.6
N-body	10.6	15.6
Cloth-ball	9.2	40.5
BART	11.4	32.4

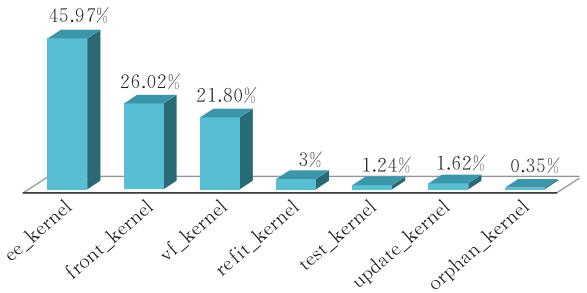

图 10 测试场景 Cloth-ball 中各核心所占GPU运行时间比例

5 比较与缺陷

本节将比较本文算法与前人工作，并指出本文算法的一些缺陷.

5.1 比 较

与利用图形处理器的光栅处理能力进行碰撞检测类算法[30-34]对比，本文算法克服了对于图像分辨率的依赖性，完全在对象空间中进行计算，确保了算法的精确性.

与基于图形处理器与 CPU 协同处理方式的算法[38-39]相比，本文算法的整个处理流程完全在图形处理器上进行，避免了图形处理器与 CPU 进行数据交换的效率瓶颈.

和本文工作相比，Lauterbach 等人基于图形硬件的碰撞检测工作[26]局限于离散碰撞检测，且无法对相邻三角形对的自碰撞检测进行有效剔除.

与基于多核处理器的算法[27]相比，本文算法采用了流式映射策略针对图形处理器架构进行了特别优化，能够充分利用图形处理器的强大处理能力. 其运行效率尽管略低于 16 核工作站的并行执行效率(10.1～13 倍加速[27])，但考虑到一台 16 核工作站的价格(HP DL580G5 服务器)约为 12 万元，而一块 AMD Radeon HD 5870 显卡的价格仅为 3500 元，基于图形处理器的解决方案显然具有较高的性价比.

5.2 缺 陷

本文算法仍存在缺陷. 首先，算法需要在图形硬件显存中存储 BVTT 前线，这对显存的容量提出了较高的要求. 其次，尽管通过使用并行流式登记算法可以有效支持变长数据结构，但该方法仍需要频繁随机读写全局内存，而没能充分利用访问速度较快的共享内存.

6 结论与未来工作

本文给出了一种图形硬件加速的柔性物体连续碰撞检测算法. 通过进行流式映射，算法将处理过程中的相关几何数据抽象为流数据，并设计了一组处理核心对这些流数据进行有效处理. 同时，本文提出了一种并行流式登记算法，可以在图形硬件上高效支持变长数据结构，解决了显存消耗过大和使用原子操作的缺陷. 这种并行流式登记算法还可以被应用于图形硬件上其它需要使用变长数据结构的应用，如粒子系统、破碎仿真等. 在实践中，本文方法大幅度提升了图形硬件平台下碰撞检测算法的性能，对比于 CPU 上的单线程优化实现，在多个测试场景的中获得了 9.2～11.4 倍加速.

还有许多问题值得深入研究. 首先，我们希望解决 5.2 节中的算法缺陷，其次，我们希望将该算法应用于其它大规模实时应用，如触摸渲染等，并采用 GPU-Cluster 的方式进一步提升算法性能. 最后，本文算法可以扩展到其它的逼近查询问题，如距离计算、穿透计算等.

参 考 文 献

[1] Govindaraju N, Knott D, Jain D, Kabul I, Tamstorf R, Gayle R, Lin M, Manocha D. Interactive collision detection between deformable models using chromatic decomposition. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH), 2005, 24(3): 991-999

[2] Provot X. Collision and self-collision handling in cloth model dedicated to design garment//Proceedings of the Computer Animation and Simulation'97. Budapest, Hungary, 1997:

177-189

[3] Redon S, Kheddar A, Coquillart S. Fast continuous collision detection between rigid bodies. Computer Graphics Forum (Proceedings of Eurographics), 2002, 21(3): 279-288

[4] Redon S, Kim Y J, Lin M, Manocha D. Fast continuous collision detection for articulated models//Proceedings of the ACM Symposium on Solid Modeling and Applications. Genoa, Italy, 2004: 145-156

[5] Zhang X, Redon S, Lee M, Kim Y J. Continuous collision detection for articulated models using Taylor models and temporal culling. ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2007, 26(3): 15

[6] Teschner M, Kimmerle S, Heidelberger B, Zachmann G, Raghupathi L, Fuhrmann A, Cani M P, Faure F, Magnenat-Thalmann N, Strasser W, Volino P. Collision detection for deformable objects. Computer Graphics Forum, 2005, 24(1): 61-81

[7] Hutter M, Fuhrmann A. Optimized continuous collision detection for deformable triangle meshes//Proceedings of the WSCG'07. Plzen-Bory, Czech Rep, 2007: 25-32

[8] Tang M, Yoon S, Manocha D. Adjacency-based culling for continuous collision detection. The Visual Computer (Proceedings of CGI08), 2008, 24(7-9): 545-553

[9] Curtis S, Tamstorf R, Manocha D. Fast collision detection for deformable models using representative-triangles//Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games. Redwood City, CA, USA, 2008: 61-69

[10] Tang M, Curtis S, Yoon S, Manocha D. ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. IEEE Transactions on Visualization and Computer Graphics, 2009, 15(4): 544-557

[11] Tang M, Manocha D, Tong R. Fast continuous collision detection using deforming non-penetration filters//Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. Washington DC, USA, 2010: 7-13

[12] Bradshaw G, O'Sullivan C. Adaptive medial-axis approximation for sphere-tree construction. ACM Transactions on Graphics, 2004, 23(1): 1-26

[13] Palmer I J, Grimsdale R L. Collision detection for animation using sphere-trees. Computer Graphics Forum, 1995, 14(2): 105-116

[14] van den Bergen G. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 1997, 2(4): 1-14

[15] Gottschalk S, Lin M, Manocha D. OBBtree: A hierarchical structure for rapid interference detection//Proceedings of the SIGGRAPH 1996. New York, NY, USA, 1996: 171-180

[16] Klosowski J, Held M, Mitchell J, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Transactions on Visualization and Computer Graphics, 1998, 4(1): 21-37

[17] Larsson T, Akenine-Möller T. A dynamic bounding volume hierarchy for generalized collision detection. Computers and Graphics, 2006, 30(3): 451-460

[18] Lauterbach C, Yoon S, Tuft D, Manocha D. RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs//Proceedings of the IEEE Symposium on Interactive Ray Tracing. Salt Lake City, USA, 2006: 39-46

[19] James D L, Pai D K. BD-Tree: Output-sensitive collision detection for reduced deformable models. ACM Transactions on Graphics (Proceedings of the SIGGRAPH), 2004, 23(3): 393-398

[20] Zachmann G, Weller R. Kinetic bounding volume hierarchies for deformable objects//Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications. Hong Kong, China, 2006: 189-196

[21] Otaduy M, Chassot O, Steinemann D, Gross M. Balanced hierarchies for collision detection between fracturing objects//Proceedings of the IEEE Virtual Reality. Singapore, 2007: 83-90

[22] Yoon S, Curtis S, Manocha D. Ray tracing dynamic scenes using selective restructuring//Proceedings of the ACM SIGGRAPH 2007 Sketches. San Diego, CA, USA, 2007: 55

[23] Garanzha K. Efficient clustered BVH update algorithm for highly-dynamic models//Proceedings of the IEEE Symposium on Interactive Ray Tracing. Los Angeles, California, 2008: 123-130

[24] Wald I. On fast construction of SAH based bounding volume hierarchies//Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing, Ulm University, Germany, 2007: 33-40

[25] Lauterbach C, Garland M, Sengupta S, Luebke D, Manocha D. Fast BVH construction on GPUs, Computer Graphics Forum (Proceedings of Eurographics 2009), 2009, 28(2): 375-384

[26] Lauterbach C, Mo Q, Manocha D. gProximity: Hierarchical GPU-based operations for collision and distance queries. Computer Graphics Forum (Proceedings of Eurographics 2010), 2010, 29(2): 419-428

[27] Tang M, Manocha D, Tong R F. MCCD: Multi-core collision detection between deformable models. Graphical Models, 2010, 72(2): 7-23

[28] Tang M, Manocha D, Tong R F. Parallel collision detection between deformable objects using SIMD instructions. Chinese Journal of Computers, 2009, 32(10): 2042-2051(in Chinese)
(唐敏, Dinesh Manocha, 童若锋. 基于 SIMD 指令的柔性物体并行碰撞检测. 计算机学报, 2009, 32(10): 2042-2051)

[29] Grand S L. Broad-Phase Collision Detection with CUDA. GPU Gems 3. New York: Addison-Wesley, 2007: 697-721

[30] Heidelgerger B, Teschner M, Gross M. Real-time volumetric intersections of deforming objects//Proceedings of the Vision, Modeling and Visualization 2003. Munich, Germany, 2003: 461-468

[31] Knott D, Pai K D. CinDeR: Collision and interference detection in real-time using graphics hardware//Proceedings of the Graphics Interface 2003. Halifax, Nova Scotia, Canada, 2003: 73-80

[32] Govindaraju N, Redon S, Lin M, Manocha D. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware//Proceedings of the ACM SIGGRAPH/EG Workshop on Graphics Hardware 2003, San Diego, CA, USA, 2003: 25-32

[33] Georgii J, Krüger J, Westermann R. Interactive collision detection for deformable and GPU objects. IADIS International Journal on Computer Science and Information Systems, 2007, 2(2): 162-180

[34] Gerß A, Guthe M, Klein R. GPU-based collision detection for deformable parameterized surfaces. Computer Graphics Forum (Proceedings of EUROGRAPHICS 2006), 2006, 25(3): 497-506

[35] Morvan T, Reimers M, Samset E. High performance GPU-based proximity queries using distance fields. Computer Graphics Forum, 2008, 27(8): 2040-2052

[36] Sud A, Otaduy M A, Manocha D. DiFi: Fast 3D distance field computation using graphics hardware. Computer Graphics Forum (Proceedings of Eurographics), 2004, 23(3): 557-566

[37] Sud A, Govindaraju N, Gayle R, Kabul I, Manocha D. Fast proximity computation among deformable models using discrete Voronoi diagrams. ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2006, 25(3): 1144-1153

[38] Zhang X, Kim Y J. Interactive collision detection for deformable models using streaming AABBs. IEEE Transactions on Visualization and Computer Graphics, 2007, 13(2): 318-329

[39] Kim D S, Heo J P, Huh J, Kim J, Yoon S. HPCCD: Hybrid parallel continuous collision detection. Computer Graphics Forum (Proceedings of Pacific Graphics 2009), 2009, 28(7): 1791-1800

[40] Horn D. Stream reduction operations for GPGPU applications, GPU Gems 2. New York: Addison-Wesley, 2005: 573-58

[41] Harris M, Sengupta S, Owens J D. Parallel Prefix sum (scan) With CUDA. GPU Gems 3. New York: Addison-Wesley, 2007: 851-876

[42] Lext J, Assarsson U, Moller T. A benchmark for animated ray tracing. IEEE Computer Graphics and Applications, 2001, 21(2): 22-31

TANG Min, born in 1974, Ph. D., associate professor. His main research interests include collision detection, GPU-based algorithm acceleration, solid modeling.

LIN Jiang, born in 1987, M. S. candidate. His research interests include GPU-based algorithm acceleration, bioinformatics.

TONG Ruo-Feng, born in 1969, Ph. D., professor. His main research interests include image/video processing, geometry model.

Background

The paper focuses on the collision detection problem among deformable objects, which is a key problem in the field of collision detection and has been studies by many researchers. Currently, most fast algorithms are utilizing computing power of multi-core processors. The algorithm has achieved real-time processing speed with graphics hardware. The project is supported by the Natural Science Foundation of China under grant No. 60803054, which aims to perform fast collision detection between deformable objects in complex simulation scenes. The research result of this paper solved a key problem of the project: How to effectively accelerate collision detection by utilizing graphic hardware? The research group has worked on this topic for several years, and achieved fruitful results. Some of the papers published recently.