基于 SIMD 指令的柔性物体并行碰撞检测

唐 敏[1)] MANOCHA Dinesh[2)] 童若锋[1)]

[1)](浙江大学计算机学院 杭州 310027)

[2)](北卡罗莱纳大学教堂山分校,教堂山,北卡罗莱纳州,美国 27599)

摘 要 复杂场景中柔性物体间的碰撞检测依然难以满足交互设计的要求.为了提高处理速度,文中给出了一种充分利用现代 CPU 的并行处理能力的碰撞检测算法.算法基于两方面的并行处理:即基于 SIMD 指令的指令级并行处理和基于多线程的任务级并行处理.算法给出了一种针对 SIMD 指令特别优化的 k-DOP 模型——SIMD-DOP,从理论上分析了该包围盒的高效性,并与常规的 16-DOP 和 24-DOP 进行了运行效率对比.通过使用 SIMD-DOP 同时在多核间进行负载均衡,算法获得了优化的并行加速.文中算法已经在一台 16 核工作站上针对一组复杂测试场景进行了验证.

关键词 连续碰撞检测;柔性物体;SIMD 指令;并行碰撞检测;包围盒层次结构

中图法分类号 TP391 **DOI 号**: 10.3724/SP.J.1016.2009.02042

Parallel Collision Detection Between Deformable Objects Using SIMD Instructions

TANG Min[1)] MANOCHA Dinesh[2)] TONG Ruo-Feng[1)]

[1)](*Department of Computer Science, Zhejiang University, Hangzhou* 310027)

[2)](*Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill*, 27599, *USA*)

Abstract Continuous collision detection among deformable objects in complex scenes is still hard to fulfill the demand for interactive design. To speedup processing, a collision detection algorithm which can fully exploit the parallel computing capability of modern CPU is proposed. The algorithm is based on a two-level parallel processing: SIMD instruction based instruction-level parallel processing and multi-thread based task-level parallel processing. A SIMD instruction friendly k-DOP, SIMD-DOP, is designed. It has been analyzed, and compared with 16-DOP and 24-DOP in running efficiency. By using SIMD-DOP and balancing computing loads among multi-cores, optimized accelerations have achieved. The algorithm has been implemented on a workstation with 16 cores, and tested by using several complex benchmarks.

Keywords continuous collision detection; deformable models; SIMD instructions; parallel collision detection; bounding volume hierarchies

1 引 言

在现实世界中,存在着大量的柔性物体,即该物体在外力作用下将产生形变.典型的柔性物体包括布料、毛发、肌肉等.随着计算机处理能力的不断提升,柔性物体已经取代了刚体、铰链模型而成为基于物理的仿真、虚拟环境、机器人运动规划等领域的重

收稿日期:2009-07-15;最终修改稿收到日期:2009-09-02.本课题得到国家自然科学基金(60803054)、教育部-英特尔信息技术专项科研基金项目(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、多核计算.E-mail: tang_m@zju.edu.cn. **MANOCHA Dinesh**,男,1963 年生,博士,教授,主要研究领域为计算机图形学. **童若锋**,男,1969 年生,博士,教授,主要研究领域为计算机图形学、图形与视频处理.

要建模工具.在仿真过程中,柔性物体不断发生形变并可能因发生破裂、爆炸而引发拓扑结构的改变.

为了获得物理正确或近似正确的仿真结果,仿真系统需要检测出碰撞发生的第一时间并做出相应的反应.由于柔性物体的动态属性,碰撞检测变得十分困难,除了需要考虑物体间的碰撞情况外,还需要计算柔性物体形变过程中产生的自碰撞情况.以图 1 中的仿真场景为例,柔性物体在发生撞击后破裂为一堆碎片,产生了大量的物体间碰撞和物体内自碰撞.连续碰撞检测方法[1]被用于检测物体在连续变形过程中可能发生的碰撞,并得到第一接触时间.对于一对三角形间的连续碰撞检测,即使是最简单的变形路径(顶点线性插值)也需要进行 15 次元素测试(6 次 Vertex/Face 测试、9 次 Edge/Edge 测试).每次元素测试需要求解一个单变量三次代数方程.对于复杂柔性物体仿真场景(10K～100K 数量级的三角形),实时碰撞检测依然十分困难.

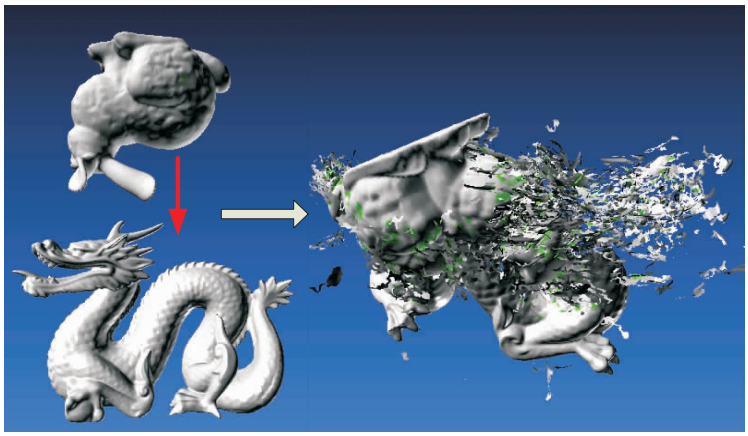

图 1　仿真场景中破裂柔性物体的碰撞情况(柔性物体在外力撞击下破裂为一堆碎片,从而产生了大量物体间的碰撞和物体内部的自碰撞)

近年来,计算硬件的性能取得了飞速提升.尽管单核处理器的摩尔定律已经失败,但多核处理器的出现使得市场主流产品的性能依然飞速发展.通过集成数目越来越多的处理单元,CPU 已经从单核发展到双核、四核、多核乃至众核.但是,简单的算法实现并不能在多核处理器上获得性能提升,只有将计算任务有效分解,使之能在各个核心上并行执行,才能从多核处理器上获益.

本文的目标是充分利用现代 CPU 的并行处理能力对碰撞检测进行加速.现代 CPU 支持两个层次的并行处理能力:指令级并行和任务级并行.本文算法将充分利用这两种并行处理能力实现复杂场景中柔性物体的实时连续碰撞检测.

本文第 2 节对柔性物体的包围盒层次结构、基于 SIMD 指令的碰撞检测和并行碰撞检测领域的前人工作进行综述;第 3 节介绍算法总体流程;第 4 节详细论述指令级碰撞检测加速算法;第 5 节给出任务级碰撞检测加速算法.实现细节和实验结果参见第 6 节;在第 7 节,将本文算法和前人算法进行对比,并对本文算法缺陷进行分析.

2　相关工作

2.1　柔性物体的包围盒层次结构

包围盒层次结构(BVH)是进行高效碰撞检测的重要工具.各种 BVH,如球树[2-4]、轴对齐包围盒(AABB)树[5]、定向包围盒(OBB)树[6]、离散定向多面体(k-DOP)树[7]等,已经被应用于刚体和柔性物体.这些层次结构可以自顶而下或自底向上进行构造.近期许多研究者都在关注柔性物体 BVH 的快速更新技术,包括了重新整理、动态或选择性重构[8-10].

对于柔性物体,除了需要考虑包围盒的紧凑性外,包围盒整理和构造的效率也至关重要.对于简单的包围盒如球体、AABB,其整理和构造十分快捷,但紧凑性较差,复杂的包围盒 OBB,虽然较为紧凑,但整理和构造计算量较大.作为两者的折中,k-DOP 兼具紧凑性和高效性,因此常被用于柔性物体的包围盒层次结构.

2.2　基于 SIMD 指令的碰撞检测加速

现代 CPU 提供了 SIMD 指令,支持对于保存在

矢量寄存器上多个标量数据(4 个、8 个或 16 个整数或浮点数)的并行操作.如果代码执行时能充分利用 SIMD 指令的并行处理能力,在理论上可以获得 4 倍～16 倍的性能提升.

来自各个制造厂商的 CPU 都提供了 SIMD 指令集,其中包括 Intel 公司 SSE、SSE2、SSE3、SSE4 指令集、AMD 公司的 3DNow!指令集、Motorola 和 IBM 公司的 AltiVec 指令集.IBM 公司的 CELL 处理器和 Sony 公司的 Play Station2 也都提供了 SIMD 指令机制.

许多研究者使用了 SIMD 指令进行计算加速.Wald 等人[11]使用了 SIMD 指令同时进行 4 条光线和一个三角形的求交计算,对比未使用 SIMD 指令的实现获得了 3.5～3.7 倍的加速.Ericson[12]使用了 SIMD 指令同时进行 4 路 sphere-sphere、sphere-AABB、AABB-AABB 的测试.Lin[13]使用了 SIMD 指令对刚体碰撞检测过程中 OBB-OBB 测试进行加速.

随着多核并行计算的不断普及,如何将指令级并行计算与任务级并行计算有机结合,充分挖掘现代 CPU 的计算潜力已经成为一个研究热点.

2.3 连续碰撞检测

许多高效的连续碰撞检测算法已经分别针对刚体[14]、铰链模型[15-16]和柔性物体[17-19]而设计.对于柔性物体,算法线性插值顶点模拟形变,使用层次结构剔除冗余,并执行元素测试计算第一接触时间.

2.4 并行碰撞检测

碰撞检测的核心步骤是层次结构遍历,这对应于 BVTT 的深度优先遍历.许多研究者设计了并行算法对深度优先遍历进行加速,这些工作可以用于碰撞检测.Rao 和 Kumar[20]指出并行深度优先搜索算法的效率主要受到负载均衡策略和硬件架构的影响.Kumar 和 Grama[21]对几种负载均衡技术的可扩展性进行了分析.Reinefeld 和 Schnecke[22]对比了两种深度优先搜索方法的负载均衡策略,并提出了一种使用固定大小工作包的细粒度方案.Kitamura 等人[23]使用了静态和动态负载均衡策略分别对碰撞检测进行加速,并指出处理器间的通信是效率受限的主要原因.Assarsson 和 Stenström[24]对于一些工业仿真场景中的 CAD 刚体模型间碰撞检测过程在 8 个处理器上获得了 3 倍加速,并验证了非置锁负载均衡策略的优越性.Grinberg 等人[25]使用了预计算方式的静态任务划分机制提取并行性.

研究者研究了多核架构下或分布式架构下的物理仿真快速算法.Chen 等人[26]使用了兆级处理器对于离线物理仿真进行加速.Thomaszewski 等人[27]和 Selle 等人[28]对布料仿真过程中的碰撞检测问题在分布式内存架构下进行了加速.Thomaszewski 等人[29]使用了多线程技术对布料仿真过程中的隐式时间积分和碰撞处理进行了加速.

研究者已经提出一些算法利用市场主流处理器的并行处理能力对碰撞检测进行加速.其中包括了基于 GPU 的算法[30-31],它们利用众核 GPU 的光栅处理能力加速干涉计算.最新的工作还包括了使用多核 CPU[32]或众核 GPU[33-34]进行 BVH 的快速计算.唐敏等人[35]给出了基于多核处理器的任务级并行算法,Kim 等人[36]使用了多核处理器和 GPU 的混合算法对碰撞检测进行加速.

3 算法概述

柔性物体的连续碰撞检测算法的总体流程如图 2 所示.该算法由以下 3 个阶段组成:

(1)更新模型和 BVH.柔性物体由三角形网格表示,仿真场景中所有的柔性模型使用一个 BVH 进行组织管理.该阶段进行顶点坐标线性插值进行模型的动态更新,在此基础上,进行 BVH 的重新计算,使其保持紧凑.

(2)BVH 遍历与包围盒测试.该阶段自顶向下遍历 BVH,进行包围盒的重叠测试,剔除空间距离较大的模型区域.

(3)精确碰撞计算.对通过了包围盒测试的三角形对,使用元素测试进行精确计算,得到碰撞发生的第一接触时间.

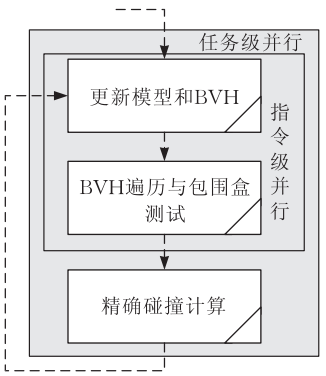

图 2 并行算法的流程(算法分为 3 个阶段:更新模型和 BVH、BVH 遍历与包围盒测试、精确碰撞计算.其中整个算法使用了任务级并行策略,同时对前两个阶段使用了指令级并行策略)

通过进行任务分解,算法的各个阶段(更新模型和 BVH、BVH 遍历与包围盒测试、精确碰撞计算)得以在多核处理器上并行展开,作为任务级并线的补充,指令级并行算法被用于对以上前两个阶段进行加速.图 3 给出了不使用两级并行策略时,对于不同的测试场景(参见第 6.2 节),串行算法各阶段的运行时间比例.从图中可以看出,前两个阶段的运行时间所占比例很大(54%～87%),因此使用 SIMD 指令对这两个阶段进行加速对总体性能的提升十分重要.

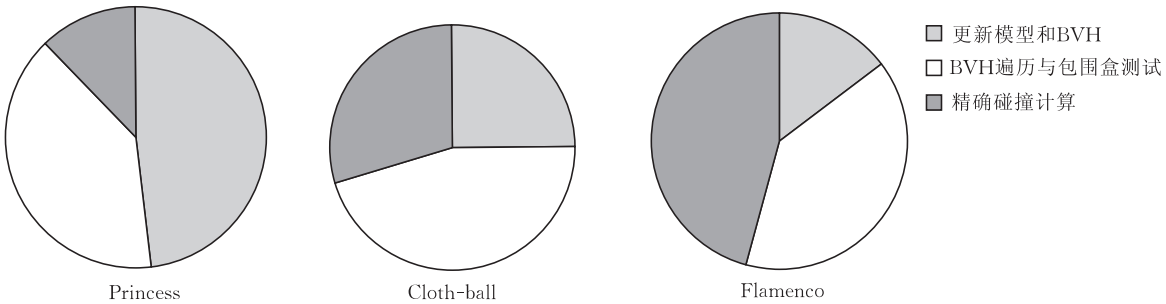

图 3　对于不同的测试场景(参见第 6.2 节),未使用并行策略时,串行算法各阶段运行时间所占总运行时间的比例

第 4 节和第 5 节将分别给出算法的指令级并行策略和任务级并行策略.算法通过使用面向 SIMD 指令优化的 k-DOP 模型——SIMD-DOP,实行了指令级并行处理(第 4 节),同时通过使用基于 BVTT 前线的并行更新策略,实现了任务级并行处理(第 5 节).

4　SIMD 指令优化的 k-DOP 模型——SIMD-DOP

4.1　基于 SIMD 指令的 k-DOP 优化

对于柔性物体,由于模型不断发生形变,因此高效的包围盒更新机制十分重要.k-DOP 是由多组平行面定义的 3D 空间封闭区域.每组中的两个平行平面限定了物体在平面法向方向上的延伸范围.如图 4 所示,对于图中的模型,k-DOP(a)提供了相对 AABB(b)更加紧凑的包围,并且与 OBB(c)相比进行构造和整理的代价较小.

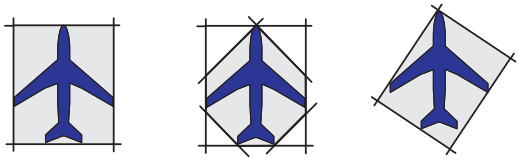

图 4　各种包围盒的实例(AABB、8-DOP、OBB)

目前最常用的 k-DOP 包括 6-DOP(即 AABB)、14-DOP、18-DOP 和 26-DOP.但为了得到 SIMD 指令优化的 k-DOP,我们提出了一种 SIMD-DOP,它是 16-DOP 的 SIMD 指令优化版本.SIMD-DOP 使有以下 8 个投影方向定义:(1,0,0)、(0,1,0)、(0,0,1)、(1,1,0)、(1,0,1)、(0,1,1)、(1,−1,0)、(1,0,−1).该 8 个投影方向的最大值和最小值共 16 个浮点值保存在 4 个 SIMD 浮点向量中(如图 5 所示).通过使用 SIMD 浮点向量保持所有数据,我们可以使用 SIMD 指令对 SIMD-DOP 的重建、整理、重叠测试进行加速,而这些操作是“更新模型和 BVH”和“BVH 遍历与包围盒测试”两个阶段的核心操作.

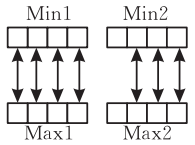

图 5　保存在浮点向量中的 SIMD-DOP 数据(8 个投影方向的最大值和最小值共 16 个浮点值保存在 4 个 SIMD 浮点向量 Min1、Min2、Max1、Max2 中)

4.2　数据表示

常规的 16-DOP 使用 16 个分量的浮点数数组进行表示:

```
float _dist[16];
```

而 SIMD-DOP 则使用 4 个浮点向量保存 16 个最大值和最小值:

```
struct {
    __m128 _max[2];
        //packing all the 16 float data into 4 __m128
    __m128 _min[2];
} v;
```

4.3　基于 SIMD 指令的重建和整理

(1)重建操作

常规的 16-DOP 在进行重建时,需要重新计算顶点的包围盒:

```
kDOP16 &operator+=(vec3f &p) {
```

```
    for (int i=0; i<3; i++)
      MinMax(p[i], _dist[i], _dist[8+i]);

    float pd[5];
    getDistances(p, pd);
    for (int i=0; i<5; i++)
      MinMax(pd[i], _dist[3+i], _dist[11+i]);

    return *this;
  }
```

其中函数 getDistances()用来得到顶点在 X、Y、Z 轴以外 5 个方向上的投影，上面的代码中共调用了 8 次函数 MinMax()，用来进行比较同时得到最大值和最小值，即共 16 次比较指令.

而使用 SIMD 指令优化后的重建代码如下：

```
kDOP16 &operator+=(vec3f &p) {
  float d[5];
  getDistances(p, d);

  __m128 t=_mm_setr_ps(p[0], p[1], p[2], d[0]);
  v._min[0]=_mm_min_ps(v._min[0], t);
  v._max[0]=_mm_max_ps(v._max[0], t);

  t=_mm_set_ps(d[1], d[2], d[3], d[4]);
  v._min[1]=_mm_min_ps(v._min[1], t);
  v._max[1]=_mm_max_ps(v._max[1], t);

  return *this;
}
```

其中共调用了_mm_min_ps、_mm_max_ps、_mm_setr_ps 这三条 SIMD 指令 6 次，即可完成 SIMD-DOP 的重建.

(2)整理操作

在底层包围盒重建的基础上，我们还需要自底向上进行整理. 常规的 16-DOP 的整理操作如下：

```
kDOP16 &operator+=(const kDOP16 &b) {
  for (int i=0; i<8; i++) {
    _dist[i]=MIN(b._dist[i], _dist[i]);
    _dist[i+8]=MAX(b._dist[i+8], _dist[i+8]);
  }
  return *this;
}
```

其中供调用 MIN、MAX 比较操作 16 次.

而使用 SIMD 优化后的整理代码如下：

```
SIMD-DOP &operator+=( SIMD-DOP &b) {
  v._min[0]=_mm_min_ps(v._min[0],
              b.v._min[0]);
  v._min[1]=_mm_min_ps(v._min[1],
              b.v._min[1]);
  v._max[0]=_mm_max_ps(v._max[0],
              b.v._max[0]);
  v._max[1]=_mm_max_ps(v._max[1],
              b.v._max[1]);
  return *this;
}
```

其中共调用_mm_min_ps、_mm_max_ps 这两条 SIMD 指令 4 次.

4.4 基于 SIMD 指令的重叠测试

为了判别两个 16-DOP 是否有可能发生碰撞，需要在 8 个投影方向进行重叠测试. 常规的 16-DOP 重叠测试的代码如下：

```
bool overlaps(const kDOP16&b) {
  for (int i=0; i<8; i++) {
    if (_dist[i] >b._dist[i+8]) return false;
    if (_dist[i+8] <b._dist[i]) return false;
  }
  return true;
}
```

最坏情况下需要进行 16 次比较操作.

而对于 SIMD-DOP 进行重叠测试的代码如下：

```
bool overlaps(SIMD-DOP&b) {
  if (_mm_movemask_ps(_mm_cmpgt_ps(v._min[0],
      b.v._max[0]))) return false;
  if (_mm_movemask_ps(_mm_cmplt_ps(v._max[0],
      b.v._min[0]))) return false;
  if (_mm_movemask_ps(_mm_cmpgt_ps(v._min[1],
      b.v._max[1]))) return false;
  if (_mm_movemask_ps(_mm_cmplt_ps(v._max[1],
      b.v._min[1]))) return false;
  return true;
}
```

最坏情况下也只需要调用 8 次 SIMD 指令. 通过调用_mm_cmpgt_ps 和_mm_cmplt_ps 指令，同时可以进行 4 路比较操作，然后使用_mm_movemask_ps 判别出如果任意一路重叠测试失败，则可以立即得出 SIMD-DOP 不可能发生碰撞的结论. 前人工作中[12]使用了 SIMD 指令同时进行 4 路包围盒测试，但这种方法难以与任务级的并行碰撞检测集成. 本文的新方法克服了以上缺陷，与任务级并行形成有效互补，从而提高了整体并行加速比. 附录 A 中给出了 SIMD-DOP 具体实现的 C++伪代码.

5 基于多线程的并行连续碰撞检测

我们使用了基于前线分解的方法[35]对碰撞检测任务进行细粒度分解，并将子任务使用负载均衡策略分配到多核处理器的各个处理核心上并行执行. 算法总体流程如下：

在初始时间点 t_0,构造初始 BVH,并通过遍历构造 BVTT 前线在每个时间点 t_i:

更新模型顶点坐标、更新 BVH;

对 BVTT 前线节点进行并行更新;

精确计算碰撞.

算法使用了 SIMD-DOP 作为包围盒,通过使用 SIMD 指令对 BVH 构造与更新进行了加速,同时在 BVTT 前线节点的更新过程中,使用了 SIMD 指令对包围盒重叠测试进行了加速. 如图 6 所示,通过对 BVTT(包围盒测试树)前线进行增量式更新,BVTT 前线节点的更新任务被分配到多核上并行执行. 在碰撞检测过程中,指令级并行性和任务级并行性都得到了充分利用,形成互补,从而使得算法总体效率得到大幅提升.

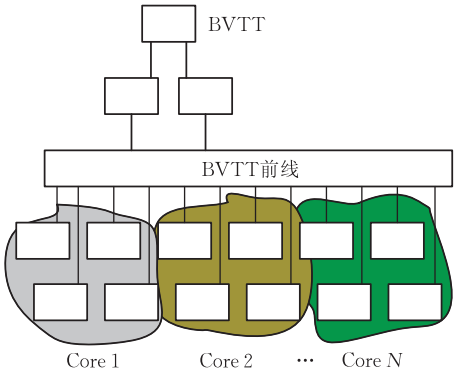

图 6　并行连续碰撞检测(通过对 BVTT 前线节点在多核上进行并行更新[35],碰撞检测任务实现了任务级并行处理,同时使用 SIMD-DOP 实现了指令级并行处理)

6　实现和结果

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.

6.1　实验平台

本文算法已经在一台具有 16 个核心(4 颗 Intel Xeon X7350 4 核处理器,2.93GHz)的 PC 机(16GB 内存)上实现. 开发工具为 Microsoft Visual Studio 2005,VC++提供的 SSE intrinsic 函数被用于 SIMD 指令开发,使用的 SIMD 指令为 Intel SSE/SSE2 指令. OpenMP 被用于多线程开发. 使用的包围盒为 SIMD-DOP. 和 AABB 或球相比,SIMD-DOP 提供了更高的剔除效率. BVH 使用了最长轴中面分割法自顶向下递归构造,并在仿真过程中不断重新整理.

6.2　测试场景

5 个各具特色的测试场景(图 7)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) Balls(34K 三角形,参见图 7(a)):由数百个随机运动的球体和圆锥体组成,它们相互撞击,产生大量物体间的碰撞.

(2) Princess(40K 三角形,参见图 7(b)):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.

(3) Cloth-ball(92K 三角形,参见图 7(c)):一块方形布料落在球体上,并不断缠绕产生大量自碰撞.

(4) BART(4K 三角形,参见图 7(d)):数百个随机运动的三角形相互撞击. 这个高度复杂性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[37].

(5) Flamenco(49K 三角形,参见图 7(e)):身着色彩绚丽西班牙舞裙的激情舞者. 这段测试场景将产生大量的自碰撞.

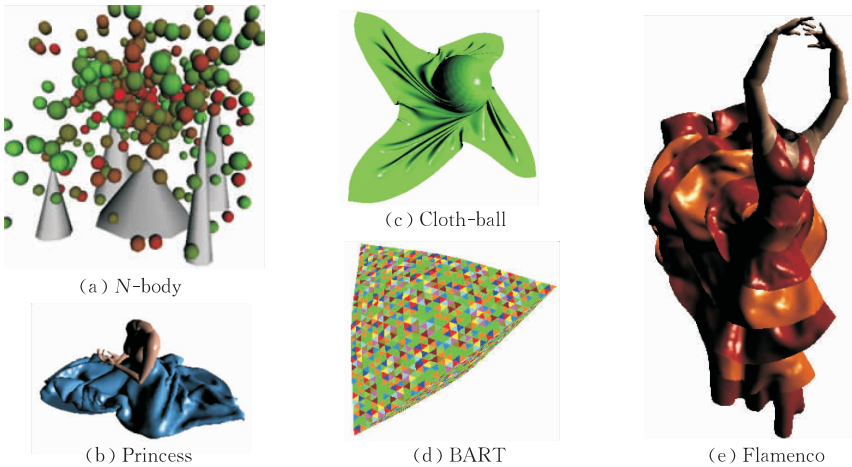

图 7　算法使用的测试场景(5 个各具特色的测试场景被用于测试算法的运行效率,它们分别来自不同的仿真试验. 三角形数目为 4K～92K)

以上测试场景都保留了一系列仿真时间点.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.

6.3 运行结果

图 8～12 给出了对以上测试场景进行并行连续碰撞检测的运行效率情况.和不使用 SIMD-DOP 的版本对比,通过使用 SIMD 指令实现指令级并行处理,系统获得了较大的性能提升(单核上最高可达 30.6%).而且该指令级并行处理和任务级并行处理形成正向互补,使得系统中多核情况下也获得了理想的加速比(16 核上最高可达 11.5%).对于在使用核心数目增加的情况下,使用 SIMD 指令获得的加速比下降的情况,原因在于多线程充分并行执行时将造成矢量寄存器的访问竞争,从而影响了 SIMD 指令的执行效率.我们使用了 Intel VTune 软件对系统运行数据进行了收集和分析,也观察到了同样的现象.

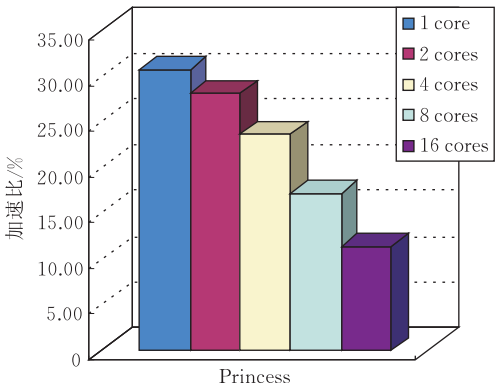

图 8 测试场景 Princess 使用 SIMD 指令获得的加速比(在单核情况下可获得 30.6%的性能加速,加速比随着使用核心数目的增加而降低,在 16 核下获得 11.5%的性能加速)

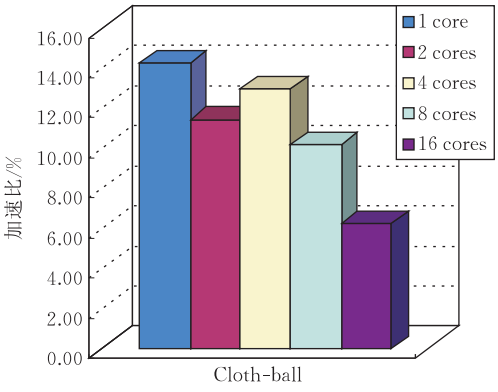

图 9 测试场景 Cloth-ball 使用 SIMD 指令获得的加速比(在单核情况下可获得 14.3%的性能加速,加速比随着使用核心数目的增加而降低,在 16 核下获得 6.3%的性能加速)

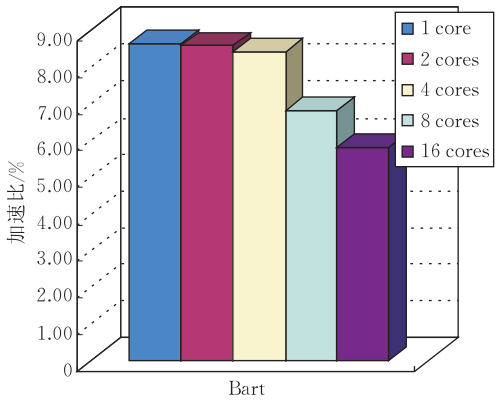

图 10 测试场景 BART 使用 SIMD 指令获得的加速比(在单核情况下可获得 8.7%的性能加速,加速比随着使用核心数目的增加而降低,在 16 核下获得 5.8%的性能加速)

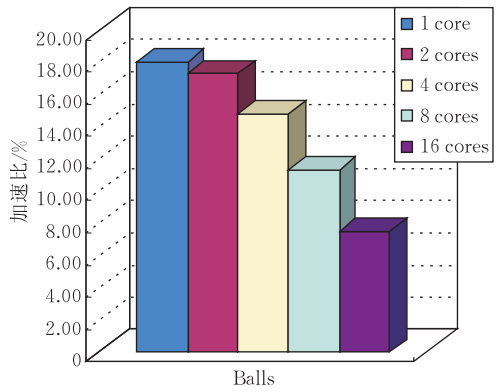

图 11 测试场景 Balls 使用 SIMD 指令获得的加速比(在单核情况下可获得 18%的性能加速,加速比随着使用核心数目的增加而降低,在 16 核下获得 7.5%的性能加速)

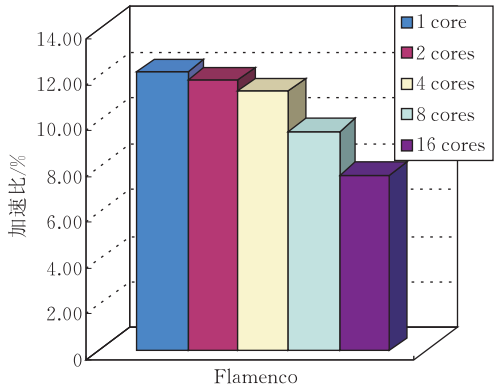

图 12 测试场景 Flamenco 使用 SIMD 指令获得的加速比(在单核情况下可获得 12.1%的性能加速,加速比随着使用核心数目的增加而降低,在 16 核下获得 7.6%的性能加速)

图 13 给出了在单核情况下分别使用 24-DOP、16-DOP、SIMD-DOP 所获得的总体性能对比,使用 SIMD-DOP 可获得 25%～45%的性能提升.

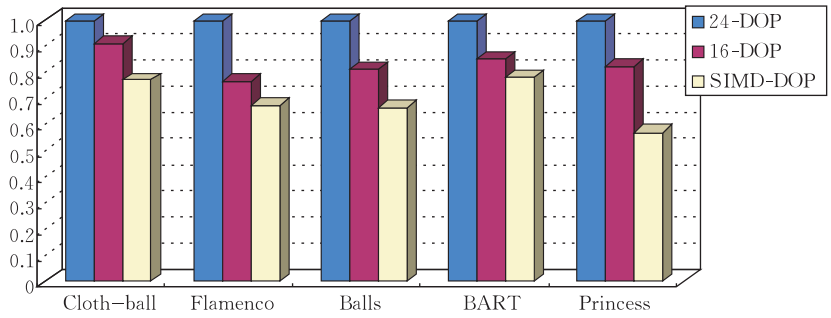

图 13　不同包围盒对总体性能的影响(在单核情况下分别使用 24-DOP、16-DOP、SIMD-DOP 所获得的总体性能对比,使用 SIMD-DOP 可获得 25%～45%的性能提升)

7　比较和缺陷

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.

7.1　比　较

前人在多核平台下的并行碰撞检测算法[35-36]都只利用了多核处理器的任务并行性,因此本文的指令并行算法将和以上工作形成互补.而前人的 SIMD 指令加速算法[11-13]则无法有效地与多核平台集成,虽然在单线程模式下获得了较好的加速,但在多核平台下特别是核心数目较大时则可能造成性能下降.本文算法克服了这些算法的缺陷,可以有效地挖掘多核处理器的指令并行性与任务并行性,在核心数目较大的情况下也提供了性能的提升.

7.2　缺　陷

本文算法仍存在缺陷,主要问题在于核心数目较大时,由于矢量寄存器的竞争访问造成 SIMD 指令获得的加速比不断下降.这个问题需要通过设计更加精细的内存访问模式从而减少寄存器竞争来解决.

8　结论和下一步工作

本文给出了一种面向 SIMD 指令优化的 k-DOP 模型——SIMD-DOP,通过使用 SIMD 指令对该模型的构造、整理、重叠测试进行加速,充分利用了现代 CPU 的指令并行性.通过使用 SIMD-DOP,一种新的柔性物体并行连续碰撞检测算法被设计,该算法利用了多核处理器的任务级并行性和指令级并行性,形成优势互补.通过使用一组复杂测试场景进行实验,在单核上获得了高达 30%的性能提升,在 16 核上也获得了高达 11%的性能提升.

未来的研究工作将包括将本文算法推广到 Intel 最新的 Larrabee 芯片,该处理器提供了 16 路的 SIMD 并行处理能力.同时,针对图形处理器进行算法优化也将是十分有趣的工作.

致　谢　感谢林江同学为本文制作了部分模型和图片!

参　考　文　献

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

[2] Hubbard P M. Interactive collision detection//Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality. San Jose, CA, USA, 1993: 24-31

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

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

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

[6] Gottschalk S, Lin M C, Manocha D. Obbtree: A hierarchical structure for rapid interference detection//Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96). New York, NY, USA, 1996: 171-180

[7] 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

[8] 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, UT, 2006: 39-46

[9] Yoon S, Curtis S, Manocha D. Ray tracing dynamic scenes using selective restructuring//Proceedings of Eurographics Symposium on Rendering. Grenoble, France, 2007: 73-84

[10] Wald I. On fast construction of SAH based bounding volume hierarchies//Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing. Ulm, Germany, 2007: 51-62

[11] Wald I, Philipp S, Carsten B, Markus W. Interactive rendering with coherent ray tracing//Proceedings of the EUROGRAPHICS 2001. Manchester, UK, 2001: 153-164

[12] Ericson C. Real-Time Collision Detection. New York: Morgan Kaufmann, 2004

[13] Lin M C, Gregory A, Ehmann S, Gottschalk S, Taylor R. Contact determination for real-time haptic interaction in 3D modeling, editing and painting//Proceedings of the 1999 Workshop for PHANToM User Group. New York, USA, 1999: 43-50

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

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

[16] 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), 2007, 26(3): 15

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

[18] 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(SI3D'08). Redwood City, CA, USA, 2008: 61-69

[19] Tang M, Curtis S, Yoon S-E, 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

[20] Rao V N, Kumar V. Parallel depth-first search, Part I: Implementation. International Journal of Parallel Programming, 1987, 16(6): 479-499

[21] Kumar V, Grama A Y. Scalable load balancing techniques for parallel computers. Journal of Parallel and Distributed Computing, 1994, 22(1): 60-79

[22] Reinefeld, Schnecke V. Work-load balancing in highly parallel depth-first search//Proceedings of the Scalable High Performance Computing Conference. Knoxville, TN, USA, 1994: 773-780

[23] Kitamura Y, Smith A, Takemura H, Kishino F. Parallel algorithms for real-time colliding face detection//Proceedings of the RO-MAN'95, 4th IEEE International Workshop on Robot and Human Communication. Tokyo, 1995: 211-218

[24] Assarsson U, Stenström P. A case study of load distribution in parallel view frustum culling and collision detection//Proceedings of the 7th International Euro-Par Conference Manchester, UK, Lecture Notes in Computer Science 2150, 2001: 663-673

[25] Grinberg I, Wiseman Y. Scalable parallel collision detection simulation//Proceedings of Signal and Image Processing. Honolulu, USA, 2007: 31-42

[26] Chen Y-K, Chhugani J, Hughes C J, Kim D, Kumar S, Lee V W, Lin, Nguyen A D, Sifakis E, Smelyanskiy M. High-performance physical simulations on next-generation architecture with many cores. Intel Technology Journal, 2007, 11(3): 121-128

[27] Thomaszewski B, Blochinger W. Physically based simulation of cloth on distributed memory architectures. Parallel Computing, 2007, 33(6): 377-390

[28] Selle A, Su J, Irving G, Fedkiw R. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. IEEE Transactions on Visualization and Computer Graphics, 2009, 15(2): 339-350

[29] Thomaszewski B, Pabst S, Blochinger W. Special section: Parallel graphics and visualization: Parallel techniques for physically based simulation on multi-core processor architectures. Computers and Graphics, 2008, 32(1): 25-40

[30] Govindaraju N, Knott D, Jain N, 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

[31] Sud A, Govindaraju N, Gayle R, Kabul I, Manocha D. Fast proximity computation among deformable models using discrete voronoi diagrams//Proceedings of ACM SIGGRAPH. Boston, Massachusetts, USA, 2006: 1144-1153

[32] Wald I, Mark W R, Günther J , Boulos S, Ize T, Hunt W, Parker S G, Shirley P. State of the art in ray tracing animated scenes//Schmalstieg D, Bittner J eds. STAR Proceedings of Eurographics 2007, The Eurographics Association. Prague, Czech Rep, 2007: 89-116

[33] Zhou K, Hou Q, Wang R, Guo B. Real-time KD-tree construction on graphics hardware//Proceedings of the ACM SIGGRAPH Asia 2008. ACM, New York, NY, USA, 2008: 1-11

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

[35] Tang M, Manocha D, Tong R. Multi-core collision detection between deformable models//Proceedings of the SIAM/ACM Joint Conference on Geometric and Physical Modeling. Department of Computer Science. San Francisco, USA, 2009: 101-116

[36] Kim D, Heo J, Yoon S. HPCCD: Hybrid parallel continuous collision detection//Proceedings of the Korea Advanced Institute of Science and Technology. South Korea, 2009: 235-240

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

附录 A.　SIMD-DOP 实现的 C++伪代码.

```
class SIMD-DOP{
public:
  struct {
    __m128 _max[2];
          //packing all the 16 float data into 4 __m128
    __m128 _min[2];
  } v;

public:
  static void getDistances(vec3f&p, float d[]) {
    d[0]=p[0]+p[1];
    d[1]=p[0]+p[2];
    d[2]=p[1]+p[2];
    d[3]=p[0]-p[1];
    d[4]=p[0]-p[2];
  }

  SIMD-DOP() {empty();}

  SIMD-DOP(vec3f &iv) {
    float d[5];
    getDistances(iv, d);

    v._min[0]=v._max[0]
          =_mm_setr_ps(iv[0], iv[1], iv[2], d[0]);
    v._min[1]=v._max[1]
          =_mm_set_ps(d[1], d[2], d[3], d[4]);
  }

  SIMD-DOP(vec3f &a, vec3f &b) {
    float ad[5], bd[5];
    getDistances(a, ad);
    getDistances(b, bd);

    __m128 s=_mm_setr_ps(a[0], a[1], a[2], ad[0]);
    __m128 t=_mm_setr_ps(b[0], b[1], b[2], bd[0]);
    v._min[0]=_mm_min_ps(s, t);
    v._max[0]=_mm_max_ps(s, t);

    s=_mm_set_ps(ad[1], ad[2], ad[3], ad[4]);
    t=_mm_set_ps(bd[1], bd[2], bd[3], bd[4]);
    v._min[1]=_mm_min_ps(s, t);
    v._max[1]=_mm_max_ps(s, t);
  }

  bool overlaps(SIMD-DOP&b) {
    if (_mm_movemask_ps(_mm_cmpgt_ps(v._min[0],
        b.v._max[0]))) return false;
    if (_mm_movemask_ps(_mm_cmplt_ps(v._max[0],
        b.v._min[0]))) return false;
    if (_mm_movemask_ps(_mm_cmpgt_ps(v._min[1],
        b.v._max[1]))) return false;
    if (_mm_movemask_ps(_mm_cmplt_ps(v._max[1],
        b.v._min[1]))) return false;
    return true;
  }

  SIMD-DOP &operator+=( vec3f &p) {
    float d[9];
    getDistances(p, d);

    __m128 t=_mm_setr_ps(p[0], p[1], p[2], d[0]);
    v._min[0]=_mm_min_ps(v._min[0], t);
    v._max[0]=_mm_max_ps(v._max[0], t);

    t=_mm_set_ps(d[1], d[2], d[3], d[4]);
    v._min[1]=_mm_min_ps(v._min[1], t);
    v._max[1]=_mm_max_ps(v._max[1], t);

    return *this;
  }

  SIMD-DOP &operator+=( SIMD-DOP &b) {
    v._min[0]=_mm_min_ps(v._min[0], b.v._min[0]);
    v._min[1]=_mm_min_ps(v._min[1], b.v._min[1]);
    v._max[0]=_mm_max_ps(v._max[0], b.v._max[0]);
    v._max[1]=_mm_max_ps(v._max[1], b.v._max[1]);

    return *this;
  }

  void empty() {
    v._max[0]=_mm_set1_ps(-FLT_MAX);
    v._max[1]=_mm_set1_ps(-FLT_MAX);
    v._min[0]=_mm_set1_ps(FLT_MAX);
    v._min[1]=_mm_set1_ps(FLT_MAX);
  }
};
```

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

MANOCHA Dinesh, born in 1963, professor, Ph. D.. He has also served as a program committee member for more than 50 leading conferences in these areas and also served in the editorial board of many journals.

TONG Ruo-Feng, born 1969, Ph. D., professor. As a participant, he was awarded second scientific and technological progress prize by the Ministry of Education. He has published more than 40 professional and academic papers, most indexed by SCI/EI.