中图法分类号: TP391.41 **文献标志码**: A **文章编号**: 1006-8961(2011)03-0454-08
论文索引信息:纪传舜, 刘卉. 基于质点的可变形体自碰撞检测[J]. 中国图象图形学报,2011,16(3):454-461

基于质点的可变形体自碰撞检测

纪传舜, 刘卉
(复旦大学计算机科学技术学院,上海 200433)

摘 要: 自碰撞检测是可变形体模拟过程中最耗时的环节,提出一种使用图形硬件的快速算法。算法以质点而非三角形作为自碰撞检测的基本单元,用球体包围以质点为中心的局部区域,再用 AABB 包围该球体的运动轨迹并将数据组织成纹理送入 GPU,通过两遍离屏渲染计算出碰撞对集合及每个碰撞对的碰撞发生时间,算法复杂度为 $O(n)$。实验结果表明,使用该算法在大规模布料模拟中检测自碰撞,效率较高。

关键词: 自碰撞检测; GPU; 层次包围盒; 布料模拟

Particle-based self collision detection for deformable objects using graphics hardware

Ji Chuanshun, Liu Hui
(*School of Computer Science, Fudan University, Shanghai* 200433 *China*)

Abstract: Self collision detection is the most time consuming process in deformable object simulation. In this paper, a novel algorithm is presented for performing self collision using graphics hardware. Each particle, rather than the traditional triangle, is taken as a primitive. The particle-centered local regions are bounded with a sphere, where the AABB is then enclosed. After initial culling, the AABBs of potentially colliding particles are organized in textures and sent to GPU. A linear time two-pass rendering procedure is proposed to compute the set of colliding pairs and the colliding time of each pair. Experimental results demonstrate this algorithm handles self collision at interactive rates in the context of cloth simulation.

Keywords: self collision detection; GPU; bounding volume hierarchy; cloth simulation

0 引 言

可变形体的自碰撞检测是计算机图形学领域的重要研究课题之一,在物理模拟、3 维游戏、医学手术以及虚拟制造等诸多领域都有广泛应用。在可变形体的动态模拟中,由于存在着大量邻近图元之间的潜在碰撞,导致碰撞检测,尤其是自碰撞检测占据了 50%~90% 的整体模拟时间。因此,自碰撞检测已成为模拟过程的瓶颈所在。

以往的可变形体碰撞检测都发生在 CPU 上,随着图形硬件的飞速发展,研究者开始考虑将碰撞检测转移到 GPU 上进行。其中具有代表性的研究成果是 Govindaraju 等人在 SIGGRAPH 2005 上提出的 CDCD 算法[1]。该算法把组成可变形体的大量图元区分为临边/非临边图元两种类型:临边图元之间的碰撞检测在线性时间内完成;针对非临边图元之间的碰撞检测,首先利用色数分解算法把图元划分为 k 个互不相交的子集,然后应用经典的 GPU 碰撞检测算法 CULLIDE[2] 进行碰撞剔除,得到一个较小的潜在碰撞集合,从而实现较高效率的自碰撞检测。因为 CULLIDE 算法的复杂度为 $O(n)$,所以非临边

收稿日期:2009-03-16;**修回日期**:2009-10-20

第一作者简介:纪传舜(1983—),男,英特尔亚太区研发有限公司工程师。2009 年于复旦大学获计算机软件与理论专业硕士学位,研究方向为 GPU 图形处理。E-mail:shawnjee@gmail.com。

图元碰撞检测的算法复杂度为 $O(kn)$,而 k 值可以控制在一个较小的范围内(1~20),因此,CDCD 算法可以看做一个渐进线性算法。但是,CULLIDE 算法的准确性由图像空间分辨率决定,由于采样误差,可能会错过一些微小图元之间的碰撞。此外,CDCD 算法本身也非常复杂。

目前,国内基于图形硬件的加速算法也是研究热点之一。王季等人[3]提出的碰撞检测算法以带深度纹理的包围盒替代物体的几何模型,利用图形硬件在纹理映射时进行深度比较,从而实现快速碰撞检测。范昭炜等人[4]提出的基于图像的碰撞检测算法则是对物体表面进行自动凸分解,将凸分解结果组织成层次结构,利用图形硬件的加速功能进行碰撞检测。但是,上述算法都是针对物体之间的碰撞检测,不能进行快速的自碰撞检测。

工作以基于弹簧-质点模型[5-6]的布料模拟作为研究对象,提出一种基于质点的快速自碰撞检测算法。算法首先利用层次结构进行初步的局部区域碰撞剔除,得到潜在碰撞质点集合;然后在 GPU 上通过两遍离屏渲染,获得确切碰撞对集合。本文算法完全不同于 CDCD 算法基于三角形面片的遮挡查询思想,而是利用质点的离散性特征,用质点作为计算的基本单元,以点代面;同时结合了包围盒技术和基于 GPU 方法的优点,把空间数据组织成纹理送入 GPU 进行计算,达到了加速的目的。由于 GPU 计算的并行性,算法复杂度可以达到 $O(n)$。此外,与 CDCD 算法相比,本文提出的算法未利用其他现有算法,算法设计得比较简洁,具有较高的实用性。

1　算法的基本思想

假设在模拟过程中,布料的拓扑结构不发生改变,模拟以离散时间步的方式进行,通过线性插值得到质点的连续位移。

在弹簧-质点模型中,对于每个质点,可以根据质点的当前速度 v 和模拟时间步长 t,预计算出质点在下一个时间步的空间位置

$$\boldsymbol{P}_{t+1} = \boldsymbol{P}_t + vt$$

则质点在当前时间步的运动轨迹——路径矢量

$$\boldsymbol{s} = \boldsymbol{P}_{t+1} - \boldsymbol{P}_t$$

在布料的拓扑结构中,假设连接两个质点的弹簧初始长度为 L,我们可以近似认为:每个质点的速度代表以该质点为中心、半径小于 $L/2$ 的局部区域的速度。在图形空间中,以质点的空间坐标为中心,用半径为 $R(R \leqslant L/2)$ 的球包围这个区域(图 1),则质点的路径矢量也是包围球的路径矢量。我们再用 AABB[7]包围每个包围球在一个模拟时间步中的路径轨迹(图 2)。可以推断,如果两个 AABB 不相交,则它们对应的包围球所包围的局部区域也不会发生碰撞,这就是我们进行碰撞剔除的基本思想。

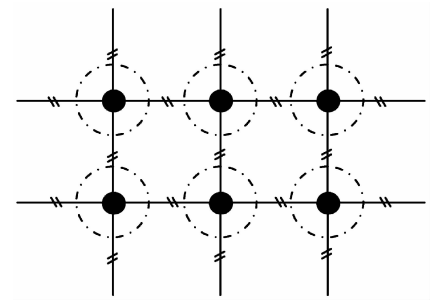


图 1　包围球包围质点的 2D 视图

Fig. 1　Particles and bounding balls

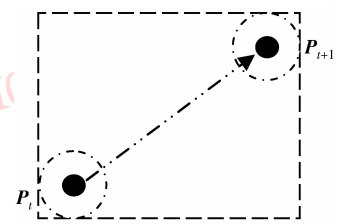


图 2　AABB 包围一个时间步中包围球的运动轨迹

Fig. 2　AABB of the trajectory of a bounding ball

之所以选择 AABB 而不是包裹得更紧密的 OBB 作为包围盒,主要是因为 OBB 的数学描述和计算较 AABB 复杂,不容易组织成用于 GPU 计算的纹理;而 AABB 只需两个顶点就能明确定义,可以非常容易地组织成纹理送入 GPU。

我们把要进行相交测试的 AABBs 作为图元,在 GPU[8-9]上实现碰撞剔除算法。假设经过预处理后得到的潜在碰撞集合由 n 个 AABB 组成,编号为 1,2,…,n,分别与 n 个片段一一对应。为了算出所有潜在碰撞对,总共需要进行 $n(n-1)/2$ 次相交测试,如图 3 所示。图中,横坐标为图元编号,纵坐标为各图元所对应的片段编号,即在片段 i 上需要进行图元 i 与其后$(n-i)$个图元之间的相交测试。可以看出,计算量集中在图中的上三角区域:片段 1 的计算量最大,而在片段 n 上却不需要进行任何计算。由于计算量不均匀,算法受计算量最大的片段的渲

染时间限制。针对这个问题,我们设计了平摊算法来平衡每个片段上的计算量。

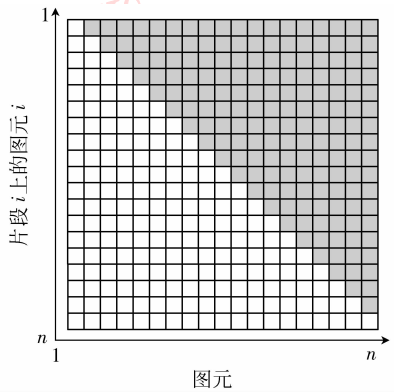


图 3 每个片段上的计算量

Fig. 3 Computation on each fragment

基于 GPU 的流计算特性,对应 n 个图元,平摊算法把所有相交测试平均到 n 个片段上并行计算。对于要进行相交测试的任意图元对(i,j),平摊算法定义为

1)如果$|i-j|\leqslant(n-1)/2$,在片段 $\min(i,j)$ 上进行相交测试;

2)否则,在片段 $\max(i,j)$ 上进行相交测试。

也就是说,在片段 i 上进行图元 i 与其后$(n-1)/2$ 个图元的相交测试。

经过平摊算法处理后,每个片段上的计算量如图 4 所示,红色部分的计算量被转移到黄色部分。也就是说,每个片段上的计算量都变为$(n-1)/2$,即,分摊到每个流处理器的工作量得到了均衡,并行处理的效率最高。

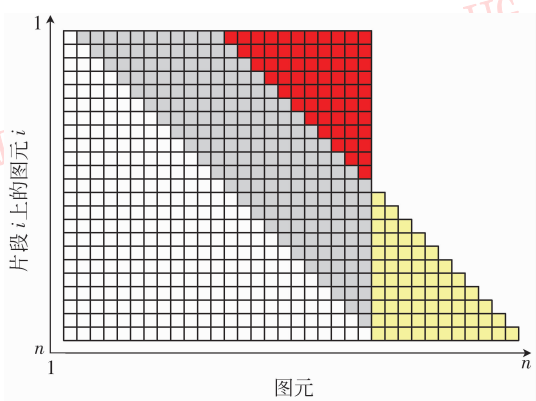


图 4 平摊后每个片段的计算量

Fig. 4 Computation on each fragment after using our balancing algorithm

上述过程即为第 1 遍 GPU 渲染,经过此次渲染,在每个片段 i 上可以得到图元 i 与其后$(n-1)/2$ 个图元发生碰撞的次数,即碰撞对个数。计算结果被封装为像素格式写入帧缓冲区。由于每个片段上的碰撞对个数不同,而且每个片段的渲染结果只能写入 RGBA 4 个颜色通道,因此无法在第 1 遍渲染中直接保存所有碰撞对的编号。换句话说,第 1 遍渲染只能返回碰撞次数,无法保存发生碰撞的图元编号。因此,我们设计了第 2 遍 GPU 渲染,根据碰撞次数获得确切的碰撞对集合。

在第 2 遍渲染之前,先统计第 1 遍渲染所获得的碰撞对总数 m,将碰撞对分别编号为 1, 2, …, m,与片段编号一一对应。在求碰撞对总数 m 的加和过程中,同时建立图元、碰撞对和片段之间的对应关系。举例来说,对于碰撞对 $k(i,j)$,碰撞对编号为 k,对应片段 k,i, j 为图元编号。第 2 遍渲染算法为

1)首先,根据第 1 遍渲染结果,计算出碰撞对(i, j)中的 j 值,即确定哪两个图元相交。

2)然后,计算图元 i, j 所对应的包围球在 3D 空间中第 1 次发生碰撞的时间,并写入帧缓冲区。

经过第 2 遍渲染,可以得到所有碰撞对及碰撞发生时间,从而完成碰撞检测。

2 算法的步骤

提出的快速自碰撞检测算法分为 3 个阶段:预处理,第 1 遍渲染,第 2 遍渲染。预处理阶段进行粗略的碰撞剔除,将潜在碰撞集合组织成纹理数据;第 1 遍渲染阶段计算每个图元与其后$(n-1)/2$ 个图元发生碰撞的次数;第 2 遍渲染阶段得到所有碰撞对和碰撞发生的确切时间。

2.1 预处理阶段

在质点数量较大的情况下,我们采用层次四叉树结构对非碰撞区域进行粗略剔除。层次四叉树的构造方式类似于传统的八叉树[10-11]:用 AABB 包围各个质点的运动轨迹,作为四叉树的叶结点;然后根据布料的拓扑结构自底向上构造父结点的 AABB(图 5),根结点为整块布料的 AABB。相交测试沿着四叉树自顶向下进行,可以剔除大量的非碰撞区域,得到一个潜在碰撞质点集合。在质点数目较少的情况下,也可以把所有质点作为潜在碰撞质点集合。

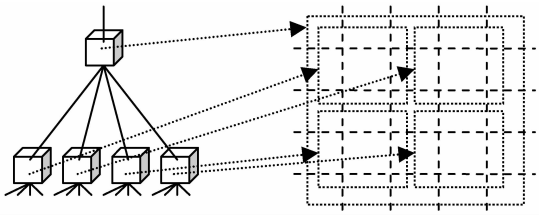


图 5　四叉树结点与布料拓扑结构的对应关系

Fig. 5　Correspondence between quadtree nodes and cloth stucture

在预处理阶段需要为每个潜在碰撞质点构造包围球,计算路径矢量及包围路径矢量的 AABB,并组织成纹理数据。

假设潜在碰撞质点集合包含 n 个质点(1,2,…,n)。对每个质点,构造半径为 R($R \leqslant L/2$,L 为弹簧长度)的包围球,并计算包围此包围球在下一时间步的运动轨迹的 AABB。为了利用 GPU 的计算能力,需要把数据组织成纹理形式送入 GPU。AABB 可以通过 3D 空间中的两个点来标记,即最大点($\boldsymbol{X}_{max}$,$\boldsymbol{Y}_{max}$,$\boldsymbol{Z}_{max}$)和最小点($\boldsymbol{X}_{min}$,$\boldsymbol{Y}_{min}$,$\boldsymbol{Z}_{min}$)。由于纹理中每个像素具有 RGBA 4 个颜色通道,因此需要两个大小为 n 的纹理来存储所有质点的 AABBs 信息。我们定义输入纹理 tex_ABmax 保存所有 AABBs 的最大点,$\boldsymbol{X}$,$\boldsymbol{Y}$,$\boldsymbol{Z}$ 值分别存入 RGB 颜色通道,颜色通道 A 保存对应质点的编号。定义纹理 tex_ABmin 保存所有 AABBs 的最小点,存储顺序与 tex_ABmax 相同。这样,只要读取这两个纹理的相同位置就可以获得对应质点的 AABB 信息。我们把组织成纹理形式的质点数据称为图元。

2.2　第 1 遍渲染

为了保证计算精度,我们采用了 OpenGL 扩展 EXT_framebuffer_object。该扩展允许我们把一个离屏缓冲区作为渲染运算的目标,而且缓冲区中每个片段的 RGBA 4 个颜色通道都是 32 位浮点数。这样一来,我们可以在 GPU 上得到一个全精度浮点数,同时也消除了限定数值范围的问题。

为了使纹理数据和渲染片段一一对应,我们定义缓冲区与纹理具有同样大小。第 1 遍渲染算法首先从纹理 tex_ABmax 和 tex_ABmin 的当前坐标读取 AABB 数据和当前图元的编号,然后依次读取当前图元之后的($n-1$)/2 个图元数据并计算是否与当前图元相交,相交则计数器加 1,计数器的最终值写入缓冲区中对应片段的 R 通道。算法伪代码如下:

算法 1

```
float4 FirstPassRender (
  in float2 coords : TEXCOORD0,
  /* 当前片段坐标 */
  uniform samplerRECT tex_ABmax,
  /* AABB 数据 */
  uniform samplerRECT tex_ABmin,
  /* AABB 数据 */
  uniform float n /* 图元总数 */) : COLOR
{
  读取当前图元 T 所对应的 AABB 数据;
  定义计数器 = 0;
  /* 对 T 和 T 后的(n-1)/2 个图元做相交测试 */
  for (i = 1; i <= (n-1)/2; i++) {
  读取图元(T + i)% n 所对应的 AABB 数据;
  if (T 的 AABB 与(T + i)% n 的 AABB 相交)
       计数器加 1;
  }
  返回 float4(计数器值,0,0,0);
}
```

由于每个图元对应一个 GPU 线程(thread),因此该算法在 GPU 的每个线程上并行执行,时间复杂度为 $O(n)$。

第 1 遍渲染算法通过平衡每个片段上的计算量,获得了片段所对应的图元与其后($n-1$)/2 个图元发生相交的数目。

2.3　第 2 遍渲染

在进行第 2 遍渲染之前,先统计第 1 遍渲染缓冲区中所有片段的 R 通道值,得到潜在碰撞对总数 m。在求和过程中,只能确定当前图元与其后($n-1$)/2 个图元发生潜在碰撞的次数,而无法确定是与哪几个图元发生潜在碰撞。为了解决这个问题,我们构造了一个图元与碰撞对的关系纹理 tex_Related。

假设潜在碰撞对编号为 1,2,…,m,则关系纹理 tex_Related 的大小和第 2 遍渲染缓冲区的大小也定义为 m。关系纹理 tex_Related 中像素的 RGBA 4 个颜色分量分别定义如下:

R—保存当前碰撞对编号;

G—保存当前碰撞对中的图元编号 i;

B—保存当前碰撞对是图元 i 与其后($n-1$)/2 个图元发生的第几次碰撞;

A—标志当前碰撞对是否是有效碰撞对。

第 2 遍渲染算法首先从关系纹理 tex_Related 的当前坐标读取像素值,得到颜色通道 B 的值。然

后再次进行当前图元 i 与其后 $(n-1)/2$ 个图元之间的相交测试,得到第 $|B|$ 次碰撞的图元编号 j,如图 6 所示。

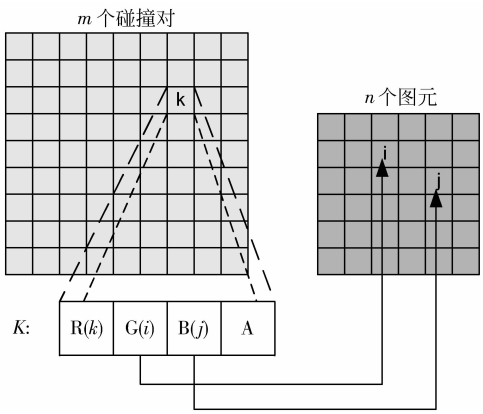


图 6 第 2 遍渲染过程中纹理之间的关系

Fig. 6 Correspondence between textures in the 2nd rendering process

在确定了潜在碰撞对 (i,j) 之后,读取它们的路径矢量并计算发生碰撞的确切时间,写入目标缓冲区。算法伪代码如下:

算法 2

```
float4 SecondPassRender (
  In float2 coords : TEXCOORD0,
  uniform samplerRECT tex_ABmax,
  uniform samplerRECT tex_ABmin,
  uniform samplerRECT tex_Related,
  uniform float m,
uniform samplerRECT tex_Vector,/*包围球路径矢量*/
uniform samplerRECT tex_Position, /*质点当前坐标*/
  uniform float R /*包围球半径*/ ): COLOR
{
  读取 tex_Related 当前位置,得到当前图元编号 i,及 i 的第|B|次碰撞;
  读取 tex_ABmax, tex_ABmin 中图元 i 的信息;
  依次读取图元 i 后(n-1)/2 个图元的数据,并计算到第|B|次相交,得到对应的图元 j;
  读取 i,j 的路径矢量;
  if (i,j 的路径矢量相交) {
    计算第一次相交时间 t;
    返回 float4(t, i, j, 1);
  }
  else
    返回 float4(0, 0, 0, 0);
}
```

第 2 遍渲染算法把计算结果封装为像素返回。其中,G、B 通道保存当前碰撞对的质点编号 i,j;R 通道保存当前碰撞对发生碰撞的时间;A 通道标志当前碰撞对是否确切发生碰撞。在第 2 遍渲染过程中,需要重复第 1 遍渲染的求交工作,以确定哪两个图元发生碰撞。

2.4 算法优化

算法通过预计算质点下一时间步的空间位置来判断是否会发生碰撞,发生碰撞的质点进行碰撞响应,而大量的非碰撞质点则运动到预计算出的空间位置。因此,在模拟过程中,可以把模拟算法和预计算过程整合在一起。这样一来,对于非碰撞质点,只需要进行一遍计算;对于碰撞质点,则碰撞响应后进入下一时间步的模拟计算。实验表明,对于预计算过程的优化,可以使该过程的计算量大大降低。

对于质点数较多的情况,本文算法在预处理阶段构造了层次四叉树进行局部区域的碰撞剔除。层次结构在提高剔除效率的同时,其自身的更新也成为模拟系统的负担之一。因此,是否使用层次结构的关键在于:如果更新效率过低,更新代价超过剔除所带来的益处,就没有必要使用层次结构。我们所采用的层次四叉树在初始化过程中构造完成,具有固定的树结构;在随后的模拟进程中,只有结点的 AABB 数据会发生变化,自底向上[12]即可更新四叉树,更新效率较高。在大规模质点模拟的场景里,采用层次结构后碰撞剔除效果明显。

由于我们把碰撞检测繁重的计算转移到 GPU 上进行,因此算法效率也取决于 GPU 的各项性能。我们采用了 OpenGL 扩展 EXT_framebuffer_object 来保证算法以 32 位全精度浮点数进行计算,这也是基于 GPU 的新特性。因此,一个支持更多新技术和更高性能的图形卡非常有助于算法的改进和优化。

3 实验与验证

我们在一台 Pentium IV 3.2 G CPU,内存 4 GB,显卡 NVIDIA Geforce8800,显存 768 MB 的机器上实现了本文算法。算法的 GPU 部分用 Cg 语言编写,用 OpenGL 的扩展 EXT_framebuffer_object 进行离屏渲染计算。布料模拟采用显式积分方法(图 7,8)。

3.1 质点包围球的半径

设包围球的半径为 R,弹簧长度为 L,我们对 R 的不同取值分别进行实验,模拟效果如表 1 所示。

图 7　模拟窗帘产生的褶皱效果,质点数为 24K

Fig. 7　Curtain simulation with 24K particles

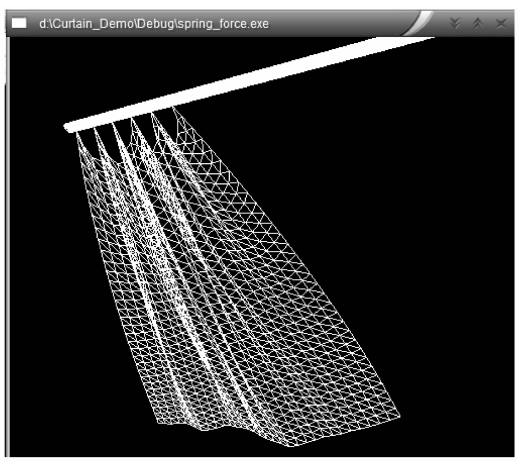


图 8　窗帘模拟的三角形视图

Fig. 8　Triangle mesh of curtain simulation

表 1　包围球半径对碰撞检测的影响

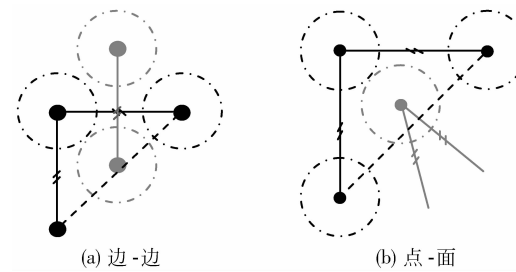
Tab. 1　The effect of the radius of the bounding ball

R 的取值	模拟效果
$R \geqslant 0.5L$	严重失真
$0.4L \leqslant R < 0.5L$	局部失真
$0.3L \leqslant R < 0.4L$	良好
$R < 0.3L$	少量穿透现象

实验表明,包围球半径取值在$[0.3L, 0.4L)$之间时,模拟和碰撞检测效果最好。

在传统的基于三角形图元的碰撞检测中,两个三角形之间的碰撞最终可归结为 9 次边-边相交测试和 6 次点-面相交测试。提出的基于质点的方法,对于边-边和点-面碰撞可以实现精确检测,不会遗漏任何碰撞。

边-边碰撞如图 9(a)所示,点-面碰撞如图 9(b)所示。可以计算出当质点包围球半径 $R \geqslant \frac{\sqrt{2}}{4}L$($R \approx 0.353L$)时,即使是正交碰撞,算法也能够检测出质点包围球是相交的。也就是说,在质点包围球半径 $R \approx 0.353L$ 时,可以检测到所有碰撞,这个结论在实验中得到了证实。



(a) 边-边　　(b) 点-面

图 9　边-边碰撞和点-面碰撞

Fig. 9　Edge-edge collision and vetex-face collision

由于包围球并不是一个严格的定义,在极少数情况下,可能会发生误检。如图 10 所示,点-面并未接触,但包围球却已经相交。不过,由于我们检测的是包围球轨迹的碰撞,图 10 所示的状态可能会发生在一个时间步中的某个瞬间。如果这种情况发生,根据质点运动轨迹的连贯性,说明在该瞬间之后当前时间步剩下的时间内,灰色质点所在的局部区域和黑色质点所在的局部区域极有可能发生碰撞。即使当前帧不发生碰撞,在下一帧的运动中也极有可能发生碰撞。因此,基于快速检测的目的,这种情况仍然可以当作碰撞发生来处理。

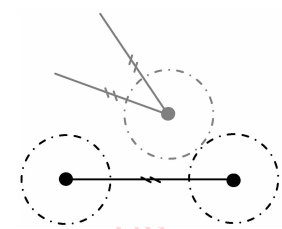


图 10　误检的情况

Fig. 10　False detection

在系统完成初始化后,质点间相同类型弹簧的长度相等。然而在模拟过程中,由于质点的受力不同,弹簧形变将会使质点的分布不再均匀,可能出现如图 11(a)所示的钝角三角形(钝角接近 180°)。为了避免该类三角形的出现,我们在实现中采用较短对角线法进行三角剖分。显然,添加较短对角线

形成的三角形都是锐角或直角三角形,优于添加较长对角线形成的三角剖分,如图 11(b)所示。根据织物易于弯曲而不易伸缩的物理特性,我们设置了较大的弹簧拉伸系数,使得弹簧的伸缩程度有限,不会出现因弹簧过度拉伸而导致碰撞漏检的现象。

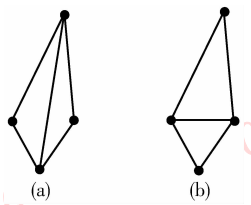


图 11 网格的三角剖分

Fig. 11 Mesh triangulation

实验中可以观察到,在包围球半径取值恰当的情况下,检测效果良好。

3.2 与 CDCD 算法的比较

由于大多数基于 GPU 的碰撞检测算法都存在图像分辨率的限制,且无法检测自碰撞,因此我们选择与 CDCD 算法进行比较来验证本文算法的有效性。CDCD 算法通过对三角形进行遮挡查询来判断是否发生碰撞,本文算法则通过基于质点的包围球路径轨迹求交来判断是否发生碰撞。在具有相同质点数的布料模拟中,本文算法和 CDCD 算法每一帧的平均自碰撞检测时间如图 12 所示。

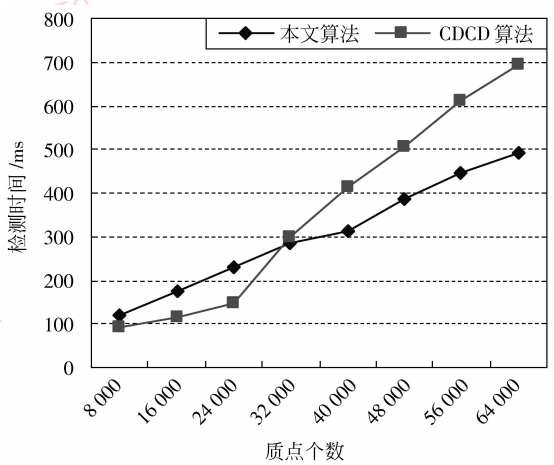


图 12 两种算法的检测时间比较

Fig. 12 Comparison of detection time between our algorithm and CDCD algorithm

从图 12 可以看出,本文算法的平均检测时间随着质点数的增加线性增加,算法复杂度为 $O(n)$,略优于 CDCD 算法的 $O(kn)$。

此外,CDCD 算法受屏幕分辨率的限制,而本文算法因为采用了离屏渲染,对屏幕分辨率没有要求,所以在屏幕分辨率方面没有进行比较。

3.3 影响检测结果的因素

算法的目标是实现快速的碰撞检测,通过对实验结果的分析,以下 3 方面因素影响着碰撞检测的结果:1)包围球半径的选取。当包围球半径过大或过小时,系统会产生明显失真的现象;2)模拟时间步长。当步长过大时,不仅模拟失真,而且弹簧可能被过度拉伸,从而出现漏检;3)预处理阶段所用时长。当层次四叉树遍历过深时,虽然可以显著减少送入 GPU 计算的质点数,但会造成预处理时间过长,因此需要根据系统情况选取一个合理的阈值。

4 结 论

提出的快速自碰撞检测算法不同于以往以三角形为计算单元的思想,考虑以质点为单位构造包围球和 AABB,计算模式简单,计算量小;同时又结合了层次数据结构和 GPU 并行计算的优点,实现了在线性时间内完成自碰撞检测,具有较高的实时性。本文算法可以扩展应用到所有基于弹簧-质点模型和粒子模型的可变形体的自碰撞检测当中,如 3D 动画和游戏中的服装模拟、虚拟医学手术等。

算法的不足之处在于第 2 遍渲染中部分重复了第 1 遍渲染中的相交计算工作。随着图形硬件的发展,如果 GPU 能够提供动态内存申请功能,我们就可以在第 1 遍渲染过程中直接保存潜在碰撞对集合。另一不足之处是计算过程中需要两次回读。在目前所有的 GPU 上,回读仍然比较耗时,而受 GPU 计算模式的限制,算法的回读问题不可避免。在 CDCD 算法中,遮挡查询结果同样也需要回读才能获得。我们相信,随着 GPU 技术的发展,本文算法仍有改进空间,算法效率也会得到相应提高。

参考文献(References)

[1] Govindaraju N K, KNnott D, Jain N. Interactive collision detection between deformable models using chromatic decomposition [J]. ACM Transactions on Graphics, 2005, 24(3): 991-999.

[2] Govindaraju N K, Redon S, Lin M C, et al. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware [C]//Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics

Hardware. San Diego, California: Eurographics Association, 2003: 25-32.

[3] Wang Ji, Zhai Zhengjun, Cai Xiaobin. Real time collision detection using depth texture [J]. Journal of Computer-Aided Design & Computer Graphics, 2007, 19(1): 59-65. [王季,崔正军,蔡小斌. 基于深度纹理的实时碰撞检测算法[J]. 计算机辅助设计与图形学学报,2007, 19(1): 59-65.]

[4] Fan Zhaowei, Wan Huagen, Gao Shuming. A fast collision detection algorithm in image space [J]. Journal of Computer-Aided Design & Computer Graphics, 2002, 14(9): 805-811. [范昭炜,万华根,高曙明. 基于图像的快速碰撞检测算法[J]. 计算机辅助设计与图形学学报, 2002, 14(9): 805-811.]

[5] Provot X. Deformation constraints in a mass-spring model to describe rigid cloth behavior [C]//Proceedings of Graphics Interface. Quebec: Canadian Information Processing Society, 1995: 147-154.

[6] Liu Hui, Chen Chun, Shi Baile. Simulation of 3D garment based on improved spring-mass model [J]. Journal of Software, 2003, 14(3): 621-627. [刘卉,陈纯,施伯乐. 基于改进的弹簧-质点模型的三维服装模拟[J]. 软件学报,2003, 14(3): 621-627.]

[7] Baraff D, Witkin A, Kass M. Untangling cloth [J]. ACM Transactions on Graphics, 2003, 22(3): 862-870.

[8] Heidelberger B, Teschner M, Gross M. Real-time volumetric intersections of deforming objects [C]//Proceedings of Vision, Modeling and Visualization. Munich: O C S L Press, 2003: 461-468.

[9] Knott D, Pai D K. CInDeR: Collision and interference detection in real-time using graphics hardware [C]//Proceedings of Graphics Interface. Halifax, Nova Scotia: Canadian Human-Computer Communications Society, 2003: 73-80.

[10] Jackins C L, Tanimoto S L. Octree and their use in representing three dimensional objects [J]. Computers & Graphics, 1980, 14(3): 249-270.

[11] Liu Xiaoping, Weng Xiaoyi, Chen Hao, et al. An improved algorithm for octree-based exact collision detection [J]. Journal of Computer-Aided Design & Computer Graphics, 2005, 17(12): 2631-2635. [刘晓平,翁晓毅,陈浩等. 运用改进的八叉树算法实现精确碰撞检测[J]. 计算机辅助设计与图形学学报,2005, 17(12): 2631-2635.]

[12] Larsson T, Akenine-Möller T. Collision detection for continuously deforming bodies [R]. Manchester, UK: Eurographics, short presentations, 2001: 325-33.