基于流的实时碰撞检测算法*

范昭炜+, 万华根, 高曙明

(浙江大学 CAD&CG 国家重点实验室,浙江 杭州 310027)

Streaming Real Time Collision Detection Using Programmable Graphics Hardware

FAN Zhao-Wei+, WAN Hua-Gen, GAO Shu-Ming

(State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310027, China)

+ Corresponding author: Phn: +86-571-87951045, Fax: +86-571-87951780, E-mail: zwfan@ntu.edu.sg, http://www.zju.edu.cn

Received 2003-10-13; Accepted 2004-05-08

Fan ZW, Wan HG, Gao SM. Streaming real time collision detection using programmable graphics hardware. *Journal of Software*, 2004,15(10):1505~1514.

http://www.jos.org.cn/1000-9825/15/1505.htm

Abstract: Real time collision detection is required by almost all computer graphics applications. However, the problem of real time collision detection is yet to be solved between complex objects. With the recent advent of high performance graphics processing units (GPUs), a dramatic shift is being seen in the traditional idea that general-purpose computation can only be performed by CPUs. This paper explores to solve the problem of real time collision detection between complex objects using programmable GPUs. The algorithm maps the computation of collision detection between two arbitrary objects onto programmable GPUs to match their parallel architectures, and produces on the fly the collision detection results via real time rendering. To do so, the problem is first converted into the problem of finding intersections between a collection of line segments and a set of triangles to realize the migration of collision detection algorithms to programmable GPUs. Based on reasonable analyses of the algorithm complexity, two optimized techniques is presented to improve its efficiency. Experimental results have shown that the optimized algorithm is advantageous over other current collision detection algorithms implemented in image space regarding efficiency, accuracy as well as practicability.

Key words: real time; collision detection; streaming computing; programmable graphics hardware; general-purpose computation

摘 要: 实时碰撞检测是计算机图形应用中不可或缺的问题之一,复杂物体间的实时碰撞检测至今仍未能得

* Supported by the National Natural Science Foundation of China under Grant No.60103003, 60021201 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA411310 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312100 (国家重点基础研究发展规划(973))

作者简介: 范昭炜(1975－),男,江西广丰人,博士,主要研究领域为计算机视觉,虚拟设计与装配,并行与分布式计算;万华根(1968－),男,博士,副研究员,主要研究领域为计算机动画,虚拟现实,科学计算可视化;高曙明(1964－),男,博士,教授,博士生导师,主要研究领域为先进产品建模技术,虚拟设计与装配,基于 Internet 的协同设计.

以很好的解决.高性能可编程图形硬件的出现,正在改变着通用计算仅能由 CPU 完成的传统观念.探索性地采用了可编程图形硬件来解决复杂物体间的实时碰撞检测问题.通过将两个任意物体间的碰撞检测计算映射到图形硬件以有效利用图形硬件的并行架构,由实时绘制过程快速产生碰撞检测结果.为此,算法首先将碰撞检测问题转化为一组线段集合与三角形的求交问题,以实现碰撞检测算法向可编程图形硬件的迁移.在对算法复杂度进行理性分析的基础上,给出了两种有效的优化技术以提升算法效率.实验结果表明,与现有的图像空间碰撞检测算法相比,该算法在效率、精确性和实用性方面具有明显优势.

关键词: 实时;碰撞检测;流计算;可编程图形硬件;通用计算

中图法分类号: TP391 **文献标识码**: A

实时碰撞检测在计算机图形学、虚拟现实、计算机游戏、动画、计算机辅助设计、机器人及虚拟制造等领域中均是经典而关键的问题.目前大多数碰撞检测算法均在物体空间实现,且采用了诸如层次包围盒、几何推理、形式化、空间分割、解析法和优化等技术[1~6].一些经典的软件包有 RAPID[7],I-COLLIDE[8],QuickCD[9],Q-COLLIDE[10]与 SWIFT++[11].但是,这些算法大都依赖于模型表示和场景复杂度.碰撞检测计算均由CPU完成.这给系统,尤其是处理大规模场景的系统,带来了沉重的计算负担.因此,研究人员开始探索其他途径来解决实时碰撞检测问题,而图形硬件性能的快速发展给人们提供了新的思路.

早在 1991 年,Shinya 和 Rossignac 等人就率先开创性地提出了图形硬件辅助的碰撞检测方法[12,13].随后,Baciu 等人进一步利用深度缓存和模板缓存的组合功能,提高了图像空间碰撞检测算法的效率,并使算法可应用于常规图形工作站甚至是普通个人台式电脑[14],但该算法仅能处理凸体.Hoff 和 Kim 等人将图像空间碰撞检测算法和物体空间碰撞检测算法结合起来,混合利用二者优点,增强了算法的功能,同时通过一定的负载平衡策略,在 CPU 与图形硬件单元(GPU)间进行合理调配[15,16].我们结合表面凸分解技术扩大了基于图像碰撞检测算法适应面,并采用绘制加速技术来提高基于图像碰撞检测算法的效率[17,18].Govindaraju 等人提出了一种面向多物体的图像空间碰撞检测算法 CULLIDE,他们将复杂物体分解为子物体,构建潜在碰撞集(PCS),并利用可见性计算减少 PCS 中物体的数目.由于在计算可见性关系时利用了图形硬件的遮挡查询功能,他们的算法具有较高的效率[19].

上述图像空间碰撞检测算法通常先将三维几何物体投影到二维图像平面(如帧缓存、深度缓存、模板缓存等)上,然后分析降维的二维空间,同时辅以深度测试,得到碰撞检测结果.它们试图尽量利用 GPU 的计算性能辅助计算,实现也相对简单,但由于图形硬件本身固有的离散性,因此不可避免地会产生一些误差,无法保证结果的准确性.

随着半导体技术的迅猛发展,GPU 在性能上已经超越 CPU,而且由于 GPU 内在的并行结构,这种性能差距将会变得越来越大.另一方面,图形硬件体系结构正在发生质的变革,由固化函数构成的图形流水线正被可编程的顶点绘制处理器(vertex shader/vertex program)和像素绘制处理器(pixel shader/fragment program)所取代[20].图形绘制流水线因此进化为一种通用的可编程流处理器,而不再只是简单地传输与绘制三角形.这种变革令 GPU 成为可用于进行密集浮点计算的高性能计算引擎之一,远远超出了它最初的设计目的.一些高性能的图形卡,如 ATI Radeon 系列和 NVIDIA GeForce FX 系列展示了它们面向高性能浮点硬件的可扩展编程接口.于是人们开始考虑如何把许多基础图形算法甚至其他数学问题映射到这种新的计算模式上.可以说,如何把 GPU 应用于通用计算目的已经成为活跃的研究领域之一.

在图形应用领域中,可编程 GPU 一般可用于过程纹理映射、实时绘制以及体可视化[21~24].最近,Carr 等人和 Purcell 将可编程 GPU 用于光线跟踪算法[25,26].他们的方法通过可编程 GPU 的片断绘制程序极大地提高了光线跟踪的效率.而非图形领域的应用同样也能利用可编程 GPU 获得好处.Bolz 等人展示了如何利用 GPU 作为数值仿真引擎[27].Harris 等人设计了一个利用可编程 GPU 处理动态现象的实时可视仿真器[28].

我们认为,可编程 GPU 的高性能和灵活性应该可以使其能够处理更为通用、传统计算意义下难以处理的计算问题,如复杂三维几何物体的实时碰撞检测.本文探索性地采用可编程 GPU 来解决实时碰撞检测问题,通过将两个任意形状物体的碰撞检测过程映射到可编程 GPU 上,由三维图形实时绘制过程计算出精确的碰撞检

测结果.在此基础上,我们提出两种有效优化措施以提高算法效率.

1 流计算模式

当前图形硬件正经历一个巨大变化,用户可编程流水线正取代由固化函数构成的流水线.如文献[26]中的图1,这种可编程流水线主要表现在两个阶段:一是顶点绘制阶段;二是片断绘制阶段.在这两个阶段中,执行的由用户自定义的程序分别称为顶点绘制程序和片断绘制程序.可以说,顶点绘制器是一种用户可编程的流处理器,它能改变送入到光栅转换器中顶点的属性[20].片断绘制阶段也是可编程的,片断绘制器能够在多个纹理坐标下对单个像素进行算术操作及获取纹理采样数据.尤其是最近像素缓存(pixel buffer)和浮点缓存(float buffer)技术[29,30]的出现,突破了片断绘制程序通常只能进行定点计算的局限,使得在片断绘制程序中处理浮点数成为可能.因此,GPU在硬件技术不断发展的过程中越来越像一个通用处理器.然而,GPU要胜任通用处理器的最大挑战在于引入可编程特性后不会使其性能受到很大影响.若非如此,可编程 GPU 就只是一种新的 CPU 而已,反而丧失了其自身的性能优势.为了保持可编程GPU的高性能,就要充分利用其并行计算的特点."流计算"概念的提出正是为了便于将相关应用映射到这一新的图形硬件体系上.

不同于传统的计算模式,流计算模式以一个数据序列作为输入数据,并采用同一段内核程序并行处理所有输入数据,最后将计算结果置于输出流输出.从该意义而言,可编程 GPU 用顶点绘制程序处理一个顶点数据流,用片断绘制程序处理像素流并输出到帧缓存.多数情况下顶点绘制程序可忽略,因此可编程GPU又可被看作数据流的片断处理器.这种新的流计算模式充分利用 GPU 并行计算的优势,有效使用高带宽,并且隐藏了内存重用的潜在危险.可编程GPU也正是通过这种方式才能有效利用VLSI的相关资源.

2 基于流的碰撞检测基本算法

利用流计算模式来进行碰撞检测,关键要将碰撞检测计算映射为流计算模式.为了有效地实现这一映射,针对两个基于网格表示的物体,我们采用检查物体的所有边与另一物体的所有三角形是否有交来确定它们是否发生碰撞.进一步,我们采用可编程 GPU 的实时绘制过程进行边与三角形的求交以具体实现映射.

基于流的碰撞检测基本算法由边纹理生成、边与三角形求交和求交结果获取三部分组成.各部分之间通过存储在纹理和帧缓存中的数据流联系在一起.设 A,B 为两个待检测的基于网格表示的物体,为叙述简便起见,下面仅讨论 A 的所有边与 B 的所有三角形之间的相交检测过程.

2.1 边纹理生成

边纹理生成过程主要依据物体 A 的边表生成边纹理.边纹理有两个,一个为起始点纹理,存放所有边的起始点坐标;另一个为方向长度纹理,用于保存边的方向和长度信息.

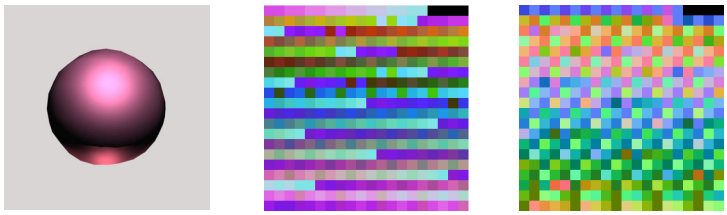
假定 A 的每条边采用参数表示为 $e(t)=orig+t\bullet dir$,其中 $orig$ 是边的起始点,dir 是边的方向单位矢量,边的长度值表示为 len.我们依据 A 的边表信息来生成起始点纹理和方向长度纹理,其具体存放方式如下:

- 起始点纹理:图像中每个像素的3个颜色通道存放一个起始点的位置向量 $orig$;
- 方向长度纹理:图像中每个像素的 3 个颜色通道用于存放一条边的方向单位向量 dir,而边的长度值 len 则由该像素的 $alpha$ 通道保存.

这两个边纹理大小相同,其每一纹理中相对应的像素表示 A 物体的一条边.纹理图像的大小由具体边数决定:

$$w=h=\lfloor\sqrt{n_e}\rfloor+1.$$

其中,n_e 为 A 的边数,w,h 分别为纹理的宽度和高度.NVIDIA的矩形纹理技术[30]可灵活地用于存储不同大小的边纹理,并保证非纯黑色像素和边一一对应.例如,图 1(a)为一个有 396 条边的球体,图 1(b)、图 1(c)分别为映射到[0,1]区间的边起始点纹理和边方向长度纹理,其分辨率均为20×20=(396+4).注意,其右上角4个黑色像素没有和任何边相对应.



(a) Sphere model (b) Origin texture (c) Direction-Length texture

Fig.1 A sphere and its edge textures

图 1 球体及其相应的边纹理

2.2 边与三角形求交

边与三角形求交过程主要通过自定义片断绘制程序处理 A 的所有边和 B 的所有三角形的求交计算.我们将这一过程处理为循环过程,且在循环的每一迭代中,对 A 的所有边和 B 的一个三角形进行求交检测.为此,需要先创建一个分辨率大小与 A 的边纹理一致的视窗口专用于求交计算,并设定相应的视域参数.同时,建立大小与视窗口一致的矩形.在每一次迭代中,将当前待处理的 B 的三角形的相关信息存储到该矩形的 4 个顶点的多纹理坐标中,绘制该矩形的光栅化过程将这些属性插值到视窗口的每个像素上.同时,通过纹理映射,可将保存在边纹理中的 A 的各条边的参数映射到各个像素上.这样就可以在每个像素上用自定义的片断绘制程序进行 A 的边和 B 的三角形的求交计算.

为加速计算过程,可禁用该视窗口的颜色缓存,并在视窗口内采用像素缓存技术[29]进行绘制,以把用于计算的绘制过程和用于模型显示的绘制过程区分开.各像素上的计算结果将输出到对应的深度缓存.值得指出的是,为便于深度测试比较,在求交计算之前将 A 所有边的长度值预先写入到深度缓存.有关求交过程的具体细节将在第 4 节阐述.

2.3 求交结果获取

求交结果获取阶段通过硬件深度测试和遮挡查询(NV_occlusion_query)[30]快速得到碰撞检测结果.由于 A 所有边的长度值在求交计算之前均已预先写入深度缓存,且片断绘制程序在各像素上将 A 的每条边与当前处理的 B 的三角形求交结果(即参数值 t)作为对应像素的深度值输出,因此,通过简单的深度比较,即可得知 A 是否存在边与当前处理的 B 的三角形相交.具体地说,若存在某像素的 $t(\geq 0)$ 值小于等于对应边长度(对应于能通过 GL_LEQUAL 深度测试),则两物体相交.否则继续迭代过程,直到处理完 B 的所有三角形.在此,我们采用 NVIDIA 最新提供的硬件遮挡查询技术,通过硬件快速、直接地读取深度测试的比较结果.作为总结,图 2 给出了我们基于流碰撞检测基本算法的总体框架.

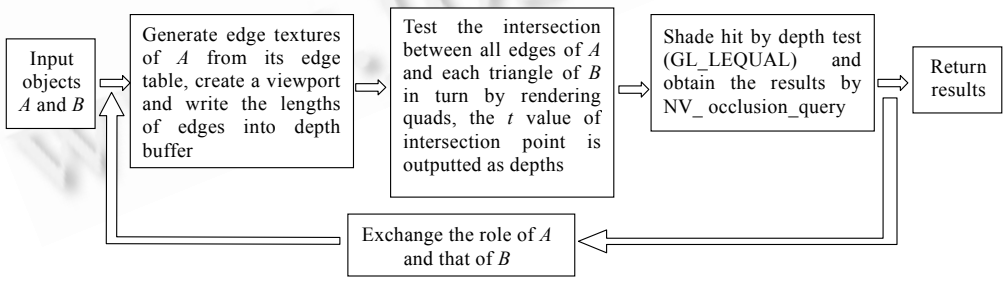


Fig.2 The general framework of the basic streaming-based collision detection algorithm

图 2 基于流的碰撞检测基本算法总体框架

3 边与三角形求交

边射线和三角形的相交检测过程由可编程 GPU 自定义的片断绘制程序实现.可以通过多种实时绘制语言

定义片断绘制程序,如北卡罗莱纳大学 PixelFlow 系统下的 PfMan[21],SGI 的 Interactive Shading Language[22],斯坦福大学的 Real-Time Shading System[23],NVIDIA 的 C for Graphics(Cg)[31]等.我们采用 NVIDIA 公司的实时绘制语言 Cg 实现用于边与三角形求交计算的片断绘制程序.像素缓存技术为采用可编程 GPU 进行通用计算提供了一个平台[29],因此我们采用了该技术.同时,为了克服片断绘制程序通常只能进行定点计算的不足,提高算法的精确度,还采用了浮点缓存技术[30],以便在像素缓存技术的基础上在绘制过程中正确处理浮点数值的计算.

3.1 数据流输入

片断绘制程序是 GPU 绘制流水线的一部分,它能够利用多纹理对单个像素进行算术操作并获取纹理采样数据.自然,其数据的输入输出必须遵从绘制流水线的规则.我们提出的基于流的实时碰撞检测算法通过纹理映射控制数据流的输入,经自定义片断绘制程序处理后,输出相应的结果.

A 的所有边与 B 的一个三角形求交测试过程需要输入的数据包括两部分:一是 A 的边集,二是 B 的三角形的顶点坐标和法向量.如前所述,边集信息由边纹理予以编码.由于纹理图像与视窗口的大小相同,经两次纹理映射后即可得到视窗口每个像素和 A 的每条边之间的一个一一映射.

三角形的顶点坐标和法向量则以多个纹理坐标的方式存入所建立的与视窗口大小一致的矩形的顶点属性内.矩形的 4 个顶点属性设置相同,均保存了三角形的所有信息.如图 3 所示,v_1,v_2,v_3 表示某个三角形的 3 个顶点坐标,n 为三角形的法向量,这些向量被写入矩形的纹理坐标向量[30].该矩形的顶点属性首先通过图形流水线的顶点绘制程序的处理,得到向量 $v_1,v_2,n,e_{12}=v_2-v_1$ 和 $e_{13}=v_3-v_1$ 作为片断绘制程序的输入参数,然后由光栅化过程插值均匀分布到视窗口的每个像素上.

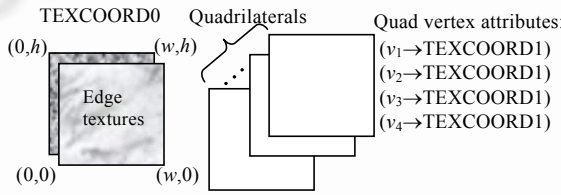


Fig.3 Input data stream

图 3 数据流的输入

采用矩形纹理技术的纹理坐标 (s,t) 是可变的,不一定必须是常数.因此,我们将所建立的矩形的 4 个顶点的纹理坐标分别设置为 $(0,0),(w,0),(w,h)$ 和 $(0,h)$.光栅化过程将纹理坐标插值到每个像素上.片断绘制程序利用这些纹理坐标可访问边纹理中的每个像素.

3.2 相交计算

Möller 等人提出了一个非常有效的射线与三角形求交算法[32].我们对其进行改造,使其适用于基于流的碰撞检测算法中的片断绘制程序.改进后的边与三角形求交算法描述如下:

$v_1o = orig - v_1$, $v_2o = orig - v_2$,

$v_1od = v_1o \times dir$, $v_2od = v_2o \times dir$,

$e_{12}d = e_{12} \times dir$, $\det = e_{13} \cdot e_{13}d$,

$t = -(n \cdot v_1o)/(n \cdot dir)$,

$u = e_{13} \cdot v_1od / \det$,

$v = -e_{12} \cdot v_1od / \det$, $w = u + v$.

当 u,v,w 均处于[0,1]区间时,边与三角形相交,同时返回交点的 t 参数.否则,二者不发生相交,返回 t 参数最大限定值作为深度值输出用于深度测试.下面是用于边与三角形求交的片断绘制程序的 Cg 代码.

```
float 4 Detect_Rays_with_Triangle(out float t, in float3 orig, in float3 dir, float3 v1, float3 v2, float n, flaot3 e12,
float3 e13)
{
  float3 v1o=orig–v1;                    float3 v2o=orig–v2;
  float3 v1od=cross(v1o,dir);            float3 v2od=cross(v2o,dir);
  float3 e12d=cross(e12,dir);
  bool bHit=false;                       float det=dot(e13,e12d);
```

```
    bHit=(det<–0.000001f)?true:bHit;        bHit=(det>=0.000001f)?true:bHit;
    float u=dot(e13,v1od)/det;              float v=–dot(e12,v1od)/det;
    float w=u+v;

    bHit=(u>=0.0f)?bHit:false;              bHit=(u<1.0f)?bHit:false;
    bHit=(v>=0.0f)?bHit:false;              bHit=(w<1.0f)?bHit:false;

    t=(bHit)?(–dot(n,v1o)/dot(n,dir)):MaxT;
    return bHit;
}
```

3.3 结果输出

边与三角形求交后得到的交点 t 值由片断绘制程序作为深度值输出,对于不与三角形相交的边则以最大深度值输出.经过深度测试以后,可由 CPU 来查询深度缓存内的值以判别边是否与三角形相交,进而获取碰撞检测的结果.但是,由于个人计算机的异步 AGP 总线结构,从 GPU 中接收数据比向 GPU 发送数据要慢得多,由 CPU 读取深度缓存的操作对于实时碰撞检测而言开销较大.幸运的是,NVIDIA 提供的硬件遮挡查询技术可通过硬件直接获取深度测试的比较结果.因此,我们采用该技术快速获取碰撞检测输出结果,从而巧妙地避免了由于图形硬件读写速度不对称、相对要慢得多的读取深度缓存操作.

4 优化的基于流的碰撞检测算法

基于流的碰撞检测基本算法中,碰撞检测过程被转化为图形硬件,每次绘制时进行多条线段和一个三角形的相交检测,算法总体效率取决于边的个数、三角形个数、读取速率和填充速率.读取速率和填充速率的大小主要取决于图形硬件的性能.因此,为提高算法的效率,应当尽可能地减少输入数据流中的边的个数和三角形个数.基于上述分析,我们给出如下两种优化方法.

4.1 层次树优化法

层次树优化法的目的就是尽量减少输入待检测的三角形个数.对物体 B 进行预处理,将其组织为一棵层次包围盒二叉树,物体 A 的所有边先与物体 B 层次包围盒树的根节点包围盒求交.该求交过程同样由 GPU 的片断绘制程序实现.如果所有边都不与之相交,则返回两物体不相交的结果;否则,递归判断其两个子节点包围盒是否与所有边相交,直到叶节点中的三角形.我们采用 OBB 树作为层次包围盒树[7].

片断绘制程序实现边与包围盒求交检测,其方法与边与三角形求交类似,只不过所建立的矩形顶点属性存放的是与包围盒有关的参数.矩形的所有顶点属性均用两个辅纹理坐标保存包围盒的最大和最小坐标向量.在视窗口中绘制矩形,由另一自定义的片断绘制程序进行边与包围盒的相交检测.Williams 等人给出了一种快速可靠的射线与包围盒求交的算法[33],经过改造可用于片断绘制程序中.

优化算法使用层次包围盒树可快速排除多数与边集不相交的三角形,从而减少所需绘制的矩形个数,提高算法总体效率.

4.2 子纹理优化法

子纹理优化法的目的是减少每次求交检测中所涉及的边的个数.当物体 A 所有边与物体 B 的一个节点包围盒发生相交时,保留所有相交的边,并重新组织构建两个新的边子纹理.为获取物体 A 所有边与 B 节点包围盒的相交结果,需要读取深度测试的结果.然而,硬件遮挡查询操作只能获取相交边的个数,无法快速获取相交边对应的像素位置.因此,优化算法在深度测试的同时增加了模板缓存测试,用模板缓存来识别相交边对应的像素位置.这样,可依据模板缓存来创建两个新的更小的边子纹理分别与当前节点的两个子节点包围盒进行求交计算,如图 4 所示.如此递归计算下去,直到叶节点中的三角形.典型地,由于边纹理在求交计算中逐层变小,GPU 所需绘制的像素个数也逐步减少,从而可大大加快绘制速度,提高算法整体效率.

4.3 优化的基于流的碰撞检测算法

基于上述两个优化方法,我们进一步提出了以下优化的基于流的碰撞检测算法.这里仍假定 A 的边纹理和 B 的 OBB 树已经创立.

首先,创建一个与 A 的边纹理大小相同的视窗口,并将 A 所有边的长度通过纹理映射写入深度缓存.其次,在视窗口中绘制一个与视窗同样大小的矩形,这个矩形的所有顶点属性中保存了 B 的 OBB 树根节点 OBB 包围盒的各项参数.绘制中,用片断绘制程序由 GPU 计算 A 的边是否与 B 的根 OBB 包围盒相交,通过深度比较和模板测试,并读出模板缓存,得到与根 OBB 包围盒相交的所有边的集合.再次,由给出的相交边重新创建新的边纹理.最后,递归计算新的边纹理是否与 OBB 下面两子节点 OBB 包围盒相交;若当前 OBB 包围盒为叶子节点,则在视窗口绘制的矩形各顶点属性改为包围盒所包含三角形的各顶点位置和法向量,通过基本算法的片断绘制程序计算剩下的边是否与三角形相交,并通过硬件遮挡查询快速获取检测结果.

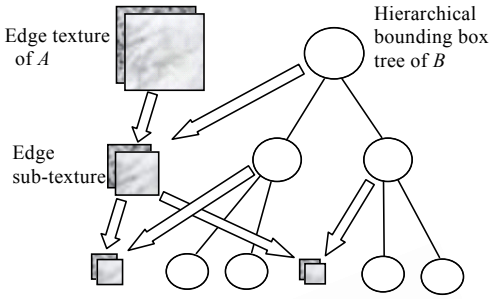


Fig.4 Sub-Textures and hierarchical bounding box tree
图 4 子纹理图与层次包围盒树

图 5 展示了优化的基于流碰撞检测算法的具体流程图.

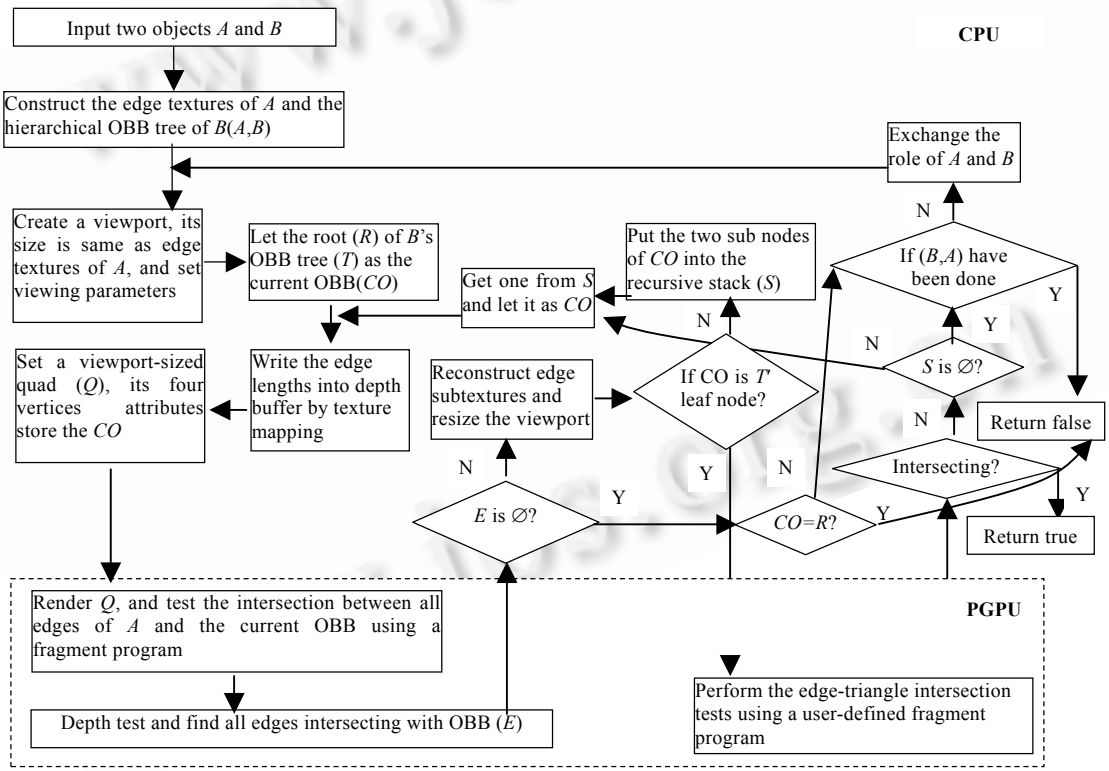


Fig.5 The flowchart of the optimized streaming-based collision detection algorithm
图 5 优化的基于流碰撞检测算法流程图

5 实验与结果讨论

我们在 CPU P4 1.6GHz 的 PC 机(内存 512M,显卡 NVIDIA Geforce FX 5800,显存 128M)上实现了该算法.NVIDIA Geforce FX5800 的图形卡支持几乎所有可编程特性,如像素缓存、浮点缓存和顶点与片断绘制程序等.我们测试了多个不同复杂度的场景,并与图像空间的典型算法 RECODE[14]和物体空间的经典算法 RAPID[7]进

行了比较.图 6 是 3 个典型测试场景,其三角形总数分别为 21 292,18 217 和 40 419.在这些场景下,用 CPU 软件模拟实现我们的算法无法实时运行,计算时间每帧超过了 10s.对于所测试的场景,我们基于可编程 GPU 的算法性能不仅优于 RECODE,而且对于复杂场景,其性能甚至明显超过了 RAPID.实验结果见表 1 并如图 7 所示.通过实验结果可以看出,算法有效地利用了可编程 GPU 高性能的计算能力.相对于 CPU 软件仿真算法,基于流的碰撞检测算法充分利用了图形硬件的并行特点,具有更高的效率.两种优化方法在较高程度上提升了算法总体效率,且场景越复杂,效率提升得越显著.

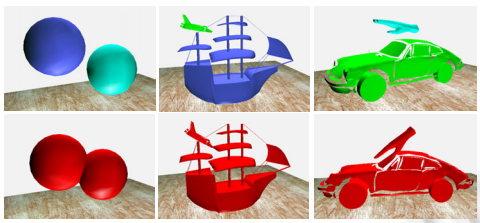


Fig.6 Three testing scenarios
(the below three pictures show collision)
图 6 3 个测试场景(下行 3 个图表示碰撞)

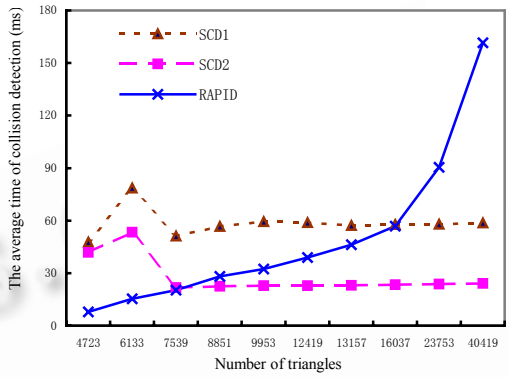


Fig.7 The comparison in scenario 3 of varying complexity
图 7 场景 3 复杂度变化的比较

Table 1 The experimental results
表 1 实验结果

Scenarios		SCD0	SCD1	SCD2	RECODE	RAPID
1	t_{cd}	6 627.5	47.6	23.0	29.3	1.0
	Speed	1	139.2	288.2	226.2	6 627.5
2	t_{cd}	2 256.0	53.2	40.4	–	1.9
	Speed	1	42.4	55.8	–	1 187.4
3	t_{cd}	15 587.4	58.7	24.1	–	161.6
	Speed	1	265.5	646.8	–	96.5

表 1 中,SCD0 表示基本的基于流碰撞检测算法;SCD1 是采用了 OBB 树的基于流碰撞检测算法;SCD2 是优化的基于流碰撞检测算法;RECODE 为文献[14]中提出的算法;RAPID 为文献[7]中提出的算法;t_{cd} 表示平均碰撞检测时间(ms);Speed 是各种方法相对 SCD0 的加速比.由于场景 2 和场景 3 中包含有非凸体,因此,有关 RECODE 的实验数据空缺.

6 结 论

随着 GPU 性能的快速增长及灵活的可编程能力的出现,可编程 GPU 正逐步成为一个新的通用计算平台.本文提出了一种基于可编程 GPU 的实时碰撞检测算法,将碰撞检测过程转化为流计算模式后映射到可编程 GPU 上并行执行并由 CPU 获取检测结果.在对算法复杂度进行分析的基础上,依据算法本身特点,还提出了两种有效的优化方法,显著提高了算法的总体效率.与常规图像空间碰撞检测算法相比,算法除了在性能上的优势之外,还具备以下两个特点:

(1) 精确性:一般的图像空间碰撞检测算法的精度取决于绘制视窗分辨率的大小,而本算法精度由浮点数存储方式决定,与基于几何的碰撞检测算法精度一致.

(2) 通用性:可处理任意三角形网格物体,而一般的图像空间算法通常仅能处理凸体.

基于流的碰撞检测算法在突破了基于常规图像空间碰撞检测算法在精确度和适应性上的局限性的同时,保持了图形硬件算法的特点,其效率与 GPU 性能密切相关,而目前 GPU 性能增长的曲线要明显高于 CPU 性能增长的莫尔曲线.可以预测,随着 GPU 的高速发展,本算法及与本算法类似的其他算法将具有更广阔的前景.

今后的工作将主要集中于以下两个方面:

(1) 将算法扩展到可用于处理变形物体间的实时碰撞检测;

(2) 进一步优化算法,提高效率.

References:

[1] Lin M, Gottschalk S. Collision detection between geometric models: a survey. In: Proc. of the IMA Conf. on Mathematics of Surfaces. 1998. 37~56.

[2] Jiménez P, Thomas F, Torras C. 3d collision detection: A survey. Computers and Graphics, 2001,25(2):269~285.

[3] Wang ZQ, Hong JZ, Yang H. A survey of collision detection problem. Journal of Software, 1999,10(5):545~551 (in Chinese with English abstract).

[4] Fan ZW, Wan HG, Gao SM. A parallel algorithm for rapid collision detection. Chinese Journal of System Simulation, 2000,12(5): 548~552 (in Chinese with English abstract).

[5] Wei YM, Wu QY, Shi JY. Research on fixed direction hull bounding volume in collision detection. Journal of Software, 2001,12(7): 1056~1063 (in Chinese with English abstract).

[6] Wang ZQ, Zhao QP, Wang CW. A fast algorithm to calculate collision point between convex polygons. Journal of Software, 1999, 10(12):1253~1258 (in Chinese with English abstract).

[7] Gottschalk S, Lin M, Manocha D. Obb-Tree: A hierarchical structure for rapid interference detection. In: Proc. of the ACM Siggraph'96. 1996. 171~180.

[8] Cohen J, Lin M, Manocha D, Ponamgi M. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In: Proc. of the ACM Interactive 3D Graphics Conf. 1995. 189~196.

[9] Klosowski JT, Held M, Mitchell JSB, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Trans. on Visualization and Computer Graphics, 1998,4(1):21~36.

[10] Chung K, Wang W. Quick collision detection of polytopes in virtual environments. In: Proc. of the ACM Symp. on Virtual Reality Software and Technology. ACM Press, 1996. 125~132.

[11] Ehmann S, Lin M. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In: Proc. of the Eurographics Conf. 2001. 500~510.

[12] Shinya M, Forgue M. Interference detection through rasterization. Journal of Visualization and Computer Animation, 1991,2: 131~134.

[13] Rossignac J, Megahed A, Schneider BO. Interactive inspection of solids: Cross-Section and interferences. Computer Graphics, 1992,26(2):353~360.

[14] Baciu G, Wong SKW, Sun H. RECODE: An image-based collision detection algorithm. Journal of Visualization and Computer Animation, 1999,10(4):181~192.

[15] Hoff III KE, Zaferakis A, Lin M, Manocha D. Fast and simple 2D geometric proximity queries using graphics hardware. In: Proc. of the ACM Symp. on Interactive 3D Graphics. 2001. 145~148.

[16] Kim YJ, Lin M, Manocha D. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. In: Symp. on Computational Geometry 2003. 2003. 386~387.

[17] Fan ZW, Wan HG, Gao SM. IBCD: A fast collision detection based on image space using OBB. Journal of Visualization and Computer Animation, 2003,14(4):169~181.

[18] Fan ZW, Wan HG, Gao SM. A fast collision detection based on image space. Chinese Journal of CAD&CG, 2002,14(9):805~809 (in Chinese with English abstract).

[19] Govindaraju NK, Redon S, Lin M, Manacha D. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In: Proc. of the ACM Siggraph/Eurographics Workshop on Graphics Hardware. 2003. 25~32.

[20] Lindholm E, Kilgard MJ, Moreton H. A user-programmable vertex engine. In: Proc. of the Siggraph. 2001. 149~158.

[21] Olano M, Lastra A. A shading language on graphics hardware: the PixelFlow shading system. In: Proc. of the SIGGRAPH'98. 1998. 159~168.

[22] Peercy MS, Olano M, Airey J, Ungar PJ. Interactive multi-pass programmable shading. In: Proc. of the SIGGRAPH 2000. 2000. 425~432.

[23] Proudfoot K, Mark WR, Tzvetkov S, Harnrahan P. A real-time procedural shading system for programmable graphics hardware. In: Proc. of the SIGGRAPH 2001. 2001. 159~170.

[24] Krüger J, Westermann R. Acceleration techniques for GPU-based volume rendering. In: Proc. of the 14th IEEE Visualization Conf. 2003. 38~43.

[25] Carr, Nathan A, Jesse D. Hall and John C. Hart, The ray engine. In: Proc. of the Graphics Hardware. 2002. 37~46.

[26] Purcell TJ, Buck I, Mark WR, Hanrahan P. Ray tracing on programmable graphics hardware. ACM Trans. on Graphics, 2002,21(3):703~712.

[27] Bolz J, Farmer I, Grinspun E, Schröder P. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In: Proc. of the SIGGRAPH 2003. 2003. 917~924.

[28] Harris MJ, Coombe G, Scheuermann TA. Lastra, physically-based visual simulation on graphics hardware. In: Proc. of the SIGGRAPH 2002/ Eurographics Workshop on Graphics Hardware 2002. 2002. 109~118.

[29] Wynn C. Using *p*-buffers for off-screen rendering in OpenGL. NVIDIA Corporation. http://developer.nvidia.com/docs/IO/1293/ATT/GDC01_PixelBuffers.pdf

[30] NVIDIA 2003, NVIDIA OpenGL extension specifications. http://developer.nvidia.com/object/nvidia_opengl_specs.html

[31] Mark WR, Glanville S, Akeley K. Cg: A system for programming graphics hardware in a C-like language. Proc. of the SIGGRAPH 2003, 2003,22(3):896~900.

[32] Möller T, Trumbore B. Fast, Minimum storage ray-triangle intersection. Journal of Graphics Tools, 1997,2(1):21~28.

[33] Williams AL, Barrus S, Morley RK, Shirley P. An efficient and robust ray-box intersection algorithm. 2001. http://www.cs.utah.edu/~awilliam/box/

附中文参考文献:

[3] 王志强,洪嘉振,杨辉.碰撞检测问题研究综述.软件学报,1999,10(5):545~551.

[4] 范昭炜,万华根,高曙明.基于并行的快速碰撞检测算法.系统仿真学报,2000,12(5):548~552.

[5] 魏迎梅,吴泉源,石教英.碰撞检测中的固定方向凸包包围盒的研究.软件学报,2001,12(7):1056~1063.

[6] 王兆其,赵沁平,汪成为.一个计算凸多面体间碰撞点的快速算法.软件学报,1999,10(12):1253~1258.

[18] 范昭炜,万华根,高曙明.基于图像的快速碰撞检测算法.计算机辅助设计与图形学学报,2002,14(9):805~809.