

浙江大学

博士 学位 论文



论文题目 实时碰撞检测技术研究

作者姓名 范昭炜

指导教师 高曙明 研究员

万华根 副研究员

学科(专业) 计算机应用技术

浙江大学 CAD&CG 国家重点实验室

浙江大学计算机学院

二零零三年十月二十九日

Research on Real Time Collision Detection



By Zhaowei Fan

Advisor : Prof. Shuming Gao

Associate Prof. Huagen Wan

A Dissertation Presented to
the Graduate School of Zhejiang University
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

State Key Laboratory of CAD&CG
College of Computer Science
Zhejiang University, P.R. China
October 2003

摘要

实时碰撞检测是机器人、动画仿真、虚拟现实等领域中一个非常关键的问题，其基本任务是确定两个或多个物体彼此之间是否发生接触或穿透。尽管针对碰撞检测已有了大量有价值的研究成果，但随着诸如虚拟现实等新兴领域的涌现及随之而来的人们对交互实时性、场景真实性要求的不断提高，碰撞检测技术所面临的问题也日益突出，其中最核心的问题是如何有效地提高碰撞检测的速度。

在对各类碰撞检测算法作出全面了解、透彻分析的基础上，针对碰撞检测技术目前存在的问题，本文分别从以下三个角度出发，设计、实现并验证了一组新的碰撞检测算法，对如何利用图形硬件的高计算性能、可编程性及多处理机的并行计算能力，加速碰撞检测过程进行了有益的探索性研究：

1. 从利用图形硬件辅助通用计算的角度出发，研究并提出一种新的基于图象空间的快速碰撞检测算法，通过将图形硬件的计算优势和简化的几何模型表示相结合，以实现复杂物体间的实时碰撞检测。算法在继承一般基于图象空间的碰撞检测算法优点的同时，突破了它们的局限性，能够在保证效率的前提下处理任意形状多面体之间的碰撞检测问题。算法首先自动将物体表面分解为凸面片，构建与凸面片相对应的凸包围体，并将凸分解结果合理地组织成层次二叉树结构以有效利用物体的空间连贯性；同时采用三角形带压缩这一绘制加速技术加快碰撞检测阶段的绘制速度。为进一步提高碰撞检测的效率，在具体绘制操作之前，采用了 OBB 包围盒树技术以尽可能早地排除彼处不发生碰撞的凸包围体，并有效利用了 OBB 包围盒重叠检测的结果以设置视域参数。
2. 从利用可编程图形硬件高并行性的角度出发，探索性地采用可编程图形硬件来解决复杂物体间的实时碰撞检测问题，提出一种基于流的、精确快速的碰撞检测算法。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进一步提升算法效率。
3. 从利用多处理机并行计算能力的角度出发，提出一种并行的快速碰撞检测算法。该算法在对动态复杂场景中的物体建构大致平衡的层次包围盒树的基础上，通过采用一种新的边包围体遍历策略有效地减少并行碰撞检测算法并行任务个数，加大并行的粒度，从而提高并行计算的加速比，加快碰撞检测的速度。算法采用多线程技术实现并行计算，使其在单处理机系统和多处理机

系统上均能运行。

关键词： 碰撞检测，实时，平行计算，图象空间，表面凸分解，图形硬件，可编程，流计算模式

Abstract

Real time collision detection is one of the most important problems in the fields of robotics, computer animation, and virtual reality, etc. Its fundamental task is to detect whether there are contacts or penetrations between two or among multiple objects. Indeed, there have been many research achievements on solving the problem of collision detection, this problem, however, is yet to be solved with the emergence of such burgeoning techniques as virtual reality, and the demands of real time interactivity and realistic simulation of motions of virtual objects thereafter.

Based on the intensive survey and comprehensive analyses of various approaches to collision detection, several algorithms have been designed, implemented and verified to explore the possibility of speeding up the process of collision detection from, respectively, the perspectives of using the high computing power of graphics hardware, exploring the programmability of graphics processing units (GPUs), and utilizing the parallel computing performance of multi-processor machines.

- a) From the perspective of using the high computing power of graphics hardware, a fast collision detection algorithm based on image space is presented, which combines graphics hardware capabilities with a simplified geometric representation to facilitate the process of collision detection between complex geometric objects. The method can process arbitrarily shaped mesh objects, while preserving the merits of image-based collision detection algorithms. This is achieved by decomposing the surfaces of an object into a list of convex pieces. High efficiency of the algorithm is obtained by organizing the convex pieces into a binary tree with nodes composed of convex pieces, and by adopting triangle strip compression, which is a kind of real time rendering techniques. Furthermore, the oriented bounding box (OBB) tree, is used to exclude those convex pieces that don't interfere with each other as early as possible, and the result of OBB overlapping test is used to facilitate the viewing parameter setting for real time rendering.
- b) From the perspective of exploring the programmability of GPUs, a streaming collision detection algorithm is proposed. The algorithm explores to solve the problem of real time collision detection between complex objects using programmable GPUs. It maps the geometry computation of collision detection between two arbitrary objects into programmable GPUs to match the parallel architecture of graphics hardware, and produces on the fly the collision detection

results via real time rendering. To do so, the problem of collision detection 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 optimization techniques are proposed to further improve the efficiency of the algorithm.

- c) From the perspective of utilizing parallel computing performance of multi-processor machines, a parallel collision detection algorithm is presented. The algorithm first builds an approximately balanced box tree for each arbitrary object in question using an adaptive space subdivision scheme, then traverses the built box trees in a parallel way to detect the occurrences of collisions. In particular, a coarse-grained traversing scheme, called the edge-box traversing scheme, is proposed and adopted to utilize the parallel architectures of multi-processor machines in an efficient way. The algorithm falls into the class of synchronous parallel algorithms and is implemented with multiple threads, which enable the algorithm to run on both single processor computers and multi-processor machines.

Keywords: collision detection, real time, parallel computing, image space, surface convex decomposition, graphics hardware, programmable, streaming computing

目 录

摘要	I
Abstract	III
目录	V
第一章 绪论	1
1. 1 碰撞检测算法分类.....	3
1. 1. 1 基于时间域的碰撞检测算法.....	3
1. 1. 2 基于空间域的碰撞检测算法分类	4
1. 2 碰撞检测算法的一般框架.....	10
1. 2. 1 初步检测阶段.....	10
1. 2. 2 详细检测阶段.....	12
1. 3 主要算法介绍.....	15
1. 3. 1 面向凸体的碰撞检测算法.....	15
1. 3. 2 基于一般表示的碰撞检测算法.....	18
1. 3. 3 基于层次包围体树的碰撞检测算法.....	21
1. 3. 4 基于图象空间的碰撞检测算法.....	25
1. 4 目前存在的问题.....	28
1. 5 研究目标和研究内容.....	30
第二章 基于图象的快速碰撞检测算法	31
2. 1 引言.....	31
2. 2 算法概述.....	34
2. 3 算法预处理阶段.....	35
2. 3. 1 基于表面的凸分解.....	35
2. 3. 2 建构层次二叉树.....	38
2. 3. 3 建构凸块的 OBB 包围盒.....	39
2. 3. 4 凸块的三角形带压缩编码.....	40

2.4 碰撞检测核心算法.....	43
2.4.1 层次二叉树的遍历.....	43
2.4.2 凸块 OBB 包围盒的相交检测方法.....	44
2.4.3 用于相交检测的视域参数的设置.....	46
2.4.4 凸块之间的相交检测.....	47
2.5 实验结果与分析.....	52
2.6 小结.....	56
第三章 基于流的实时碰撞检测算法.....	57
3.1 引言.....	57
3.2 流计算模式.....	59
3.2.1 可编程图形硬件.....	59
3.2.2 流计算模式.....	60
3.3 基于流的碰撞检测基本算法.....	61
3.3.1 边纹理生成.....	61
3.3.2 边与三角形求交.....	62
3.3.3 求交结果获取.....	62
3.4 边与三角形求交.....	64
3.4.1 数据流输入.....	64
3.4.2 相交计算.....	66
3.4.3 结果输出.....	67
3.5 优化的基于流的碰撞检测算法.....	69
3.5.1 层次树优化法.....	69
3.5.2 子纹理优化法.....	70
3.5.3 优化的基于流的碰撞检测算法.....	71
3.6 实验与结果讨论.....	73
3.7 小结.....	77
第四章 并行碰撞检测算法.....	79
4.1 引言.....	79
4.2 并行算法的设计思想.....	80
4.2.1 并行算法的定义与分类.....	80
4.2.2 并行计算模型.....	80

4.2.3 并行算法的设计	82
4.3 简单并行碰撞检测算法	82
4.3.1 基本碰撞检测算法	82
4.3.2 建构平衡包围盒树	83
4.3.3 简单串行碰撞检测算法	85
4.3.4 简单并行碰撞检测算法	88
4.4 基于边包围体遍历策略的并行碰撞检测算法	89
4.5 算法实现	92
4.5.1 算法实现	92
4.5.2 实验结果	92
4.6 小结	95
 第五章 总结与展望	97
 5.1 全文工作总结	97
5.2 今后工作展望	98
 参考文献	99
 攻读博士学位期间发表或录用的论文情况	111
 致谢	113

第一章 绪论

摘要

本章首先介绍了碰撞检测领域的研究背景和研究意义，并对目前现有的碰撞检测算法进行了分类归纳，同时总结了一般碰撞检测算法所采用的总体框架。接着，着重介绍与分析了目前四类主要的碰撞检测算法。最后指出了实时碰撞检测技术中存在的主要问题，描述了本文的主要研究目标及主要研究内容。

关键词： 碰撞检测，虚拟现实，计算机仿真

碰撞检测是机器人、动画仿真、虚拟现实等领域不可回避的问题之一，其基本任务是确定两个或多个物体彼此之间是否发生接触或穿透。之所以需要在物体间进行碰撞检测，是基于现实世界中的一个简单观察事实，即，在不破坏物体的前提下，两个或多个物体不可能同时占有同一空间区域。

有关碰撞检测问题的研究源于 1970 年代。当时，在机器人路径规划、自动装配规划等领域中，为检测机器人与场景中的物体或零件与零件之间是否发生碰撞，产生了一系列碰撞检测算法。但由于当时的计算条件及具体应用的特性（如事先可预知机器人运动的轨迹）等因素，碰撞检测计算大都不是实时完成的。

随着计算机软硬件及网络等技术的日益成熟，尤其是计算机动画仿真、虚拟现实等技术的快速发展，人们迫切希望能对现实世界进行真实模拟，而这其中亟需的关键技术之一即是实时碰撞检测。目前，三维几何模型越来越复杂，虚拟环境的场景规模越来越大，同时人们对交互的实时性、场景的真实性的要求越来越高。严苛的实时性和真实性要求在向研究者们提出巨大挑战的同时，也令实时碰撞检测再度成为研究热点。

1.1 碰撞检测算法分类

近三十年来，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，其中有些工作影响重大。碰撞检测算法种类繁多，对其分类也可谓是仁者见仁、智者见智。本文从两个角度对碰撞检测算法进行分类：一是从时间域的角度来分；二是从空间域的角度来分。下面分别进行阐述。

1.1.1 基于时间域的碰撞检测算法分类

从时间域的角度来分，碰撞检测算法可分为静态碰撞检测算法、离散碰撞检测算法和连续碰撞检测算法三类。

静态碰撞检测算法是指当场景中物体在整个时间轴 t 上都不发生变化时，用来检测在这个静止状态中各物体之间是否发生碰撞的算法。离散碰撞检测算法则是在时间轴的每个离散点 t_0, t_1, \dots, t_n 上不断地检测场景中所有物体之间是否发生碰撞的算法。而连续碰撞检测算法是指在一个连续的时间间隔 $[t_0, t_n]$ 内，判断运动物体是否与其他物体相交的算法。

静态碰撞检测问题在计算几何中有着广泛的研究，一般对这类算法没有实时性的要求，但往往对算法精度要求较高 [Dobkin 1985][Agarwal 1991][Chazelle 1989]。

从本质上说，离散碰撞检测算法在每一时间离散点上可以通过类似于静态碰撞检测算法的方法来实现的，但它更注重算法效率。从整个时间轴来看，由于算法的时间离散特性，这类算法至少存在以下两个问题：(1) 存在刺穿现象。当时间步长过大时，两物体可能已发生了一定深度的刺穿才被检测到已发生碰撞，因此无法保证物体的运动真实性；(2) 会遗漏发生碰撞的情况。对于较狭窄的物体，当运动物体在相邻时间离散点上的两个位置恰好处于该狭窄物体两侧时，离散算法将无法正确地检测出物体所发生的碰撞。

连续碰撞检测算法的研究一般涉及到四维时空问题或结构空间精确的建模 [Cameron 1990][Canny 1986][Redon 2001][Redon 2002a]，这类算法能较好地解决离散碰撞检测算法存在的上述两个问题，但通常计算速度比较慢，尤其是在大规模场景中无法实现实时的碰撞检测。

离散碰撞检测算法尽管存在一些问题，但由于其检测过程的快速性能较好地迎合多数应用对实时碰撞检测的需求，所以仍是目前碰撞检测算法研究的重点和热点 [Lin 1998][Jiménez 2001]。此外，人们还可通过一些优化方法在一定程

度上减轻或降低离散碰撞检测算法的上述两个问题的影响。例如，自适应步长技术和可中断的碰撞检测技术等均有助于改善这两个问题[Hubbard 1995] [Dingliana 2000] [Dingliana 2001] [O'Sullivan 1999]。

1.1.2 基于空间域的碰撞检测算法分类

从空间域的角度来分，碰撞检测算法大体可分为两大类：一类是基于物体空间的碰撞检测算法；一类是基于图象空间的碰撞检测算法。

这两类算法的主要区别在于是利用物体三维几何特性进行求交计算还是利用物体二维投影的图象加上深度信息来进行相交分析。目前在基于物体空间的碰撞检测上，研究人员已经做了相当多的工作[Lin 1998][Jiménez 2001]。基于图象空间的碰撞检测算法是一类比较新的碰撞检测算法，它能有效利用图形硬件的绘制加速功能来提高碰撞检测算法的效率。近几年图形硬件技术的飞速发展，图形加速卡在性能不断迅速提高的同时甚至出现了可编程的功能，使得基于图象空间的碰撞检测算法进入了一个新的发展阶段[Lin 2002]。

1.1.2.1 基于物体空间的碰撞检测算法

基于物体空间的碰撞检测算法一直是人们研究的重点，已有相当的研究成果。研究人员把各种技术如层次表示法、几何推理、代数范式、空间划分、解析方法和最优化方法等应用到碰撞检测中[Lin 1998][王志强 1999][Jiménez 2001]。本文从两个方面来进一步区分基于物体空间的碰撞检测算法：一是基于物体的不同表示模型；二是基于不同的空间结构。

(1) 采用一般表示模型的碰撞检测算法

碰撞检测算法与物体采用的表示模型相关。面向不同表示模型的碰撞检测算法各有特点，算法效率也有不同。目前在 CAD/CAM 和三维图形领域中存在多种几何模型，但其中最主要的是多边形表示模型和非多边形表示模型两类。

多边形表示模型又可再细分为多边形集合（polygon soup）和结构化模型两类；而非多边形表示模型则可分为 CSG 表示模型、隐函数曲面、参数曲面和体表示模型等。图 1-1 给出了几何模型的一个分类。

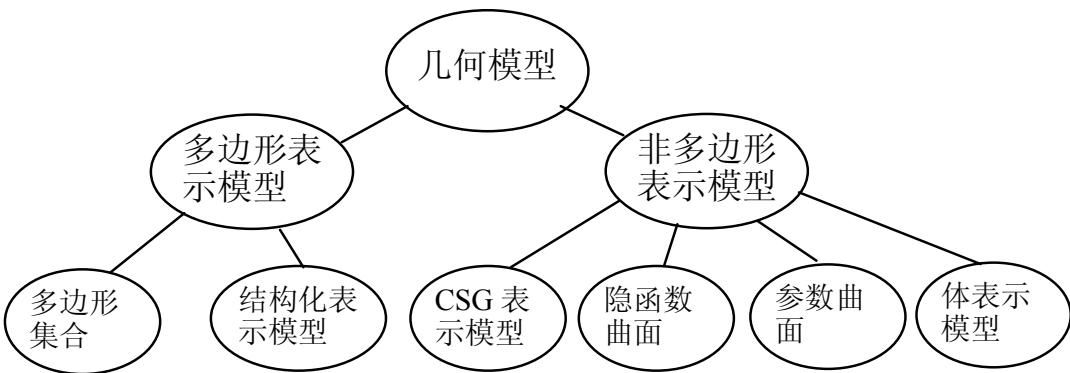


图 1-1 几何模型表示法分类

多边形表示模型是计算机图形学中最常用的几何模型，其特点是表示简单、通用，因此很多碰撞检测算法都面向多边形表示模型设计。多边形表示模型中多边形集合是一种常用的几何表示模型。这种表示模型中，物体表面被简单地表示为一组多边形的集合。面向这种表示模型的碰撞检测算法[Hubbard 1995][Gottschalk 1996] [Klosowski 1998][Zachmann 1998]一般有较广的适用范围。

当多边形表示模型的表面多边形形成了一个封闭的闭包，并由此确定了内部和外部区域，这种模型就称为多边形表示的结构化模型或称为多面体。多面体模型又有凸多面体和非凸多面体之分。为叙述简便，本文将凸多面体简称为凸体，将非凸多面体简称为非凸体。多面体为凸的含义是多面体内部的任意两点连线完全处于该多面体的内部。多面体尤其是凸体良好的空间结构特性如空间连贯性可被利用来优化碰撞检测的效率。因此，基于多面体，尤其是基于凸体的碰撞检测算法一直是碰撞检测算法中的一个研究重点[Lin 1998][Jiménez 2001]。

面向凸体的碰撞检测算法大体上又可分为两类：一类是基于特征的碰撞检测算法；另一类是基于单纯形（Simplex）的碰撞检测算法。这两类算法分别由两个从不同角度解决碰撞检测问题的经典算法发展演变而来。

基于特征的碰撞检测算法基本上都是源自于 Lin-Canny 的“最邻近特征算法”[Lin 1991, Lin1993]。它们都按照多面体的特征（如顶点、边、面等）进行区间剖分，并对每个特征建构其相应的 Voronoi 区域。一个特征的 Voronoi 区域是指那些距离当前特征比其他特征更近的点所组成的区域。图 1-2 显示了一个物体的顶点，边，和面所对应的 Voronoi 区域。这类算法依据 Voronoi 区域的特性利用物体空间的连贯性提高效率，其他主要代表算法有[Lin 1995], [Cohen 1995], [Chung 1996], [Mirtich 1998], [Ehmann 2000], [Ehmann 2001]。

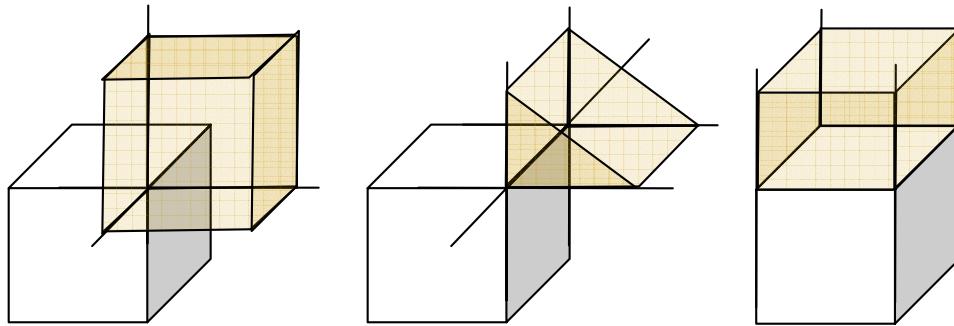


图 1-2 一个立方体顶点, 边和面分别对应的 Voronoi 区域

而基于单纯形的碰撞检测算法, 最初由 Gilbert、Johnson 和 Keerthi [Gilbert 1988][Gilbert 1990] 提出, 即 GJK 算法。这类算法通常将两物体看作一个单纯形, 通过单纯形的几何特性来计算它们之间的分离或刺穿距离。[Cameron 1997][Bergen 1999] 对 GJK 算法作出了有益的改进, 使算法具有良好的实时性和强壮性。

几何模型除了多边形表示模型外, 还有一类称为非多边形表示模型, 包括了 CSG 表示模型、隐函数曲面、参数曲面和体表示模型等。

CSG(Constructive Solid Geometry) 表示模型用一些基本体素如长方体、球、柱体、锥体和圆环等, 通过集合运算如并、交、差等操作来组合形成物体[Requicha 1992, Hoffmann 1989]。CSG 表示的优点之一是它使得物体形状的建构更直观, 即可以通过剪切(交、差)和粘贴(并)简单形状物体来形成更复杂的物体。这也使得它更容易找到碰撞的确切位置[Cameron 1991]。由 CSG 表示的特殊性, 基于该类模型表示的碰撞检测算法[Zeiller 1993][Su 1996][Poutain 2001] 主要面向 CAD 应用。

隐函数曲面是指由隐函数定义的曲面模型。当隐函数把三维空间映射到实数域上, $f: R^3 \rightarrow R$, 隐函数曲面就是由所有满足 $f(x,y,z) = 0$ 的点组成的曲面。这时隐函数明确地定义了物体内部, 即 $f(x,y,z) < 0$, 和物体外部, 即 $f(x,y,z) > 0$ 。因此隐函数曲面一般都具有封闭的理想特点。隐函数代数曲面的特例是二次曲面, 即在 x, y, z 上最高幂为二次的多项式所表示的曲面。它可以在一个统一的框架下表示锥体, 球体和圆柱体等曲面。由于二次曲面在许多实际工程中应用较广, 研究人员因此提出了一些面向二次曲面的特定碰撞检测算法[Farouki 1989][Miller 1991][Shene 1991] 来处理二次曲面之间的求交计算。

参数曲面表示是二维平面的子集到三维空间的映射, $f: R^2 \rightarrow R^3$ 。不同于隐函数曲面, 参数曲面一般来说不是封闭的。但参数曲面比隐函数曲面更易于用多边形来离散表示, 也更易于绘制。其中一个特殊的类别非均匀有理 B 样条曲线(NURBS) 在 CAD 领域中使用非常广泛[Lane 1980][Farin 1993]。基于参数曲

面的碰撞检测算法也有部分研究工作[Turnbull 1998]。

体表示模型用简单体素来描述物体对象的结构，其基本几何构件一般为立方体或四面体。与面模型不同，体模型一般用于软体对象的几何建模，它拥有对象的内部信息，能表达模型在外力作用下的变化特征（变形、分裂等），但其计算时间和空间复杂度也相应增加。目前，基于体模型的碰撞检测算法主要用于虚拟手术中可变形物体的碰撞检测[Heidelb 2003][Boyles1999][魏迎梅 2001]。此外，将复杂的三维物体转化为简单体素进行表示，进行处理，使得基于该表示的求交算法简单而高效，因此在对碰撞检测速度要求较高的场合也往往有一些应用[McNeely 1999]。

（2）采用空间结构的碰撞检测算法

物体空间的碰撞检测算法可采用不同的空间结构来提高效率，根据所用空间结构的不同可将它们分为两类：空间剖分法（space decomposition）和层次包围体树法(hierarchical bounding volume trees)。这两类方法都是通过尽可能减少进行精确求交的物体对或基本几何元素的个数来提高算法效率的。不同的是，空间剖分法采用对整个场景的层次剖分技术来实现，而层次包围体树法则是对场景中每个物体建构合理的层次包围体树来实现。

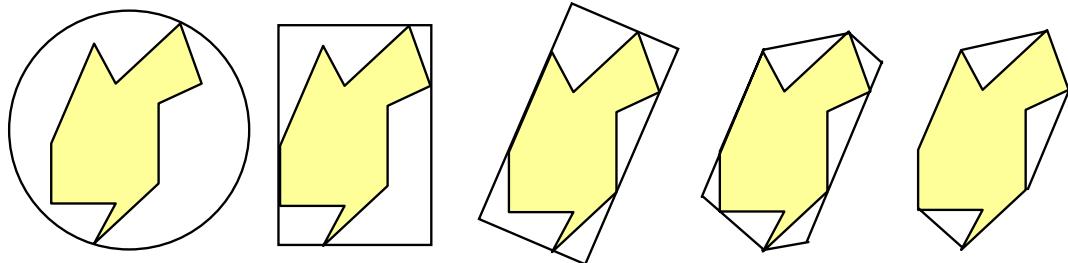
● 空间剖分法

场景的层次剖分方法主要有均匀剖分、BSP 树、k-d 树和八叉树（Octree）等[Samet 1989][Naylor 1990][Bourma 1991]。基于这些空间剖分技术的碰撞检测算法也频繁出现[Naylor 1990][Garcia 1994][Hamada 1996][McNeely 1999]。但这类碰撞检测算法较难在处理不同的场景和具有不同形状及复杂度的物体时保持比较一致的检测效率。

● 层次包围体树法

物体的层次包围体树可以根据其所采用包围体类型的不同来加以区分，主要包括层次包围球树[Hubbard 1993][Hubbard 1995][Palmer 1995][O’Sullivan 1999]、AABB 层次树（Aligned Axis Bounding Box）[Zachmann 1997][Bergen 1997][Larsson 2001]、OBB 层次树（Oriented Bounding Box）[Gottschalk 1996]、k-dop 层次树（Discrete Orientation Polytope）[Klosowski 1998][Zachmann 1998]、QuOSPO 层次树(Quantized Orientation Slabs with Primary Orientations) [He 1999]、凸块层次树[Ehmann 2001]以及混合层次包围体树[Wan 2001]等等。图 1-3 给出了各种包围体类型在二维上的示例。建构物体层次包围体树既可采取自顶

向下的策略，也可自底向上来进行。目前基于层次包围体树的算法多数采取自顶向下的方式来建构物体的层次包围体树。采用不同的层次包围体树的碰撞检测算法各有其优缺点，我们将在主要算法介绍中对其作进一步分析。



(a) 包围球 (b) AABB 包围盒 (c)OBB 包围盒 (d) 6-dop 包围体 (e)凸包包围体

图 1-3 包围体二维示意图

1.1.2.2 基于图象空间的碰撞检测算法

基于图象空间的碰撞检测算法一般利用图形硬件对物体的二维图象采样和相应的深度信息来判别两物体之间的相交情况。图形硬件有时也称为图形处理单元 (Graphics Process Unit, GPU)。如不特别指明，本文中二者完全对等。

这类算法优势在于能有效利用图形硬件加速技术来减轻 CPU 的计算负荷，从而达到提高算法效率的目的。但是基于图象空间的碰撞检测算法由于其检测结果的不精确性和依赖硬件支持而一直发展较慢。近些年，随着图形硬件计算性能的迅速增长，基于图象空间的碰撞检测算法进入了一个新的快速发展阶段。

Shinya 等[Shinya 1991]和 Rossignac 等[Rossignac 1992] 于 1991 年前后开创性地提出了图形硬件辅助碰撞检测的方法。随后，Myszkowski 等[Myszkow 1995]利用模板缓存 (stencil buffer) 检测每帧的变化状况，提出了一种有效的改进算法。Baciu 等[Baciu 1997][Baciu 1999]进一步利用深度缓存和模板缓存的组合功能，提高图象空间碰撞检测算法的效率，并使算法可在常规图形工作站甚至是普通个人台式电脑上运行。Hoff 等[Hoff 2001]和 Kim 等[Kim 2002]将图象空间碰撞检测算法和物体空间碰撞检测算法结合起来，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) 间进行合理调配来保证算法的整体效率。这方面的其他研究工作还包括[Lombardo 1999][Vassilev 2001][Heidelb 2003][Govindar 2003]。

做为综述，下面给出所有碰撞检测算法的分类图，如图 1-4。

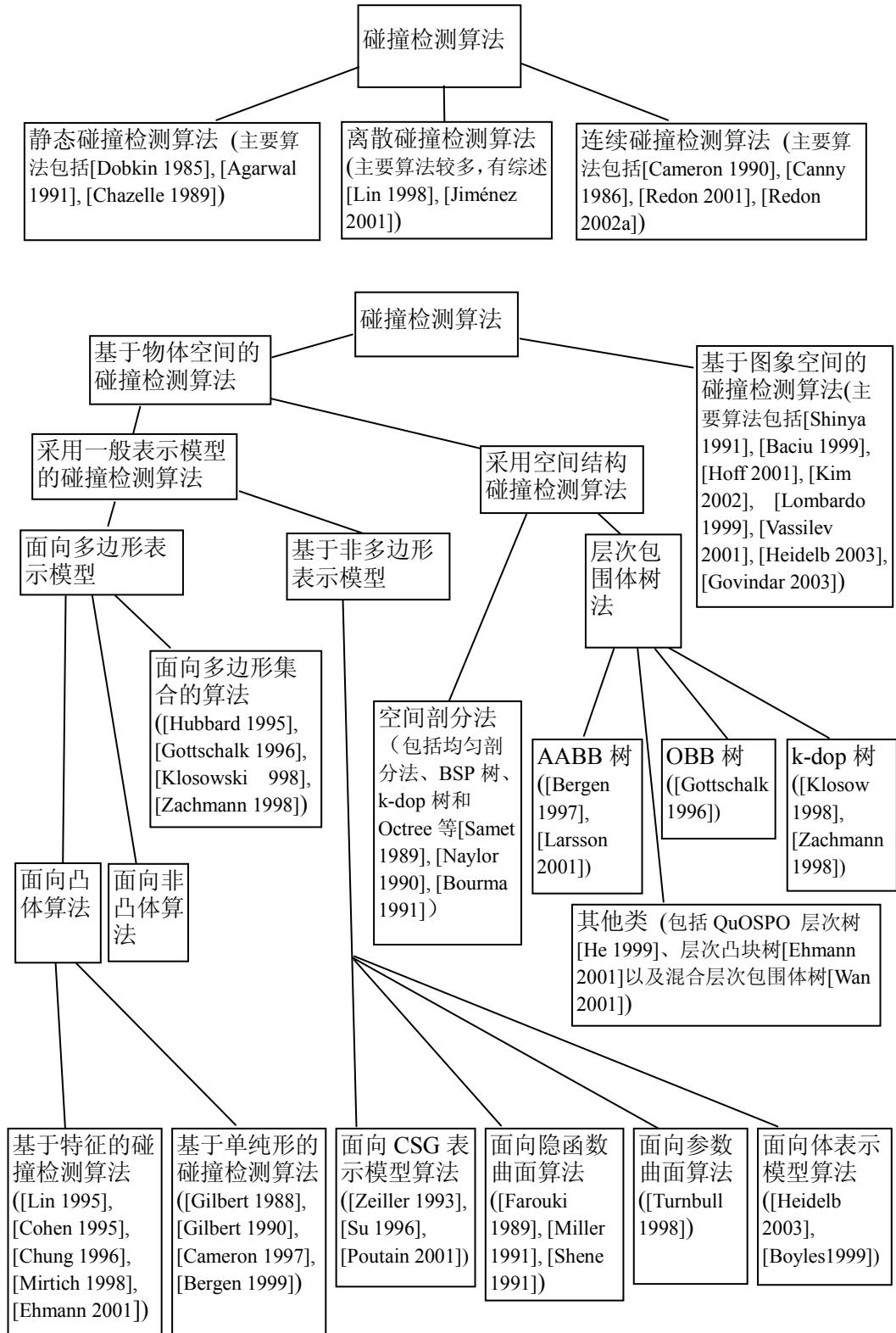


图 1-4 碰撞检测算法的分类总体图

1.2 碰撞检测算法的一般框架

总体而言，碰撞检测算法在处理包含大量物体的复杂场景时，或者首先将多数明显不相交的物体对进行快速排除，然后再对可能相交的物体对进行进一步检测[Cohen 1995]，或者采用场景空间剖分（如场景八叉树）来快速确定可能存在物体相交的区域，然后在这些潜在的相交区域中进行下一步操作[Zachmann 2001]。我们把这个过程统称为碰撞检测算法的初步检测阶段。而对于初步检测阶段的后继阶段，我们称之为碰撞检测算法的详细检测阶段。在详细检测阶段中，基于层次包围体树的碰撞检测算法首先会同时遍历物体对的层次树，递归检测层次树节点之间是否相交，直到层次树叶子节点，进而精确检测叶子节点中所包围的物体多边形面片或基本体素之间是否相交。而基于空间剖分的碰撞检测算法在详细检测阶段则逐步地对潜在相交区域进行细分，并检测细分后的子区域内是否有物体相交，直到分子区域中发现有不同物体的基本体素或多边形面片之间发生精确相交。

分析这些算法在详细检测阶段的步骤，可将其划分为两个层次，一是逐步求精层；二为精确求交层。在逐步求精层中算法进行层次树的遍历或者逐步细分潜在的相交区域。而精确求交层中算法主要处理多边形面片或基本体素之间的精确相交检测。由此可归纳出一个统一的碰撞检测算法框架，如图 1-5 所示。

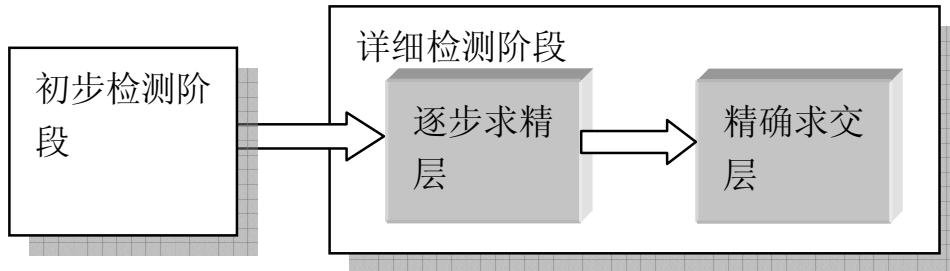


图 1-5 碰撞检测算法的一般框架

1.2.1 初步检测阶段

当动态场景中物体的个数超过两个，碰撞检测遇到的最明显的问题就是需要对所有 N 个物体进行两两求交检测，其时间复杂度达到 $O(N^2)$ 。这个问题也被称为“完全物体对检测问题”。很明显，这是任何碰撞检测算法在处理多物体

的场景时都会遇到的严重影响算法效率的问题。因此，当场景中物体个数较多时，非常有必要利用一些优化策略或方法来快速排除明显不发生碰撞的物体，找出潜在的相交区域或潜在的相交物体对。初步检测阶段所采用的技术主要有两种：一种是空间剖分法[Samet 1989][Hubbard 1995a][Hubbard 1995b][Zachmann 2001]；另一种是基于插入排序的“掠扫和裁剪”法(Sweep and Prune)[Cohen 1995]。

空间剖分法将场景均匀剖分成一个个小方块区间，检查这些小方块内是否有物体存在，否则将不包括物体的区间剔除，从而快速判断出潜在的相交区域。空间剖分方式如图 1-6 所示。

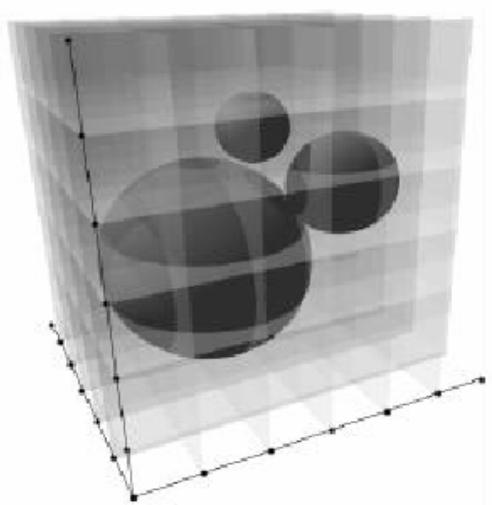


图 1-6 空间均匀剖分示例

另一种初步检测的方法是 Cohen 等[Cohen 1995]提出的“掠扫和裁剪”法。掠扫和裁剪法将场景中所有物体的 AABB 包围盒分别投影到 x, y, z 三个坐标轴上，并对每个物体在各坐标轴投影区间的边界值进行排序。两个物体包围盒在所有坐标轴的投影区间均有重叠时表明这两物体的包围盒相交。图 1-7 给出了物体包围盒在一个轴向的投影情况。由于时间连贯性，场景中物体发生移动时每帧之间物体的相对位置不会发生明显的改变，它们的包围盒在各坐标轴的投影位置也就不会有明显的变化。插入排序法在处理基本有序的队列排序时非常快速有效，因此算法采用插入排序法来有效保持各轴上的区间边界表有序，同时检查投影位置的变化情况。当发现物体 AABB 包围盒有重叠时才触发算法详细检测阶段的运行。

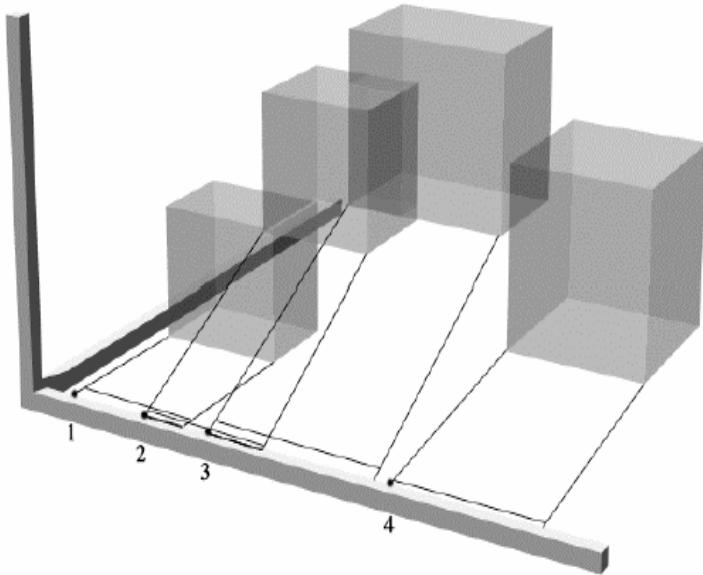


图 1-7 掠扫与裁剪方法中物体包围盒在一个轴向的投影情况

“掠扫和裁剪”方法的时间复杂度为 $O(N)$ ，因此用其处理“完全物体对检测问题”非常有效，计算负荷也很小。它虽然不能处理固定步长所引起的刺穿问题，但对于物体个数一定的场景可以确保在常数时间内完成初步碰撞检测。这就使得它在处理有大量运动物体且要求稳定帧率的动态场景中的碰撞检测时具有一定的优势。

1.2.2 详细检测阶段

碰撞检测算法经过初步检测阶段确定了潜在的相交区域或潜在的相交物体对集合之后，转入详细检测阶段。详细检测阶段根据已经确定的潜在的相交区域或潜在的相交物体对集合做进一步的相交检测。这个过程又分为两个层次，一个是逐步求精层；一个是精确求交层。

1.2.2.1 逐步求精层

一般算法框架中详细检测阶段的逐步求精层通常采用两种典型的加速技术：空间层次剖分技术和层次包围体树技术。

空间剖分技术与初步检测阶段的空间剖分技术相类似，但要把潜在的相交区域子空间继续剖分下去，直到找到相交物体的多边形面片。

层次包围体结构树技术是指算法利用预先建构好的物体层次包围体树，通过同时遍历物体对的层次包围体树，递归检测它们层次包围体树上的各层节点

包围体之间的相交情况，直到各个层次树的叶子节点，最终获取物体对的相交检测结果的技术。不同层次树叶子节点所包含的多边形之间的相交检测则由精确求交层的相交检测过程来完成。作为示例，图 1-8 给出了一个遍历层次包围体二叉树的递归算法伪代码。构建层次包围体树的好处就是物体包围体之间的相交测试极为简单，能够排除不交情况。层次包围体树在每一层中不断逼近物体，这本质上也是一种用层次细节（Level of Detail, LOD）表示物体的方法，但又不同于在快速绘制复杂物体或山脉地形表面时采用多分辨率方法中使用的多边形层次细节。在实时绘制中使用层次细节技术的目的是为了在加快绘制速度的前提下，尽可能地保证绘制结果与初始模型近似。而在碰撞检测的逐步求精层采用 LOD 技术总是尽量保守地逼近原物体模型，选择包围体的首要准则不是如何更逼近原模型，而是保证包围体之间的相交检测速度。

```

输入：a, b 两物体的层次包围体二叉树 BVTa, BVTb。输出：布尔值。true 为相
交，false 为不相交。
bool Detect_recursive(BVTa, BVTb)
{
    if (检测到BVTa与BVTb两个包围体之间不相交)
    {
        返回两物体不发生碰撞的结果;
    }

    if (BVTa, BVTb均为叶子节点)
    {
        精确检测BVTa, BVTb所包围的多边形面之间是否相交;
        返回精确求交检测的结果;
    } else if (BVTa 为叶子节点, BVTb为非叶子节点){
        Detect_recursive(BVTa, BVTb 的左子节点);
        Detect_recursive(BVTa, BVTb 的右子节点);
    } else if (BVTa 为非叶子节点, BVTb为叶子节点){
        Detect_recursive( BVTa 的左子节点, BVTb );
        Detect_recursive( BVTa 的右子节点, BVTb );
    } else{ //BVTa, BVTb均为非叶子节点
        Detect_recursive(BVTa, BVTb 的左子节点);
        Detect_recursive(BVTa, BVTb 的右子节点);
        Detect_recursive( BVTa 的左子节点, BVTb );
        Detect_recursive( BVTa 的右子节点, BVTb );
    }
}

```

图 1-8 层次包围体二叉树的递归遍历算法

1.2.2.2 精确求交层

如前所述，任何碰撞检测算法都要依赖于物体的表示模型。详细检测阶段的精确求交层便极大地依赖于物体所采用的表示方法。在精确求交层中，碰撞检测算法主要处理多边形面片或基本体素之间的精确相交检测。

基于多边形表示的碰撞检测算法在精确求交层中需要进行多边形与多边形之间的相交检测。由于多边形均可转化为三角形，并且三角形之间的相交检测更简单，因而有很多研究工作集中于三角形之间的快速相交检测[Gottschalk 1996][Möller 1997b]。

精确求交并不一定总是通过三角形的相交检测来实现。对于凸体之间的碰撞检测可以通过计算两者之间的距离或刺穿深度来实现。例如 Lin 和 Canny[Lin 1991][Lin 1993]所提出的基于特征的碰撞检测算法就是通过找出最邻近特征的距离来判断两物体是否精确相交。类似算法还有 Gilbert 等[Gilbert 1988]提出的基于单纯形的碰撞检测算法。该算法也是通过计算物体之间的距离来判断物体是否相交。

1.3 主要算法介绍

下面介绍碰撞检测领域的代表性研究工作和最新研究成果，具体分为以下四个方面：(1) 面向凸体的碰撞检测算法；(2) 基于一般表示的碰撞检测算法；(3) 基于层次包围体树的碰撞检测算法；(4) 基于图象空间的碰撞检测算法。

1.3.1 面向凸体的碰撞检测算法

面向凸体的碰撞检测算法大体上又可分为两类：一类是基于特征的碰撞检测算法；另一类是基于单纯形的碰撞检测算法。

1.3.1.1 基于特征的碰撞检测算法

顶点、边和面称为多面体的特征。基于特征的碰撞检测算法主要通过判别两个多面体的顶点、边和面之间的相互关系进行它们之间的相交检测。所有基于特征的方法基本上都源自于 Lin-Canny 的工作[Lin 1991][Lin 1993]。

Lin-Canny 算法通过计算两个物体间最邻近特征的距离来确定它们是否相交。该算法利用了连贯性来加快相交检测的速度。具体地，因为在连续的两帧之间最邻近特征一般不会明显变化，因此可通过将当前的最邻近特征保存到特征缓存中来加快下一帧的相交检测速度。当最邻近特征发生了变化后，算法依据特征的 Voronoi 区域先查找与上一帧中保留特征的相邻特征，以此提高查找效率，从而提高相交检测的效率。当碰撞检测的时间步相对物体移动速度较小时，算法可在预定的常数时间内进行特征跟踪。这里假定碰撞检测算法每个循环计算是在场景运动变化后进行的。

I-Collide[Cohen 1995]是以 Lin-Canny 算法为基础，结合了时间连贯性的一个精确的碰撞检测共享库，可用于由凸多面体构成的模型，并能够处理多个运动物体组成的场景。

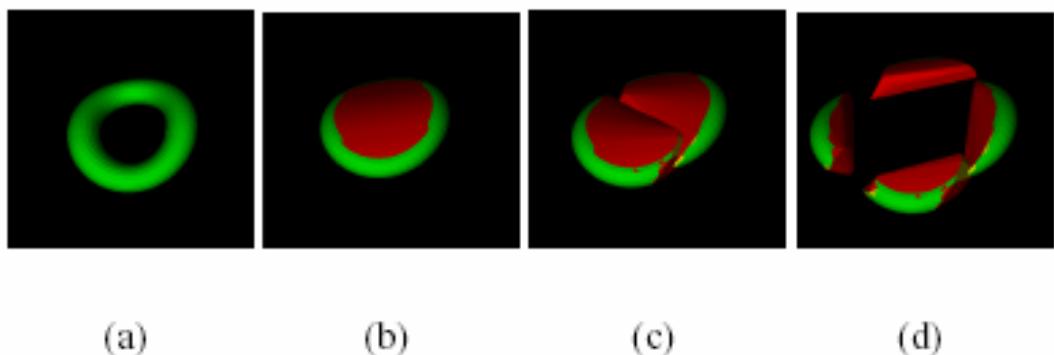
然而，Lin-Canny 算法并不能够处理刺穿多面体的情况。面对刺穿情况，算法会进入死循环。一个简单的解决办法是在算法达到最大循环数后强制中止，返回物体发生碰撞。但这种解决方法太慢，且无法判定物体是否真正相互刺穿。事实上，只要物体不是移动太慢或碰撞检测的时间间隔不是十分短，物体相互刺穿的现象非常常见。算法无法检测刺穿，这对实时应用环境如游戏和虚拟现实是不可想象的。当相互刺穿发生又要求更精确的接触时间信息时，就要回溯

到碰撞发生的精确瞬间，这种过程也是非常缓慢和繁琐的。[Ponamgi 1997] 等引入了凸多面体的伪内部 Voronoi 区域来克服这个问题。但同时引入了需要解决的其它问题，包括如何处理平行特征面等特殊情况和如何配置容错阈值以将算法调整到理想性能等问题。

另一个具有代表性的基于特征的碰撞检测算法是 V-Clip (Voronoi-Clip) [Mirtich 1998]。Mirtich 宣称解决了 Lin-Canny 算法的局限性。V-Clip 算法能处理刺穿情况，不需要用容错阈值来调整，而且不会有死循环的现象。同时由于特例情况少而实现起来更加简单。它是目前所有算法中，处理凸体之间的碰撞检测最快速有效的算法之一。VClip 既可以处理凸体，也可处理非凸体，甚至还可处理不连通的物体，在物体发生刺穿时还能返回刺穿深度。它的算法效率相对 I-Collide 和 Enhanced GJK 有明显的改善，而且比较强壮。

之后，北卡罗莱纳大学 Ehmann 等[Ehmann 2000]开发的 SWIFT (Speedy Walking via Improved Feature Testing) 算法达到了更优的算法效率。该算法结合了基于 Voronoi 区域的特征跟踪法和多层次细节表示两种技术，可适用于具有不同连贯性程度的场景，并能够提高碰撞检测的计算速度。它比 I-Collide、VClip 和 Enhanced GJK 算法速度更快，也更强壮，但在少数情况下还是会陷入死循环。比较可惜的是，它一般只处理凸体或由凸块组成的物体。算法能够检测出两物体的相交情况并计算出最小距离，确定出相交的物体对，但并不能求出刺穿深度。

针对 SWIFT 算法仅能处理凸体的缺陷，Ehmann 等[Ehmann 2001]通过对 SWIFT 算法进行扩展提出了 SWIFT++算法。SWIFT++算法在预处理阶段将场景中所有物体进行表面凸分解，并重新组织凸分解产生的结果凸片，建构出各个物体的凸块层次树（图 1-9）。与 RAPID, PQP 和 QuickCD 等算法相比，SWIFT++的性能更加可靠，不受场景复杂度的影响。算法能处理任意形状物体间的碰撞检测，它除了可返回两物体对的相交检测结果，还可计算出最小距离和确定相交部分的信息（如点，边，面等），但仍不能求出刺穿深度。



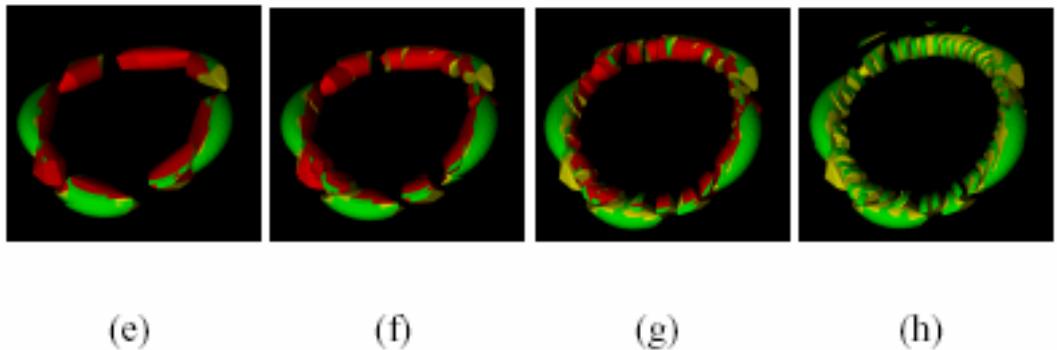


图 1-9 表面凸分解和层次树的建构

1.3.1.2 面向单纯形的碰撞检测算法

这类算法是由 Gilbert、Johnson 和 Keerthi [Gilbert 1988]率先提出的，称为 GJK 算法。面向单纯形的碰撞算法是与基于特征的算法相对应的一类算法。

GJK 算法以计算一对凸体之间的距离为基础。假定两凸体 A 和 B ，用 $d(A, B)$ 来表示 A 与 B 之间的距离，则距离可以用下面公式(1-1)表示：

$$d(A, B) = \min \{ \|x - y\| : x \in A, y \in B \} \quad (1-1)$$

算法将返回两物体间最邻近的两个点 a, b ，它们满足公式(1-2)：

$$\|a - b\| = d(A, B) \quad a \in A \text{ 且 } b \in B \quad (1-2)$$

A, B 之间距离用 Minkowski 和 $A - B$ 的形式可表示为公式(1-3)：

$$d(A, B) = \|v(A - B)\| \quad (1-3)$$

其中 $v(C)$ 为 C 中的点到原点的距离，也即公式(1-4)：

$$v(C) \in C \quad \text{且} \quad \|v(C)\| = \min \{ \|x\| : x \in C \} \quad (1-4)$$

从而就有 $a - b = v(A - B)$ 。

这样算法就将 A, B 两凸体间的相交检测转化为在单纯形($A - B$)上找出距离原点最近的点。单纯形是三角形在任意维度上的概括性名称，多数情况下都可把多面体看作一个点集的凸包。相交检测所有操作都在这些点的子集所定义的单纯形上进行。

GJK 算法的主要优点在于除了可检测出两物体是否相交，还能返回刺穿深度。[Rabbitz 1994]利用连贯性对 GJK 进行了改进。Cameron 等[Cameron 1997]又进一步改进了该算法，提出了 GJK 增强算法（Enhanced GJK）。GJK 增强算法在 GJK 的基础上引入了爬山思想（Hill Climbing），提高了算法效率。该算法

性能与 I-Collide 和 VClip 算法性能相近，能够在常数时间内计算出两凸体对之间的距离。该算法在时间复杂度上基本和 Lin-Canny 相同，但同时它又克服了 Lin-Canny 算法主要的弱点。Mirtich 称 V-Clip 算法比 Enhanced GJK 算法需要更少的浮点运算，效率更高，但同时他也承认 GJK 类的算法能更好地计算刺穿深度。

Bergen 等[Bergen 1999]开发的 SOLID 算法(Software Library for Interference Detection)，也是一个基于 GJK 的碰撞检测算法。它除了采用 GJK 的基本思想，还结合了基于 AABB 的掠扫和裁剪的增量剔除技术，并通过缓存上一帧中物体对的分离轴，利用帧与帧的连贯性来判别潜在的相交物体对，以加快算法效率。

所有面向凸体的算法都宣称精确求交处理非凸的多面体也是简单的，认为非凸多面体可由凸子块组成的层次结构表示。这些算法先对凸子块的包围体进行相交检测，如果发现相交再进一步检测包围体内的凸子块是否相交，因此称为“开包裹法”。这些算法本身虽然对凸体特别有效，但当物体的非凸层次增加时，它们的检测速度会迅速下降。因此这些算法更适用于对包含少量凸体的场景进行实时碰撞检测。对于其他情况，基于层次表示的碰撞检测算法将更加实用。

1.3.2 基于一般表示的碰撞检测算法

碰撞检测算法中有不少是专门面向某种具体表示模型而设计的，包括面向 CSG 表示模型的碰撞检测算法、面向参数曲面的碰撞检测算法和面向体表示模型的碰撞检测算法等。

1.3.2.1 面向 CSG 表示模型的碰撞检测算法

Zeiller [Zeiller 1993]提出了一种面向 CSG 表示模型的碰撞检测算法。他将算法分为三个部分。第一部分求出 CSG 树的每个节点的包围体用于快速确定可能的相交部分；第二部分针对所有 CSG 树表示的物体创建类似八叉树的层次结构，采用这种结构找到同时包含两物体体素的子空间；在最后一部分，检测子空间中基本体素之间的相交关系。

Su 等[Su 1996]在算法预处理阶段首先将 CSG 表示模型转化为边界表示模型(Brep)，然后混合两种表示，把每个 CSG 的基本体素与对应 Brep 的面片关联起来。此外，还对 CSG 树中的非叶子节点建构相应包围体，并在相交检测时采用自适应的包围体选择策略以快速确定潜在相交区域，从而提高算法效率。

该算法结合了包围体技术的快速性和基于多边形表示相交检测的精确性来提高碰撞检测算法效率。与 Su 算法相类似还有 Poutrain 等[Poutrain 2001]提出的一种混合边界表示和 CSG 表示的碰撞检测算法。算法利用了包括 CSG 在内的多种表示方法，将包围体、层次细分和空间剖分等技术融合起来实现实时碰撞检测。

1.3.2.2 面向参数曲面的碰撞检测算法

Turnbull 等[Turnbull 1998]提出了一种面向 NURBS 表示凸体的碰撞检测算法，该算法借助“支持映射”(support mapping) 来求出两凸体之间的距离。“支持映射”通过给定的支持函数(support function) 和方向，获取两个凸体之间的最小距离，同时返回两物体距离最近的两个顶点，如图 1-10 所示。利用这种思想 Turnbull 提高了 NURBS 曲面表示物体间的碰撞检测速度。[Farouki 1989][Miller 1991][Shene 1991]等也提出了其他面向参数曲面的碰撞检测算法。

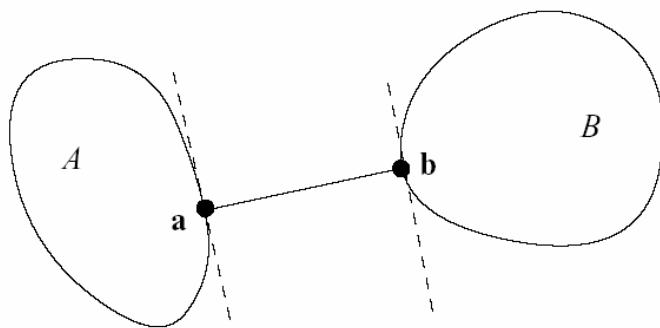


图 1-10 支持映射示例

1.3.2.3 面向体表示模型的碰撞检测算法

面向体表示模型的碰撞检测算法通常用于虚拟手术，因为体表示模型可以表示物体内部的相关数据。体表示的简单性也使其可用于对碰撞检测算法速度要求极高的应用，如面向触觉反馈的碰撞检测计算。触觉反馈中由于人对触觉的敏感度，系统对碰撞检测的计算要求非常高，通常要求刷新频率达到 1000Hz。对于如此高的计算速度要求，除结合具体场景的特点来加速算法外，往往会考虑以牺牲精度为代价来提高碰撞检测的速度。

McNeely 等[McNeely 1999]在 1999 年针对 Boeing 公司飞机设计时所遇到的问题，提出了一种相当快速的面向触觉反馈的实时碰撞检测算法 Voxelmap

PointShell。该算法将整个场景先均匀分割为小的立方体，称为体素（voxel），然后把场景中静止的部分组织为一个类似八叉树的层次结构树，同时从运动物体所占用的体素中获取点壳（PointShell）来表示运动物体(图 1-11)。如此，检测运动物体是否与静止物体发生碰撞就只需判断点壳上的点是否位于包含静止物体的体素之内。该算法的特点是能处理任意形状的物体，碰撞检测速度非常快速且强壮，但遗憾的是它不能有效处理含有大量运动物体的动态场景，且碰撞检测的精度也比较低。图 1-12 显示了一个 Voxelmap PointShell 算法的测试场景。



(a)茶壶的三维模型 (b)茶壶的体素模型 Voxelmap (c)茶壶的点壳模型 PointShell

图 1-11 茶壶的三维模型及其体素模型和点壳模型

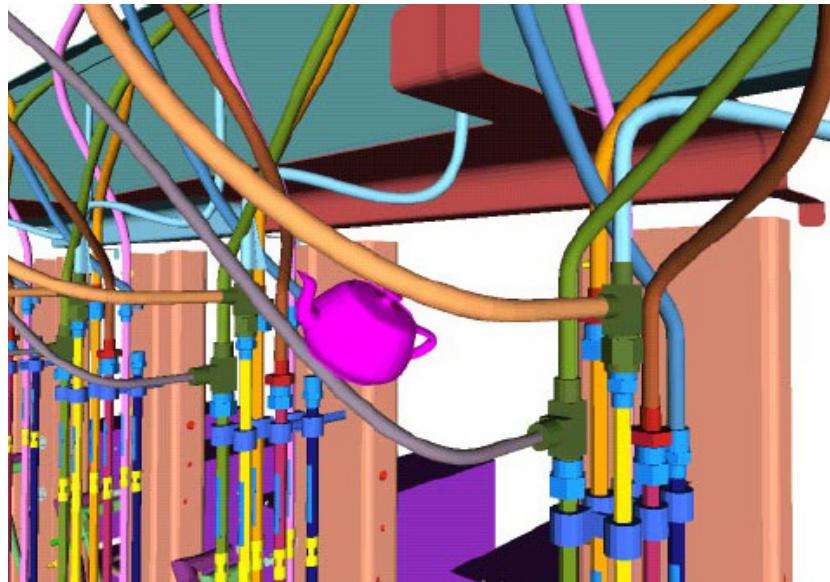


图 1-12 Voxelmap Pointshell 算法所用的一个测试场景

面向特定表示模型的碰撞检测算法一般有其特殊的应用领域，例如，面向 CSG 表示模型和面向参数曲面表示的碰撞检测多用于 CAD 应用中，它们检测速度较慢，但一般比较精确。而面向体表示的碰撞检测算法在虚拟手术中较常用，也有用于触觉反馈中的。这类算法的优势在于可以对物体的内部进行处理，并能够达到较快的检测速度，但由于体表示的不精确性使其很难保证碰撞检测

结果的精确性。

1.3.3 基于层次包围体树的碰撞检测算法

在算法分类中，曾讨论过在碰撞检测算法中所使用的包围体类型。对应于每一类的包围体都有一个代表性的碰撞检测算法。下面一一讨论，并对它们的优缺点进行比较。

1.3.3.1 基于AABB层次包围盒树的碰撞检测算法

AABB 层次包围盒树，是利用 AABB（Axis Aligned Bounding Box）构建的层次结构二叉树 [Zachmann 1997]。AABB 的建构比较简单，相互之间的求交也很快捷，但由于包围物体不够紧密，有时会增加许多不必要的检测，反而影响算法效率。

一般的 AABB 树由于包围较松散，会产生较多的节点，导致层次二叉树的节点过多的冗余，从而影响 AABB 树的碰撞检测效率。为此，Bergen [Bergen 1997] 提出了一种有效的改进算法。该算法采用分离轴定理（Separate Axis Theorem）加快 AABB 包围盒之间的相交检测，同时又利用 AABB 局部坐标轴不发生变化的特性加速 AABB 树之间的碰撞检测。他的算法与 Gottschalk 等 [Gottschalk 1996] 提出的采用 OBB 树的碰撞检测算法相比，计算性能上相差不大。由于 AABB 树原本就具有建构简单快速，内存开销少的特点，能较好地适应可变形物体实时更新层次树的需要，因此 Bergen 又把他的算法用于进行可变形物体之间的相交检测。

Larsson 等[Larsson 2001]针对可变形物体的碰撞检测问题提出了一种有效建构、更新层次包围盒树的方法。他通过多种启发式搜索策略构建结构良好的 AABB 层次树，并在碰撞检测阶段结合了自顶向下和自底向上的两种层次树更新策略来保证层次包围体树的快速更新，有效加快了变形物体之间碰撞检测的速度。

1.3.3.2 基于层次包围球树的碰撞检测算法

Palmer 等[Palmer 1995]提出的一种快速碰撞检测算法分为三个阶段：首先通过全局包围体快速确定处于同一局部区域中的物体；然后，依据一个基于八叉树建构的层次包围球结构来进一步判断可能的相交区域；最后，检测层次包围

球树叶子节点中不同物体面片的相交情况。他提出的层次包围球树算法简单，但处理大规模场景较为困难。

Hubbard [Hubbard 1995a][Hubbard 1995b][Palmer 1995] 利用球体建构物体的层次包围体树，可以比较快捷地进行节点与节点之间的检测（如图 1-13）。基于包围球体与 AABB 树存在同样的问题，就是包围物体不够紧密，建构物体层次树时会产生较多的节点，导致大量冗余的包围体之间的求交计算。

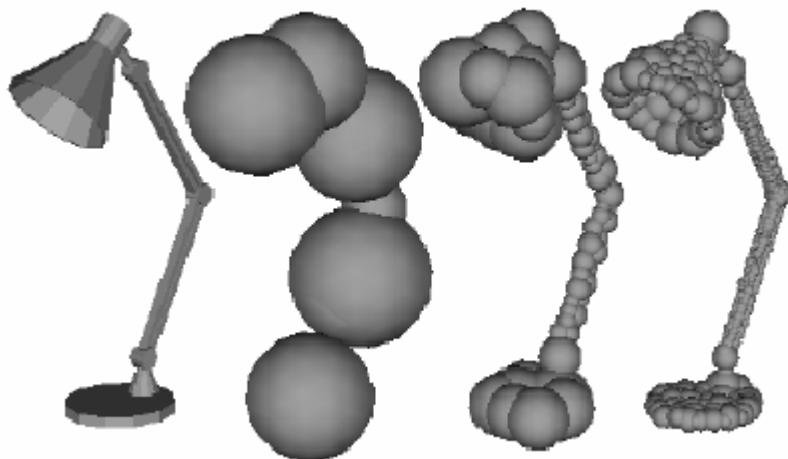


图 1-13 层次包围球树的建构

Hubbard 还提出了一种自适应时间步长的技术来解决离散碰撞检测算法可能出现的遗漏和错误检测的情况。其方法的关键是在初步检测阶段，采用了一种称为时空边界的四维结构，其中第四维是指时间。该结构可保守地估计出物体在后面可能的运动位置。当所有边界有重叠后，算法就会触发详细检测阶段进一步进行检测。此外，他还通过在详细检测阶段引入自适应精度，提出所谓可中断的碰撞检测算法 (interruptible collision detection algorithm)。为了保证碰撞检测的计算速度，该方法允许在给定的时间内逐步提高碰撞检测的准确度。包围球之间的碰撞检测按层次树的层次逐步增加层次细节，同时算法在每个循环中遇到中断时就减少所有包围球树参加碰撞检测的层次个数，以此确保在指定的时间内快速给出可能不精确的结果。

O'Sullivan 等[O'Sullivan 1999][Dingliana 2000][Dingliana 2001]在可中断碰撞检测算法方面进行了更深入的研究工作。该算法通过使用物理响应的优化方法得到最近似的相交信息，合理地降低碰撞检测精度来满足系统响应的时间要求。

1.3.3.3 基于 OBB 层次包围盒树的碰撞检测算法

Gottschalk 等[Gottschalk 1996]于 1996 年提出了一种基于 OBB (Oriented Bounding Box) 层次包围盒树的碰撞检测算法，称为 RAPID 算法。他们采用 OBB 层次树来快速剔除明显不交的物体。OBB 的建构方式如图 1-14 所示。很明显 OBB 包围盒比 AABB 包围盒和包围球更加紧密地逼近物体，能比较显著地减少包围体的个数，从而避免了大量包围体之间的相交检测。但 OBB 之间的相交检测比 AABB 或包围球体之间的相交检测更费时。为此，Gottschalk 等提出了一种利用分离轴定理判断 OBB 之间相交情况的方法，可以较显著地提高 OBB 之间的相交检测速度。

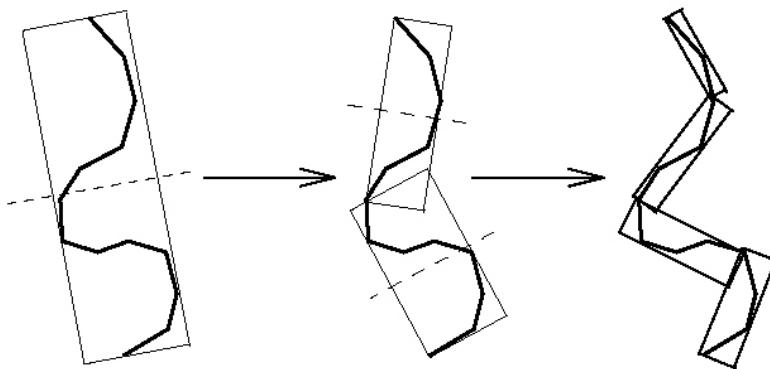


图 1-14 OBB 树的建构

算法首先确定了两个 OBB 包围盒的 15 个分离轴，这 15 个分离轴包括两个 OBB 包围盒的 6 个坐标轴向以及三个轴向与另三个轴向相互叉乘得到的 9 个向量。然后将这两个 OBB 分别向这些分离轴上投影，再依次检查它们在各轴上的投影区间是否重叠，以此判断两个 OBB 是否相交。

不同 OBB 树的叶子节点内包围的三角形之间的相交检测也利用分离轴定理来实现。算法首先确定两三角形的 17 个分离轴，它们包括两三角形的两个法向量、每个三角形的三条边与另一三角形的三条边两两叉乘所得到的 9 个向量以及每个三角形各条边与另一个三角形的法向量叉乘得到的 6 个向量。依次检查这两个三角形在这 17 个分离轴上的投影区间是否有重叠来获取它们的相交检测结果。

RAPID 的缺陷在于无法用来判断两三角面片之间的距离，只能得到二者的相交结果。此外 RAPID 也没有利用物体运动的连贯性，其算法需要有预处理时间，一般只适用于处理两个物体之间的碰撞检测。

1.3.3.4 基于 k-dop 层次包围体树的碰撞检测算法

Klosowski 等[Klosowski 1998]利用“离散有向多面体”(discreted orientaton polytope 或 k-dop)建构的层次包围体树来进行碰撞检测。Zachmann [Zachmann 1998]也提出了类似的方法。QuickCD 是基于该算法的共享软件包。

“k-dop”包围体是指由 $k/2$ 对平行平面包围而成的凸多面体, k 为法向量的个数(图 1-15)。可以看出 k-dop 包围体能比其他包围体更紧密地包围原物体, 创建的层次树也就有更少的节点, 求交检测时就会减少更多的冗余计算。但 k-dop 包围体之间的相交检测会更复杂一些。Klosowski 等通过判别 $k/2$ 个法向量方向上是否有重叠的情况来判定两个 k-dop 包围体是否相交。所以, 法向量的个数越多, k-dop 包围体包围物体越紧密, 但相互之间的求交计算就更复杂, 因此需要找到恰当个数的法向量以保证最佳的碰撞检测速度。

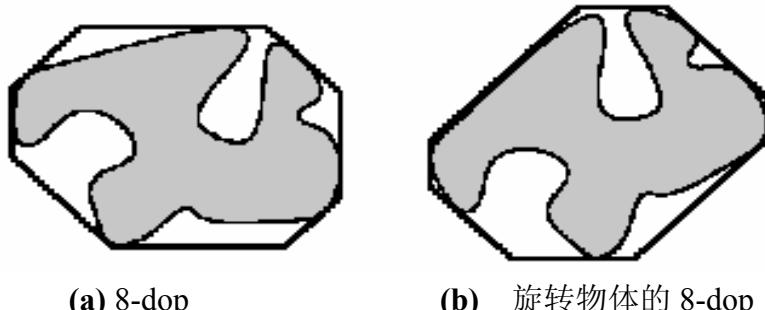


图 1-15 一个物体 k-dop 包围体的例子

1.3.3.5 基于扫成球层次包围体树的碰撞检测算法

PQP(Proximity Queries Package) 是 Larsen 等[Larsen 1999]所提出算法的实现。它的主要思想源自于 RAPID, 但又与之不同。PQP 所采用的包围体为扫成球包围体(Swept Sphere Volume), 并生成 SSV 层次树(图 1-16)。而且 PQP 不但可以返回相交检测结果, 还能进行最近距离和容错值的查询。也就是说, 算法并不局限于处理碰撞检测问题, 它能处理包括碰撞检测在内的邻近查询。PQP 所能处理的物体对象也比较广泛, 一般只要是三角形网格的模型就能处理, 对于一些特殊情况如裂缝, 空洞等无需进行特别的处理。

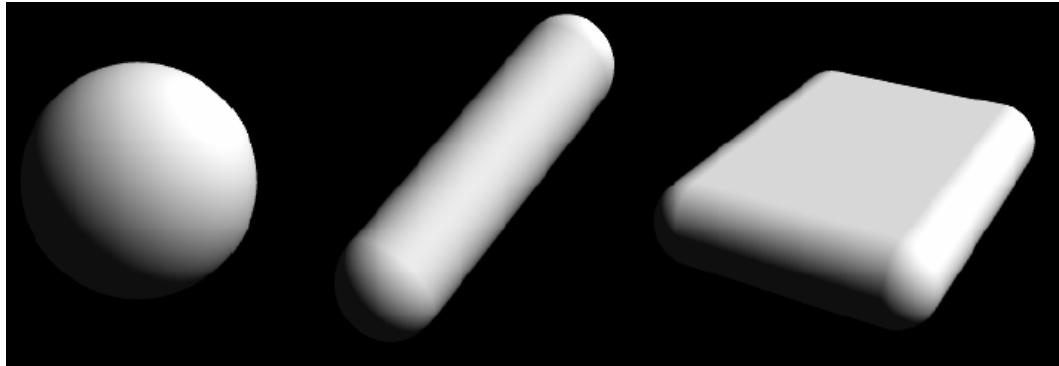


图 1-16 点, 线, 面的扫成球包围体(SSV)

所有基于层次包围体树的碰撞检测算法都通过递归遍历层次树来检测物体之间的碰撞。一般地, 其算法性能受两个方面影响: 一是包围体包围物体的紧密程度; 二是包围体之间的相交检测速度。包围体包围物体的紧密度影响层次树的节点个数, 节点个数越少在遍历检测中包围体检测次数也就越少。OBB 和 k-dop 能相对更紧密地包围物体, 但建构它们的代价太大, 对有变形物体的场景往往无法实时更新层次树。AABB 和包围球包围物体不够紧密, 但它们层次树更新快, 可用于进行变形物体的碰撞检测。在包围体相交检测的速度方面 AABB 和包围球具有明显优势, OBB 和 k-dop 则需要更多的时间。但总体上对于刚体而言, 基于 OBB 的碰撞检测算法最优。

1.3.4 基于图象空间的碰撞检测算法

Rossignac [Rossignac 1992]等利用深度缓存和模板缓存来辅助进行机械零件之间的相交检测。他们通过移动图形硬件的裁剪平面, 判断平面上的每个象素是否同时在两个实体之内来确定物体是否相交。

Shinya 和 Forgue[Shinya 1991]等提出在绘制凸体的同时, 保存视窗口中每个象素上物体的最大和最小深度序列, 并将它们按大小顺序排列, 然后检测物体在某一象素上的最大深度值是否与其最小深度值相邻来判别相交情况。图形硬件可以支持物体最大最小深度的计算。但该方法并不实用, 因为它要求大量的内存来保存深度序列, 而且从图形硬件中读取深度值本身就非常费时。

Myszkowski 等[Myszkow 1995]将深度缓存和模板缓存结合在一起进行相交检测。它们用模板缓存值来保存视窗口中每个象素上所代表的射线进入一物体前进入和离开其他物体的次数, 并读取模板缓存中的值来判断两物体是否相交。该算法仅能处理两个凸体之间碰撞检测问题。

Baciu 和 Wong[Baciu 1997][Baciu 1999]改善了 Myszkowski 等的方法，他们先用几何的方法确定两物体包围盒的相交区域，然后在该相交区域中利用图形硬件的加速绘制进行相交检测。他们分析了两个物体的各种相交情况，并将这些相交情况按深度值顺序位置进行了分类，同时用模板缓存值来表示这些分类，如图 1-17 所示。之后，算法通过检查模板缓存值来判别两物体之间是否发生碰撞。算法主要缺点是仅能处理凸多面体或由凸体组成的多面体。

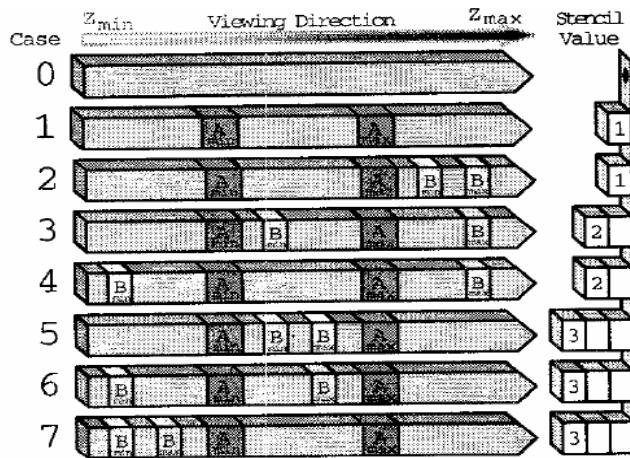


图 1-17 两物体位置与模板缓存值的对应图

Vassilev 等[Vassilev 2001]在人体与衣物之间的碰撞检测中也采用了基于图象的方法，他们利用深度缓存和颜色缓存来判别衣服和虚拟人是否发生碰撞。这种方法利用图形硬件加速了碰撞检测速度，但对虚拟人的姿势有较大限制，且碰撞检测的精度不高。

Hoff 等[Hoff 2001]提出的 PIVOT 算法(Proximity Information from Voronoi Techniques)结合了 Voronoi 区域的几何特性，利用图形硬件来处理二维模型之间的邻近性查询。算法首先采用几何的方法快速确定两个平面模型的包围盒的相交区域(图 1-18)，然后用图形硬件的帧图象缓存快速找出更准确的相交区域，并生成 Voronoi 图用于进一步计算相交区域的距离场，最终计算出分离距离或刺穿距离以及接触点和法向量等邻近查询所要求的信息。该算法的缺陷在于只能处理二维模型之间的碰撞检测。他们[Hoff 2003]最近提出的改进算法已经拓广到三维物体上，可以用图形硬件来处理三维封闭网格表示的物体，包括非凸物体和可变形物体。该算法同样结合了物体空间的几何技术大致定位潜在的碰撞区域，利用多遍绘制技术以及快速距离域计算方法来加快底层的精确邻近查询。算法对潜在的相交区域进行三维均匀网格采样后，采用体表示该区域，并用图形硬件加速邻近查询或碰撞检测过程。该算法通过混合使用基于几何与基于图象的方法来平衡 CPU 和 GPU 的计算负载。

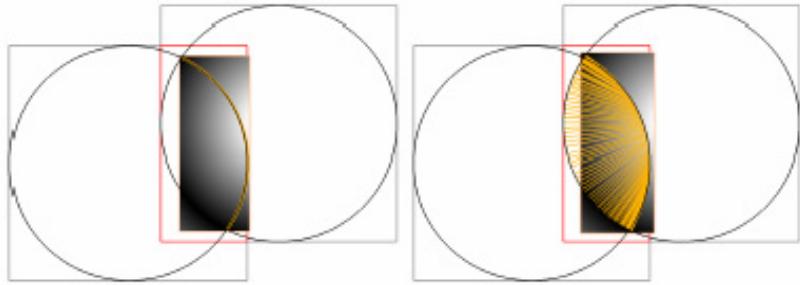


图 1-18 采用几何方法确定潜在相交区域

Kim 等[Kim 2002]更进一步把基于物体空间和基于图象空间的两种算法结合起来，在利用基于物体空间的方法快速查找出两物体对潜在相交区域后，再利用基于图象的方法快速求出两个物体的分离距离或者刺穿深度，从而达到较好碰撞检测效率。

Govindaraju 等[Govindar 2003]最近利用图形硬件快速剔除大规模的场景中明显不发生相交的物体，即进行初步检测阶段的物体剔除。然后利用几何的快速相交检测算法得到碰撞检测的结果。该算法与一般的基于图象的碰撞检测算法不同之处在于，使用图形硬件的方式不同。该算法在初步检测阶段利用图形硬件加速检测过程，而一般的基于图象的碰撞检测则在详细检测阶段的精确求交层利用图形硬件加速计算。

Heidelberger 等[Heidelb 2003]提出了一种面向体表示，能处理可变形物体的碰撞检测算法。算法首先将两物体的相交区域按层次深度分解为层次深度图 (layered depth image)，然后通过图形硬件绘制过程来判断两物体在层次深度图的每个象素上是否有相交区间存在，从而确定物体是否发生碰撞。

基于图象的碰撞检测算法的实现一般比较简单，而且它们可以有效利用图形硬件的高性能计算能力，缓解 CPU 的计算负荷，在整体上提高碰撞检测算法效率。随着图形硬件的发展，基于图象的碰撞检测算法还具有广阔的发展前景。但是基于图象的碰撞检测算法普遍存在以下三个缺陷：(1) 由于图形硬件绘制图象时本身固有的离散性，不可避免地会产生一定误差，从而无法保证检测结果的准确性；(2) 多数基于图象的碰撞检测算法仍只能处理凸体之间的碰撞检测；(3) 由于使用图形硬件辅助计算，基于图象的碰撞检测还需要考虑如何合理地平衡 CPU 和图形硬件的计算负荷。

1.4 目前存在的问题

尽管有关碰撞检测的研究成果已经比较丰富，但随着计算机图形学的不断发展，以及包括虚拟现实在内的新兴领域的涌现，实际应用对碰撞检测技术的要求越来越高。目前碰撞检测领域仍然存在着以下问题亟需解决。

(1) 处理大规模复杂场景的能力。目前多数基于物体空间的碰撞检测算法的效率与场景中物体复杂度成反比关系。图形硬件技术的发展，系统已经可以实时显示大规模场景，这些场景常常有成百上千万的面片，甚至包含数据量大到内存都无法容纳的物体。在这种场景中进行碰撞检测，对碰撞检测算法就提出了更高的要求。随着场景中多边形面片数的增多，多数算法的效率往往会迅速下降。这样就无法保证碰撞检测的实时性和稳定性。

(2) 处理非凸物体的能力。一些高效的碰撞检测算法主要局限于凸体间的碰撞检测，如基于特征类的碰撞检测算法和基于图象的碰撞检测算法都仅能快速检测凸体之间的碰撞。而在实际的虚拟场景中，大多数物体均是非凸物体。虽然有些改进算法提出用凸块来组织成非凸体，但这样做对算法性能会有较大影响。

(3) 基于图象的碰撞检测算法的精度保证。基于图象的碰撞检测算法因受图象分辨率的限制，往往难以保证碰撞检测的精度。该类算法虽然可以有效利用图形硬件来加速碰撞检测速度，减轻 CPU 的负荷，进而提高算法效率，但由于图象本身均是空间离散采样，其精度受图象分辨率的约束，从而影响碰撞检测算法的精度。如果以提高分辨率来保证精度，则又会影响图形绘制的速度，反而降低碰撞检测的效率。

(4) 面向特定应用领域的高效碰撞检测算法。一些特定应用领域中的碰撞检测问题，如针对衣物布料等可变形物体的碰撞检测和虚拟现实中的触觉计算等，往往有更特殊甚至更苛刻的要求，而目前多数碰撞检测算法还无法满足其要求。一方面，衣物布料等可变形物体在仿真中模型的结构不断发生变化，需要重新建构物体的层次包围体树，而目前多数碰撞检测算法都要求有较长的预处理时间来完成层次包围体树的重新建构，通常很难达到实时；另一方面，在虚拟现实的触觉计算中，由于人对触觉的敏感度，往往要求碰撞检测的计算达到每秒上千帧的速度。一般的碰撞检测算法对此是无法胜任的。

(5) 连续碰撞检测算法。目前多数算法是离散的碰撞检测算法，这类算法有两个共同的缺陷，一是存在刺穿现象，当离散检测步长过大时，两物体可能已发生了一定深度的刺穿才被检测到已发生碰撞，这就无法保证物体的运动真

实性；二是会遗漏发生碰撞的情况，对于较狭窄的物体，当运动物体在相邻离散时间点处于该狭窄物体两侧时，离散算法无法正确地检测出应有的碰撞。采用连续碰撞检测算法虽然可以解决这两个问题，但连续算法的计算开销太大，往往将成为实时系统的计算瓶颈。因此，如何有效结合离散碰撞检测算法和连续碰撞检测算法的优势成为目前碰撞检测领域中有待解决的一个问题。

1.5 研究目标和研究内容

针对碰撞检测技术目前存在的问题，本文围绕如何有效地提高碰撞检测算法的效率以及如何突破基于图象算法在适用性和检测精度上的局限性等问题展开了研究，并以解决上述问题作为主要的研究目标。具体研究内容包括以下三个方面：

- 利用表面凸分解和凸块层次树使基于图象的碰撞检测算法能够处理任意形状物体之间的碰撞检测，同时采用绘制加速技术提高算法效率；
- 通过可编程图形硬件提高基于图象的碰撞检测算法的速度与精确性。
- 利用并行计算来提高碰撞检测效率；

本文后面各章的组织方式是：

第二章介绍一种新的基于图象的快速碰撞检测算法，该算法将图形硬件的计算优势和简化的几何模型表示结合起来实现复杂物体间的实时碰撞检测；

第三章介绍一种基于流的快速精确的碰撞检测算法，该算法探索性地采用可编程图形硬件来解决复杂物体间的实时碰撞检测问题；

第四章介绍并行碰撞检测算法，该算法在建构动态复杂场景中物体的平衡层次包围体树的基础上，通过一种新的边包围体遍历策略有效地提高并行碰撞检测算法的效率；

最后，在第五章中对本文的工作进行总结并提出进一步的研究方向。

第二章 基于图象的快速碰撞检测算法

摘要

基于图象的碰撞检测算法能有效利用图形硬件绘制加速功能，减轻 CPU 计算负荷，达到提高碰撞检测效率的目的。本章提出了一种新的基于图象的快速碰撞检测算法，将图形硬件的计算优势和简化的几何模型表示如 OBB 层次树结合起来实现复杂物体间的实时碰撞检测。算法在继承一般基于图象的碰撞检测算法优点的同时，突破了它们的局限性，能够在保证效率的前提下处理任意形状多面体之间的碰撞检测问题。算法预先对物体表面进行自动凸分解，并将凸分解结果合理地组织成层次二叉树结构以适应基于图象的碰撞检测过程，同时采用三角形带压缩这一绘制加速技术和 OBB 包围盒技术来加快碰撞检测阶段的绘制速度，从而提高碰撞检测的效率。

关键词： 碰撞检测 基于图象 OBB 包围盒 绘制加速 凸分解

2.1 引言

近年来，随着集成电路设计技术和深亚微米制造技术的迅速发展，计算机图形硬件的处理能力不断得到突破，其计算性能已经超越 CPU。并且，由于图形硬件内在的并行结构，这种性能上的差异在可预见的将来势必进一步扩大。因此，除了将图形硬件用于加速绘制过程外，研究者们开始思考如何把图形硬件作为一个协处理器为 CPU 提供更多的辅助功能，其中之一便是图形硬件辅助 CPU 进行实时碰撞检测。

Rossignac [Rossignac 1992]等利用深度和模板缓存来辅助进行机械零件之间的相交检测。他们通过移动图形硬件的裁剪平面，判断平面上的每个象素是否同时在两个实体内来确定物体是否相交。Shinya 和 Forgue[Shinya 1991]等提出在绘制凸体的同时，保存视窗口中每个象素上物体的最大和最小深度序列，并将它们按大小顺序排列，然后检测物体在某一象素上的最大深度值是否与其最小深度值相邻来判别相交情况。图形硬件可以支持物体最大、最小深度的计算。但该方法并不实用，因为它要求大量的内存来保存深度序列，而且从图形硬件中读取深度值本身就非常费时。他们的算法对图形硬件要求都比较高，只能在高端的图形工作站上运行，而且碰撞检测结果的精度也受到图形硬件性能的影响。

随着图形硬件性能的快速提高，利用图形硬件辅助碰撞检测重新激起了研究

人员的热情。Myszkowski 等[Myszkow 1995]将深度缓存和模板缓存结合在一起进行相交检测。它们用模板缓存值来保存视窗口中每个象素上所代表的射线进入一物体前进入和离开其他物体的次数。之后，读取模板缓存中的值来判断两物体是否相交。该算法仅能处理两个凸体之间的碰撞检测问题。

Baciu 和 Wong[Baciu 1997][Baciu 1999]在他们的论文中开始把图形硬件辅助的碰撞检测算法称为基于图象的碰撞检测算法，以与基于物体几何空间的碰撞检测算法[Lin 1998, Jiménez 2001]相区分。他们改进了 Myszkowski 等的方法，先采用几何的方法确定两物体包围盒的相交区域，然后在该相交区域中利用图形硬件的加速绘制来进行相交检测。该算法分析两物体的各种相交情况，并将这些相交情况按深度值顺序位置进行了分类，然后用模板缓存值来表示这些分类。这样，算法可以通过检查模板缓存值来判别两物体是否发生碰撞。

Lombardo 等[Lombardo 1999]用图形硬件来辅助进行虚拟手术中的碰撞检测。他们将虚拟手术所用的探头，用简单的长方体来表示。然后，将该长方体设定为图形显示中的视域，通过硬件的视域裁剪操作来实现探头与其他复杂物体的碰撞检测。该算法的局限性在于用长方体来表示物体过于简单，不具有通用性。

Vassilev 等[Vassilev 2001]在人体及衣物运动仿真中采用了基于图象的碰撞检测方法来检测人体与衣服之间的碰撞。他们利用图形硬件的深度缓存和颜色缓存来判别衣服和虚拟人是否发生碰撞。该算法有效利用了图形硬件来提高虚拟人与可变形衣物之间碰撞检测的速度，但对虚拟人的姿势有一定限制，且碰撞检测的精度不高。

Hoff 等[Hoff 2001] 结合了 Voronoi 区域的几何特性，利用图形硬件来处理二维模型之间的邻近性查询。算法首先采用几何的方法快速确定两个平面模型的包围盒的相交区域，然后用图形硬件的帧图象缓存快速找出更准确的相交区域，并生成 Voronoi 图用于进一步计算相交区域的距离场，最终计算出分离距离或刺穿距离以及接触点和法向量等邻近查询所要求的信息。该算法的缺陷在于只能处理二维模型之间的碰撞检测。

基于图象空间的碰撞检测方法一般将三维几何物体通过图形硬件投影绘制到图象平面上，降维得到二维的图象空间，然后通过在图象空间中对保存在各类缓存中的信息进行查询和分析，检测出物体之间是否发生干涉。然而，多数基于图象的碰撞检测算法仅能在凸体间进行碰撞检测，而且由于其绘制过程往往直接针对物体进行，缺乏必要的优化手段，其性能也不够理想。有鉴于此，本章提出一种能自动处理任意形状物体的、基于图象的实时碰撞检测算法。除了采用实时绘制技术以有效利用图形硬件的实时处理能力外，算法还结合 OBB 包围盒树来辅助图形绘制参数的设定以有效提高碰撞检测的效率。

本章其余部分安排如下：第二节主要对算法思想进行概述，算法分为预处理阶段与运行阶段；第三节阐述算法在预处理阶段的各个步骤；在第四节中，详细描述算法在运行阶段的主要任务；第五节给出实验结果并与相关算法进行比较分析；最后在第六节中对本章内容进行小结。

2.2 算法概述

算法从总体上分为两个阶段：一个是预处理阶段；另一个是碰撞检测运行阶段。

为了能处理任意形状多面体之间的碰撞检测，算法首先采用一种新的表面凸分解技术将物体分解为一系列凸面片的集合，并基于凸面片构造其相应的凸块（凸多面体）。随后，采用自顶向下策略将这些凸块合理地组织成为一棵层次二叉树，同时为每个凸块建构相应的 OBB 包围盒。最后，对该层次树的每个节点凸块进行三角形带（triangle strip）压缩编码以利于后面的实时绘制过程。这些步骤均在预处理阶段完成，其中凸块的 OBB 包围盒主要服务于两个目的：其一，作为凸块简化的几何表示用于进行潜在的碰撞初判；其二，用于快速设定绘制凸块时的视域参数。

在碰撞检测运行阶段，算法将遍历物体对的层次凸块二叉树。层次树中每个节点是由 OBB 包围的凸块。在同时遍历两物体的层次树时，先检测凸块节点所对应的 OBB 包围盒是否相交；当两 OBB 包围盒相交时，则基于两 OBB 包围盒的相交结果快速设置视域参数，实时解码、绘制凸块的压缩三角形带，并对绘制结果进行实时查询，分析得出碰撞检测的正确结果。算法的整体框架见图 2-1。

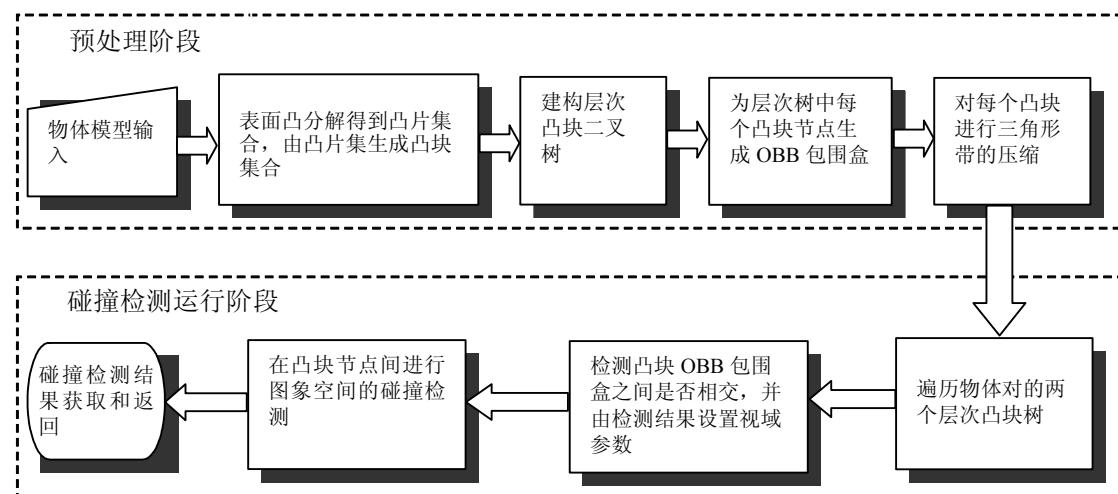


图 2-1 基于图象碰撞检测算法的基本框图

2.3 算法预处理阶段

如上所述，算法在预处理阶段中将对场景中每个物体进行以下四个步骤的处理：

- (1) 对复杂的非凸物体进行凸表面分解，得到一组凸片集合，并针对每一凸片构建与其相应的凸块；
- (2) 将凸块集合组织为层次二叉树；
- (3) 为层次树中每个凸块构建一个 OBB 包围盒；
- (4) 对层次二叉树上的所有节点中的凸块进行三角形带压缩编码。

下面我们分别对每一步进行详细介绍。

2.3.1 基于表面的凸分解

与一般物体相比，凸体更易于表示、操作和绘制。因此，当直接处理一般物体有困难时，一个常用的方法是首先将一般物体分解为凸体的集合，然后将问题转化为对凸体的处理。本章的算法也是这样。

凸分解方法分为两种：一是基于实体的凸分解；二是基于表面的凸分解 [Chazelle 1997a][Chazelle 1997b]。

基于实体的凸分解是将非凸物体分解为若干凸实体的集合，可由公式(2-1)来表示：

$$P = \bigcup_i C_i \quad \forall i, j : i \neq j \quad C_i \cap^* C_j = \emptyset \quad (2-1)$$

其中 P 表示物体， C_i 为凸块实体。

而基于表面的凸分解则是把非凸物体的表面分解成一些凸面片的集合，可用公式(2-2)表示如下：

$$\begin{aligned} boundary(P) &= \bigcup_i c_i \\ \forall i, j : i \neq j \quad c_i \cap^* c_j &= \emptyset \quad c_i \subseteq boundary(C_i) \end{aligned} \quad (2-2)$$

其中 P 表示物体， c_i 为凸的表面片，即凸片， $C_i = CH(c_i)$ ，表示凸片 c_i 的凸包体，本文称为凸块（如图 2-2）， $boundary(X)$ 表示物体 X 的边界面。

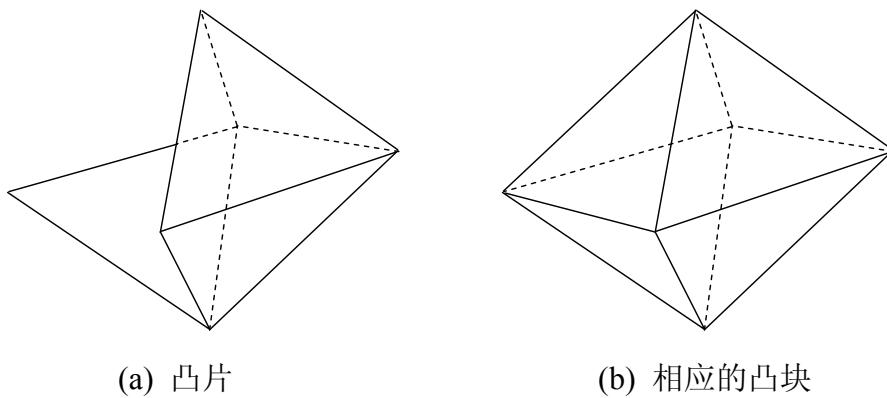


图 2-2 一个凸片及其相应的凸块实例

Chazelle 等[Chazelle 1997a]指出凸分解中如何找到最小个数的凸片集合是一个 NP 问题，并提出了将一非凸模型分解为较小数目的凸片集合的几种方法。Ehmann 等[Ehmann 2001]以 Chazelle 方法为基础，提出了一种面向多面体表示物体的凸表面分解方法使凸分解结果满足由公式(2-3)所表示的约束：

$$S \cap C_i = c_i \quad (2-3)$$

其中 S 为物体的表面。该约束的用意在于限定任意凸块与物体表面的交集必须是凸块所对应的凸片。Ehmann 等通过要求表面分解满足该约束来保证其基于物体空间碰撞检测算法的正确性。本章算法在预处理阶段采用了他们的表面凸分解方法对多面体进行凸表面分解。

该表面凸分解方法依据多面体表面面片的相邻关系，采用对偶图搜索方法和增量凸壳算法来搜集凸片集合。由多面体的顶点和边组成的图被称为多面体表面图。而物体表面的对偶图则是通过将多面体表面图中点与面的角色互换，边保持不变而得到的图。假设 S 是待分解表面，则将 S 分解为一组凸片的算法步骤为：

- (1) 首先，在 S 中选择一多边形作为种子面 $seed$ ，则种子面 $seed$ 构成当前连通凸片 c ；
- (2) 随后从 $seed$ 出发，通过深度优先或广度优先搜索物体表面 S 的对偶图，并依据一定的判别准则来决定是否把 $seed$ 的相邻面加入到当前连通凸片 c 中，如此递归搜索下去直到判别准则不能成立，此时即找到最大的当前连通凸片 c ；最后，重复上述过程直至处理完表面 S 中的每个多边形，得到所有凸片的集合。
- (3) 对所有的凸片求凸包得到与其相应的凸块。

为了保证连通凸片的构造，需要通过一定的判别准则来选择加入当前连通凸片的面。假定当前凸片为 c ，待判定的候选面为 f ，当前面与 f 相邻的边为 e ， f 上与 e 相对的顶点为 v 。将 v 加入 c 就相当于把 f 加入到 c 中。则判定 f 是否满足条件有以下三个准则：

准则 1: e 不能为凹边;

准则 2: 从顶点 v 看出去, 不能观察到当前凸片 c 在表面 S 中的面片;

准则 3: 把顶点 v 加入到凸块 C 中后, 形成的新凸块 C' 不能与任何不在原凸片 c 中的面相交。

图 2-3 展示了创建某一凸片时的遍历图例。其中, *seed* 为种子面, 面 F_1 由于违反了准则 1 不能被加入当前凸片, 面 F_2 则由于违反了准则 2 也不能被加入到当前凸片。图 2-4 和图 2-5 分别给出了对圆环面和对人手模型进行表面凸分解的结果。

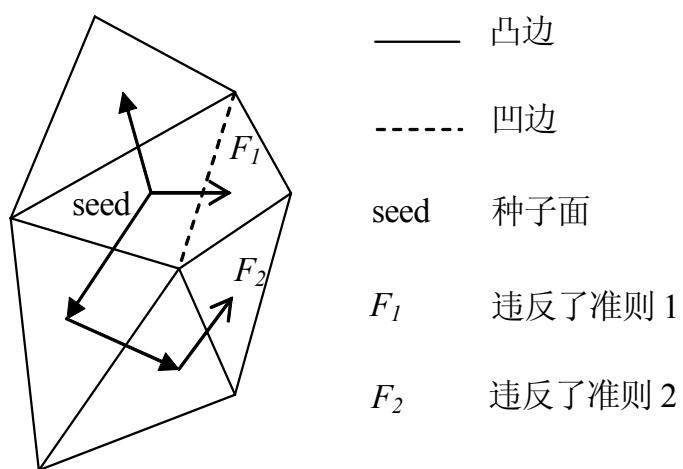


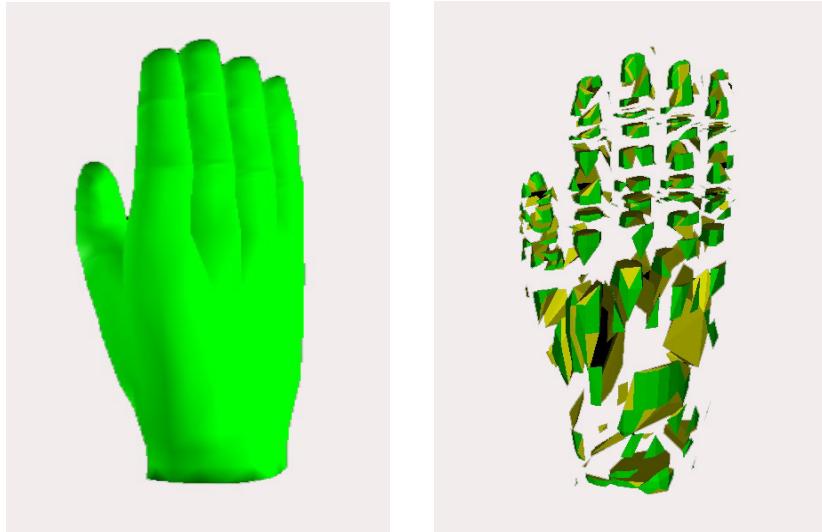
图 2-3 凸片生成时的遍历示例



(a) 圆环模型

(b) 圆环经表面凸分解后的凸片集合

图 2-4 圆环模型的凸分解



(a) 人手模型 (b) 人手模型经表面凸分解后的凸块集合
图 2-5 人手模型的凸分解

2.3.2 建构层次二叉树

对多面体 P 而言，上述凸分解过程产生出一组凸片集合 (c_i) 和与其对应的凸块集合 (C_i) 。值得注意的是，所有凸块的并集一般并不等同于实体 P 。尽管如此，可以肯定的是，由于上述凸分解过程遵循公式 (2-2) 这一约束，所有凸块的并集完全包含了多面体 P 的边界表面。因此，可将所有凸块的并集看作多面体 P 边界表面的包围体，用于辅助碰撞检测。

由于表面凸分解过程是一取决于物体形状的递归过程，因此所生成的这组凸块在空间排列上是无序的。为有效利用物体的空间连贯性以加速碰撞检测过程，本算法采用一种自顶向下的方法将凸块集（或称为凸块列表）组织成层次二叉树结构。具体步骤如下：

(1) 生成所有凸块列表 (L) 的凸包 (CH)，创建出层次树的根节点 N ，节点数据中包括 CH 和指向 L 的指针；

(2) 将列表 L 按照空间位置划分为不相交的两组子列表 (L_1 和 L_2)，具体方法后面再详细介绍；

(3) 分别生成两子列表 (L_1 和 L_2) 的凸包 (CH_1 和 CH_2)，并分别创建两个子节点 (N_1 和 N_2)，子节点数据分别包括了 L_1 或 L_2 ，以及指向 CH_1 或 CH_2 的指针；

(4) 指定节点 N 为 N_1 和 N_2 两个子节点的父节点。

(5) 递归上述过程，直到节点内凸块列表的个数为 1，即子列表中只有一个凸块的情况。

其中第(2)步的实现方法是先用凸块的质点来代表整个凸块，求出所有质点凸包的协方差矩阵，如公式(2-4)所示。

$$M_{jk} = \frac{1}{n} \sum_{i=0}^n (\nu^i - \frac{1}{n} \sum_{s=0}^n \nu^s)_j (\nu^i - \frac{1}{n} \sum_{s=0}^n \nu^s)_k \quad (2-4)$$

$$1 \leq j, k \leq 3$$

其中， ν^i 是指第*i*个凸块的质点的坐标向量，*n*为凸块的总数， M_{jk} 是 3×3 的协方差矩阵中的某一元素。

之后，求出该协方差矩阵的特征向量。由于协方差矩阵*M*是对称矩阵，故所求出的三个特征向量相互正交。将所有质点分别在三个特征向量的方向上做投影，其中，质点投影间隔距离最长的方向即为质点凸包的最大伸展方向。以此方向为剖分轴，划分质点的同时就将*L*划分为两个子凸块列表。该方法的框图如图2-6所示。

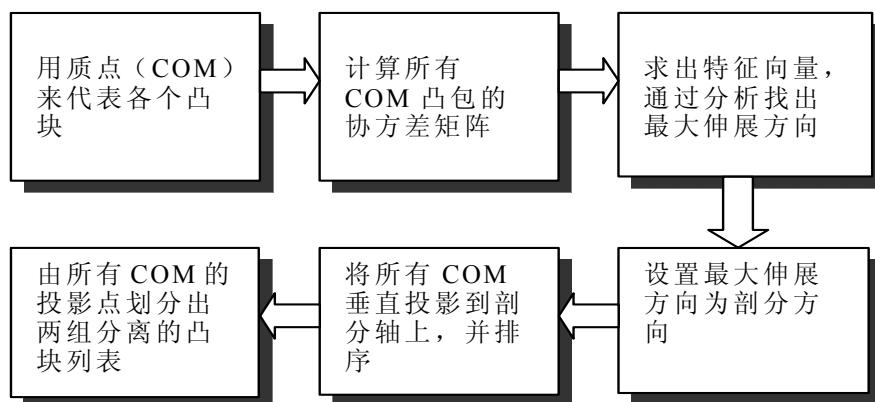


图2-6 凸块集(列表)的划分过程

划分列表的具体步骤是：

- 计算*L*中每个凸块的质点(COM)，并由COM来代表凸块；
- 计算所有COM凸包的协方差矩阵；
- 求出协方差矩阵的特征向量，通过分析找出COM的最大伸展方向；
- 将该最大伸展方向设置为剖分轴；
- 将所有COM垂直投影到剖分轴上，并排序；
- 按所有COM投影点的中间点将*L*划分为两个分离的列表*L₁*和*L₂*。

2.3.3 建构凸块的OBB包围盒

OBB包围盒已经被广泛应用于计算机图形学的各个领域，如碰撞检测和光线

跟踪等。它是指用一个与多面体有相近尺寸和方向的包围盒来近似多面体。本文采用了 Gottschalk 等[Gottschalk 1996]提出的一种计算三角网格体的 OBB 包围盒的方法来建构凸块的 OBB 包围盒，具体步骤如下：

首先，累计凸块所有顶点的坐标向量获取平均向量 μ ，如公式 (2-5) 所示；

其次，由平均向量计算出协方差矩阵 C ，如公式 (2-6) 所示；

再次，求出协方差矩阵 C 的特征向量，确定 OBB 包围盒局部坐标的三个轴向。由于协方差矩阵 C 是对称矩阵，其三个特征向量相互正交。将这三个特征向量单位化后，设定它们为凸块 OBB 包围盒的局部坐标的三个轴向；

最后，将凸块在三个轴向上的最大投影距离定为 OBB 包围盒的尺寸大小。

$$\mu = \frac{1}{3n} \sum_{i=0}^n (p^i + q^i + r^i) \quad (2-5)$$

$$C_{jk} = \frac{1}{3n} \sum_{i=0}^n (\bar{p}_j^i \bar{p}_k^i + \bar{q}_j^i \bar{q}_k^i + \bar{r}_j^i \bar{r}_k^i) \quad (2-6)$$

$$1 \leq j, k \leq 3$$

其中 p^i 、 q^i 和 r^i 是指第 i 个三角形面片的三个顶点的坐标向量， n 为三角形面片的总数， $\bar{p}^i = p^i - \mu$ ， $\bar{q}^i = q^i - \mu$ ， $\bar{r}^i = r^i - \mu$ ， C_{jk} 是 3×3 的协方差矩阵中的元素。

2.3.4 凸块的三角形带压缩编码

在基于图象的碰撞检测算法中，两物体间进行一次碰撞检测往往需要对物体进行多次绘制，因此加快绘制速度是提高基于图象碰撞检测算法效率的关键。物体加速绘制的常用方法有许多种，如视域裁剪技术、可见性裁剪技术、层次细节技术（LOD）、连通性压缩以及基于图象的绘制技术等等。其中，三角形带绘制技术通过重新组织物体的三角形面片以减少绘制过程中的数据传输量，并不改变物体的几何结构，因此，能较好地满足基于图象的碰撞检测算法中加速绘制过程的要求。

大家知道，对于图形硬件绘制而言，绘制速度受到顶点数据传输到图形硬件的速率的影响。对于三角形网格物体而言，绘制时若要将三角形的每个顶点都传送给图形硬件，会有许多重复。为减少重复数据传输的次数，一般可利用相邻三角形共享边这一属性，重新排列物体中三角形的顺序，形成由一系列连续相邻三角形组成的三角形带。这样，前一个三角形的两个顶点在绘制与其相邻的下一三角形时就可以重用。图 2-7 给出了一个三角形带的例子。

三角形带的序列:

(1, 2, 3, 4, 5, 6, 7, 8)

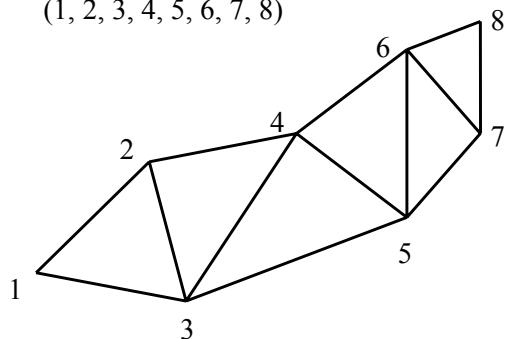


图 2-7 一个三角形带示例图

绘制三角形带时，除其中的首个三角形外，每次只需传输一个顶点即可绘制出一个新的三角形。所以，这种数据传输方式的理想加速比接近 3: 1。具体加速比例取决于三角形带的质量。一个模型中三角形带的长度越长，三角形带的数量越少，绘制加速比就越大。

目前可用于生成物体的三角形带的算法较多[Akeley 1990] [Evans 1996] [Xiang 1999] [Isenburg 2000]。Akeley 等[Akeley 1990]较早提出了一种 SGI 算法将三角形网格转化为三角形带。该算法以尽可能地减少孤立三角形为原则创建三角形带。作为一种贪婪算法，它尽量选择有最少相邻面的三角形作为下一个候选三角形，因此，仅能使用局部相邻信息来建构三角形带。

Evans 等[Evans 1996]在将 SGI 算法的局部搜索思想与全局搜索思想结合起来的基础上，采用一种称为“分片法”的方法对多边形网格模型的结构进行整体分析，并通过静态和动态全局的三角化过程来改善局部搜索算法的质量。即使对于那些不能完全三角化的多边形网格模型，“分片法”也能得到不错的结果。Evans 等还基于该算法开发了一个可快速生成三角形带的软件包“STRIP”。

Isenburg [Isenburg 2000]指出生成一个好的三角形带集合属于计算复杂问题，所以最好能一次性将其做完，把由三角形网格模型生成的三角形带保存起来以供绘制时使用。基于这种思想，他提出了一种网格编码压缩技术。该技术采用 STRIP 软件包的思想生成三角形带，然后利用三角形网格模型的连通性对生成的三角形带进行编码压缩。

本章算法采用了 Isenburg 用启发式搜索法生成三角形带并对三角形带进行编码的策略。在算法预处理阶段一次性对物体以及层次二叉树中每个节点凸块的三角形网格进行处理，得到比较理想的三角形带，并对这些三角形带进行压缩编码。

随后，在碰撞检测运行阶段，通过实时解码事先保存的三角形带来绘制凸块，可显著提高绘制速度，进而加速碰撞检测过程。图 2-8 展示了针对人手模型生成的三角形带，其中深黑折线指示出三角形带的走向。

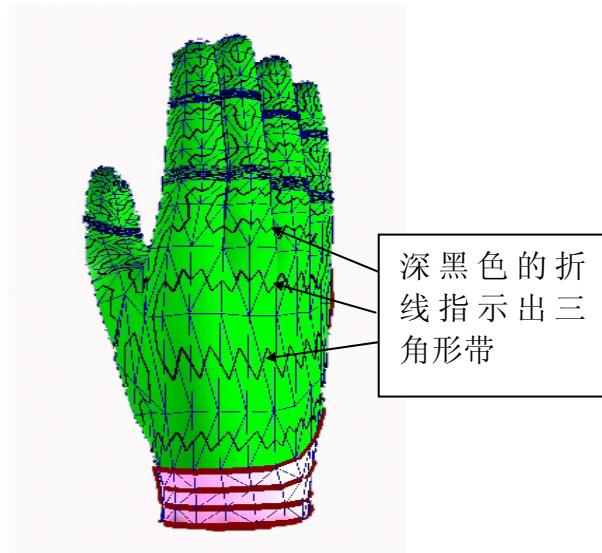


图 2-8 人手模型的三角形带

2.4 碰撞检测核心算法

在完成了上述预处理之后，碰撞检测算法转入核心阶段。算法首先同时遍历物体对的层次凸块二叉树，检测凸块节点所对应的 OBB 包围盒是否相交。在 OBB 包围盒相交的情况下，再利用相交结果设置视域参数，通过图形硬件的绘制过程进一步判别凸块之间是否发生碰撞。

2.4.1 层次二叉树的遍历

在预处理阶段，算法已为物体对的每个物体(A 和 B)各建构了一个层次二叉树，层次树的每个节点表示一个由 OBB 包围盒包围的凸块。于是检测两物体是否相交可通过同时递归遍历它们的层次二叉树进行。遍历采用广度优先(BFS)的策略进行，具体步骤如下：

(1) 遍历过程首先从两物体层次树的根节点开始，令 $BV_{A,0}^0, BV_{B,0}^0$ 分别为 A, B 层次树的根节点，设它们为当前遍历的节点。

(2) 判别两个当前节点所对应凸块的 OBB 包围盒之间是否相交，如果这两个 OBB 包围盒之间不相交，则递归返回。此时，若两个当前节点均为根节点，则 A, B 两物体不会发生碰撞，此遍历过程中止。

(3) 若当前两节点所对应凸块的 OBB 包围盒相交，则进一步判断两凸块之间是否相交，这个过程由图象空间的图形硬件绘制过程实现，将在 2.4.4 小节中详述。同样，如果两凸块不相交，则递归返回。此时如果当前节点为根节点时，返回 A, B 两物体不会发生碰撞，遍历过程中止。

(4) 如果这两凸块相交，遍历过程将从上而下递归遍历 $BV_{A,0}^0$ 的子节点 $BV_{A,1}^0, BV_{A,1}^1$ 以及 $BV_{B,0}^0$ 的子节点 $BV_{B,1}^0, BV_{B,1}^1$ 。依次设定当前节点对为 $(BV_{A,0}^0, BV_{B,0}^0), (BV_{A,1}^0, BV_{B,1}^0), (BV_{A,1}^1, BV_{B,1}^0)$ 和 $(BV_{A,1}^1, BV_{B,1}^1)$ ，递归执行上述(2)、(3)步骤，直到递归过程到达两个层次树的叶子节点后返回；

图 2-9 是层次二叉树相交检测遍历过程的一个例子。

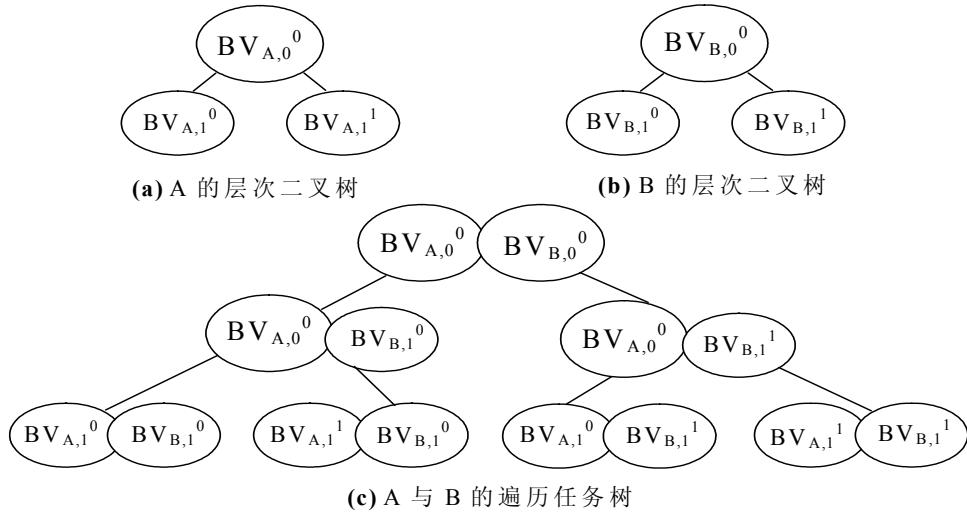


图 2-9 层次二叉树的遍历

2.4.2 凸块 OBB 包围盒的相交检测方法

凸块 OBB 包围盒之间的相交检测方法采用了 Gottschalk 等[Gottschalk 1996] 基于分离轴定理提出的相交检测方法。本算法在完成凸块 OBB 包围盒之间的相交检测的同时，还保留了部分相交结果的信息以快速地设置用于凸块之间相交检测的视域参数。

依据分离轴定理，首先指定 15 个分离轴。它们分别为两个 OBB 包围盒的 6 个轴向和由三个轴向与另三个轴向两两叉乘得到的 9 个向量。将这两个 OBB 包围盒分别向这 15 个分离轴投影得到相应的投影区间，通过检测这些投影区间是否重叠来判断两 OBB 包围盒是否相交。

具体实现中，算法逐次将两包围盒向所有 15 个分离轴做投影。对于每一个分离轴而言，均先将两包围盒中心点的矢量距离向该分离轴投影并计算出投影距离的长度。然后，分别计算两包围盒在该轴上的投影区间半径（投影区间半径是指投影区间长度的一半）。如果前者大于后两者之和，则在该分离轴上的两包围盒投影区间不重叠，进而得知两包围盒处于分离状态；反之，如果在所有 15 个分离轴上两包围盒的投影区间均发现重叠，则两包围盒相交。

每个分离轴的重叠判断方法进一步可描述如图 2-10 所示。

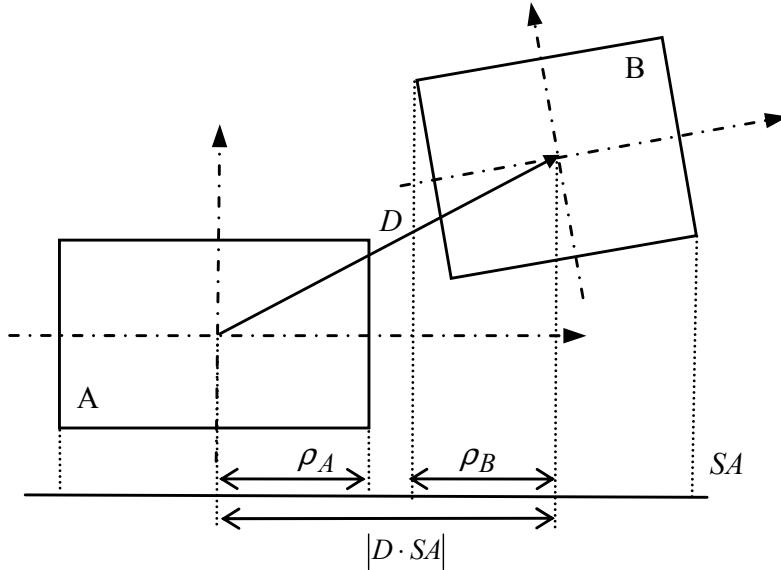


图 2-10 OBB 包围盒的相交测试

图中 A 和 B 为两个 OBB 包围盒, 向量 D 为 A 中心点至 B 中心点的距离矢量。单位向量 SA 为 15 个分离轴之一。两个投影区间中点之距离长度为 $|D \cdot SA|$ 。 ρ_A 和 ρ_B 分别为 A, B 在 SA 上的投影区间半径。于是, 我们可以得出投影区间处于分离状态的充分必要条件为:

$$|D \cdot SA| > \rho_A + \rho_B \quad (2-7)$$

即:

$$\varphi > 0 \quad \text{其中} \quad \varphi = |D \cdot SA| - (\rho_A + \rho_B) \quad (2-8)$$

这里 φ 的绝对值为在分离轴 SA 上的投影区间重叠部分的长度或为投影区间分离间隙的长度。令 a_i 和 b_i ($i = 1, 2, 3$) 分别为 A, B 的半径, 即分别为 A, B 两包围盒长宽高的一半; A^i 和 B^i ($i = 1, 2, 3$) 分别为两包围盒的坐标轴单位向量; 则两包围盒在 SA 上的投影区间半径 ρ_A 和 ρ_B 可由公式(2-9)得到。

$$\rho_A = \sum_i |a_i A^i \cdot SA| \quad \rho_B = \sum_i |b_i B^i \cdot SA| \quad (2-9)$$

当两 OBB 包围盒相交, 则它们在 15 个分离轴上的投影区间都是相交的。这时算法将保留在它们六个包围盒轴向上的投影区间重叠长度, 即公式(2-9)中的 φ 的绝对值。这些保留的 φ 的绝对值将用于凸块之间相交检测中的视域参数设定, 以加快绘制过程来提高碰撞检测速度。

2.4.3 用于相交检测的视域参数的设置

算法在检测到两凸块包围盒发生相交后，需要进一步判断两凸块是否相交。判断两凸体是否相交，算法是通过图形硬件对两凸块的绘制过程来实现的。为实现该过程，算法首先必须为此绘制过程设定相应的视域参数。

这些参数与一般的图形绘制相同，也包括了相机位置、视线方向、视口大小和投影透视矩阵。在本算法中，投影透视方式采用平行投影，其他视域参数将直接从 OBB 包围盒相交检测的结果中得到。

首先分两步找出视域的视线方向和视口大小：

(1) 在每个 OBB 包围盒的三个坐标轴中，确定重叠区间最长的一个坐标轴方向，剩下的两重叠区间构成一个矩形。

(2) 比较两个 OBB 包围盒中由最小重叠区间构成的矩形面积，选择面积小的矩形作为视口，则该矩形所在包围盒的第三坐标轴方向（即有最长重叠区间的坐标轴）设定为视线的方向。

然后，再依据视线方向、视口大小和 OBB 包围盒的尺寸来设定相机位置。如图 2-11 所示。

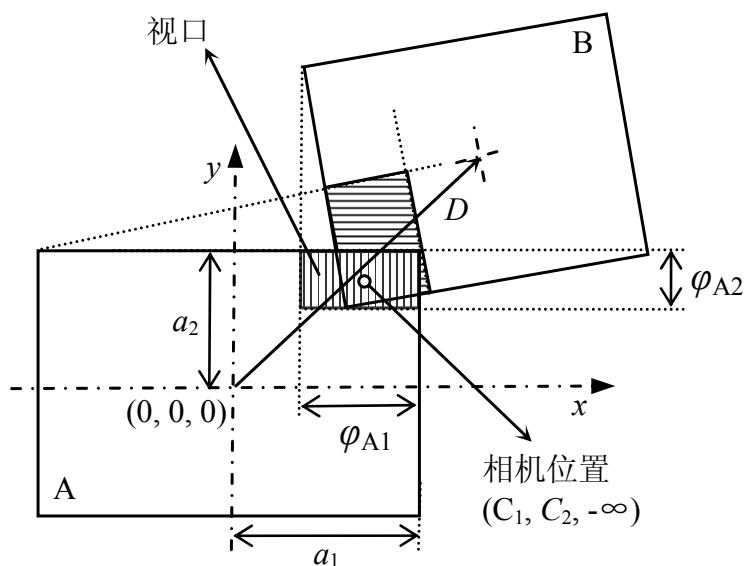


图 2-11 视域参数设置

图中，不失一般性地假设 OBB 包围盒 A 的 z 方向被设定为视线方向，而 x 和 y 方向上的重叠区域被设定为视口， $a_i (i = 1, 2, 3)$ 为 A 的半径， $\varphi_{Ai} (i = 1, 2, 3)$ 为两 OBB 包围盒在 A 的三个轴向上的投影重叠长度。则相机位置由 $(C_1, C_2, -\infty)$ 来确定，其中 C_i 由公式(2-10)来决定。

$$C_i = \pm(a_i - \varphi_{Ai} / 2) \quad i = 1, 2 \quad (2-10)$$

这里正负符号由 B 相对 A 的位置所决定，具体地说，当($D_i > 0$)，则 C_i 为正值；否则 C_i 为负。其中 D_i 为向量 D 的分量。

通过利用 OBB 的相交检测结果来设定视域参数不但可以迅速实现视域参数的设定，而且可以确保所设定的视口大小尽可能小，从而达到加快绘制过程的目的。

2.4.4 凸块之间的相交检测

两个凸块之间的相交检测通过利用图形硬件对它们进行实时绘制来实现。绘制时使用了深度缓存 (depth buffer) 和模板缓存 (stencil buffer)。深度缓存保持了在各象素上视点到当前所绘制物体之间的距离在[0,1]区间的映射值。深度缓存常用于隐藏面消除。模板缓存是帧缓存中的一个平面集，每个象素包含了一个或多个模板位。模板缓存中的值无法直接看到，但可以进行更改，主要用于控制相应颜色值的变化[Neider 1993][McReyn 1996]。

众所周知，对于任一凸体，经过屏幕象素的任意一条光线与它最多有两个交点，这两交点构成一段区间。不妨设 z 轴为深度方向，设两凸块为 A 和 B ，则 A 与 B 发生碰撞的充分必要条件为：

$$R_{xy}(A) \cap R_{xy}(B) \neq \emptyset \quad \text{且} \quad I_z(A) \cap I_z(B) \neq \emptyset \quad (2-11)$$

这里 $R_{xy}(X)$ 是物体 X 在 xoy 平面上的垂直投影区域； $I_z(X)$ 是物体 X 在 z 轴上所占的区间。换言之，两凸块 A 与 B 发生碰撞当且仅当 A 和 B 沿 z 轴的投影在屏幕上存在重叠区域，且在该重叠区域中，至少存在一个象素，在其上， A 与 B 在 z 轴方向上有重叠。基于这一条件，可以将三维物体的碰撞检测问题降维到二维图象空间，并最终简化到深度方向上的一维区间进行重叠检测。

对两凸块 A 、 B 而言，在某象素上 z 轴区间的重叠情况仅有 8 种，如图 2-12 所示。图中的 A_{min} 和 A_{max} ， B_{min} 和 B_{max} 分别为凸块 A 和 B 在该象素上的最小和最大深度值。

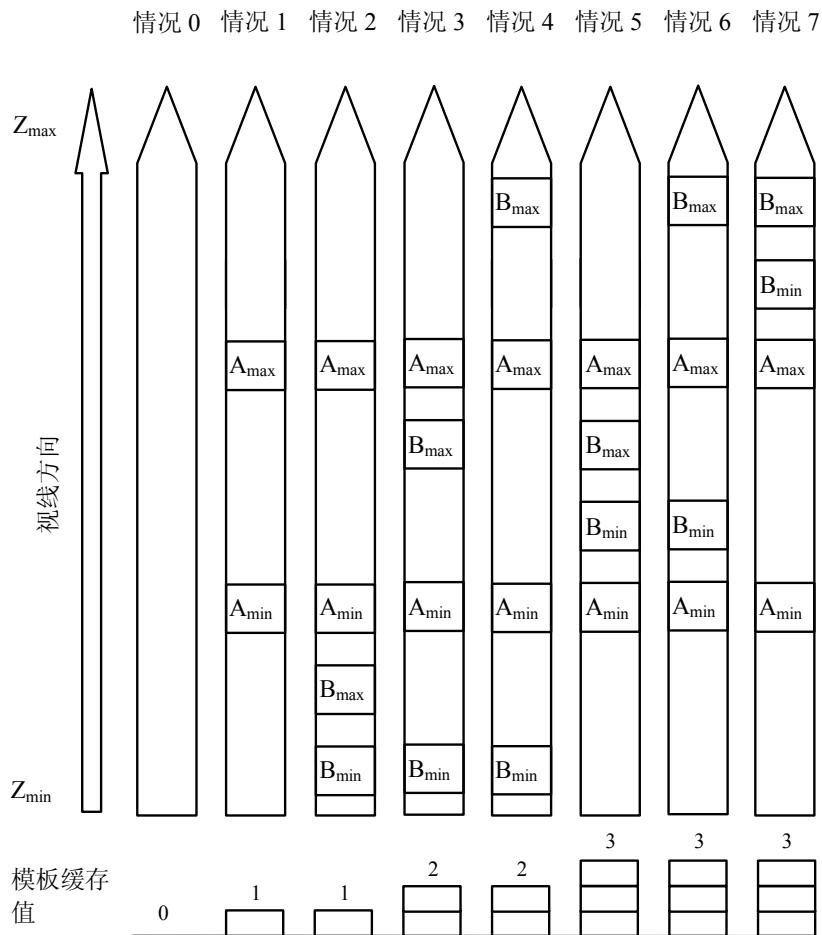


图 2-12 对应 z 的区间的 8 种位置关系图

当只比较 A 的正面和 B 的前后面之间的位置关系时, 又可进一步将这 8 种情况区分为以下四类情况, 并且可以用模板缓存的值 0~3 对它们进行区分:

- (1) 情况 0: 两个凸块的投影都没有覆盖象素, 这时模板缓存的值保持为 0;
- (2) 情况 1 和情况 2: 绘制 A 的正面面片所覆盖的象素, 其模板缓存值加 1。其中情况 2 中虽然 B 的正面和背面均存在面片覆盖上述象素, 但由于它们均处于 A 物体前面, 模板缓存值并不增加。因此, 模板缓存值为 1 表明在上述象素 A 与 B 沿 z 方向不相交;
- (3) 情况 3 和情况 4: 被 A 的正面面片覆盖的象素所对应的模板值先加 1。当 B 的正面存在面片在 A 物体之前, 同时 B 的背面部分存在面片在 A 物体之间, 模板缓存值增加到 2。象素中有模板缓存值为 2 的情况出现时, A 与 B 相交;
- (4) 情况 5, 情况 6 和情况 7: B 物体均在 A 的正面面片的后面, 即 B 物体比 A 的正面更远离视点。这些象素相对应的模板值将变为 3。这时 A , B 是否相交取决于 B_{\min} , A_{\min} 和 A_{\max} 的关系。

图 2-13 给出了在两凸块之间利用图形硬件进行相交检测的算法流程图。

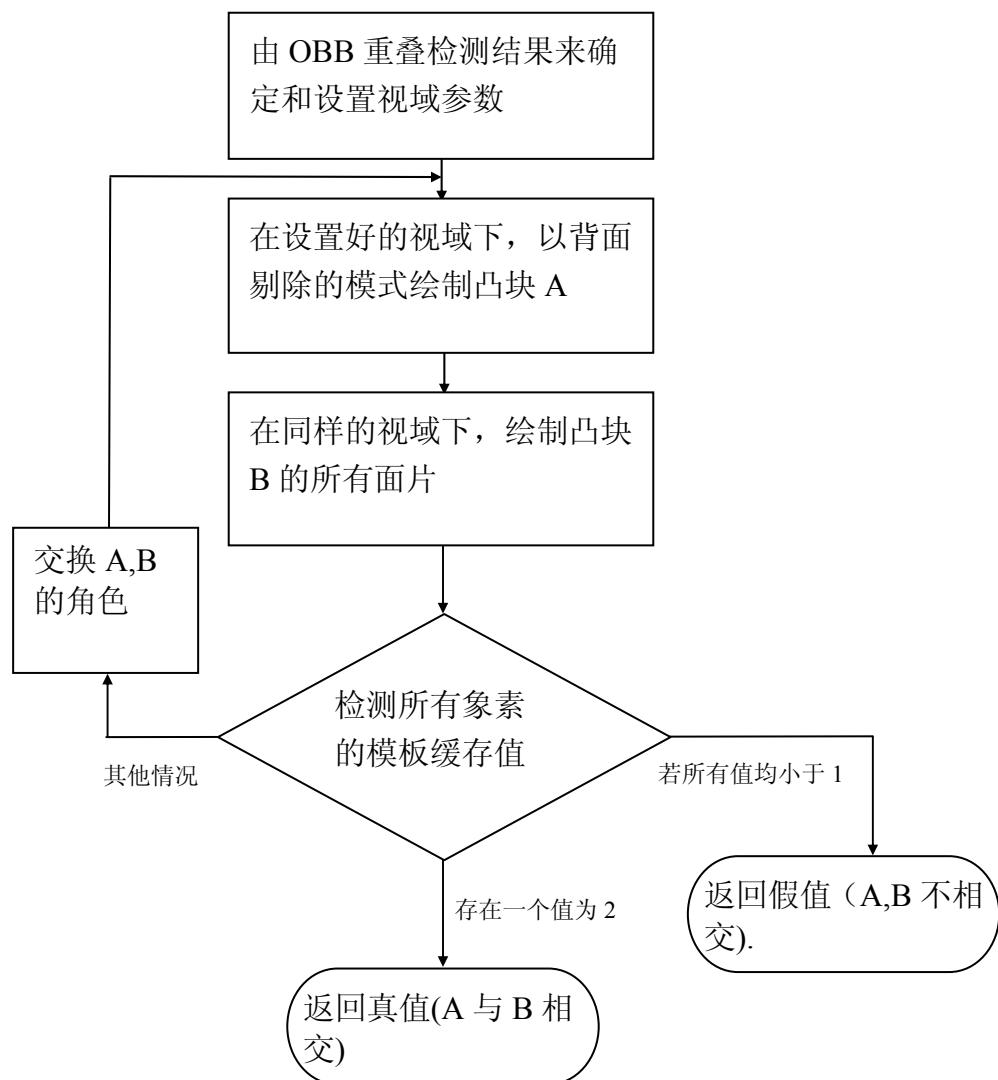


图 2-13 两凸块间基于图象碰撞检测算法的流程图

相交检测算法总体上分为以下五个步骤：

步骤 1：基于 OBB 相交检测的结果，确定并设定视域参数；

步骤 2：以此视域参数来绘制凸块 A，绘制时剔除了背面，从而在 A 所覆盖的象素区域上保留了 A 的最小深度值。同时设定对应的模板缓存值为 1。绘制时通过实时解码绘制以三角形带编码保存的凸块。

步骤 3：在同一视域内将凸块 B 分正面和背面两次进行三角形带解码绘制。每次绘制时，将 B 上象素点的深度值与当前深度缓存中的值进行比较，若小于当前深度值，则该象素的模板缓存值加 1。注意此步骤中深度缓存值保持不变；

步骤 4：检查视域中所有象素模板缓存的值，若全部小于或等于 1，则两凸体 A, B 不相交，返回假。若发现至少有一个值为 2，则 A, B 相交，返回真；其他情况，继续下一步；

步骤 5：互换 A 和 B 的角色，重复步骤 2~4，最终确定 A, B 是否相交。

图 2-14 给出了两凸块间基于图象碰撞检测算法的伪代码。

```

// 输入凸块 A 和 B; 输出 A 和 B 是否相交的布尔值。true 为相交, false 为不相交;
Bool ImageBasedCollisionDetection(A, B);
{
    If( A 的 OBB 与 B 的 OBB 不相交)          /* 检测的同时保存相交时的结果*/
        return false;                            /* A, B 不相交*/
    计算并设置视域参数;
    禁止写 frame buffer;                     初始化 Z-buffer 和 stencil buffer;
    调用 myRenderSetStencilByZ(A);            调用 myRenderTestStencilByZ(B);
    SecondRender = false;
    for( 视口中的每个象素 ){
        if( 发现某个象素的模板缓存值为 2 )    return true; /* A 与 B 相交 */
        if( 发现某个象素的模板缓存值为 3 )    SecondRender = true;
    }
    if(SecondRender==false)                   return false; /* A, B 不相交*/
    /* 交换 A 和 B 的角色 */
    重新初始化 Z-buffer 和 stencil buffer;
    调用 myRenderSetStencilByZ(B);            调用 myRenderTestStencilByZ(A);
    for( 视口中的每个象素 ){
        if( 发现某个象素的模板缓存值为 2 )    return true; /* A 与 B 相交 */
    } // end for
    return false;                            /* A, B 不相交*/
} // ImageBasedCollisionDetection Function end

myRenderSetStencilByZ(X)
{
    glEnable(GL_CULL_FACE);                /*采用背面剔除的绘制模式*/
    glCullFace(GL_BACK);
    glStencilFunc(GL_ALWAYS,0x01,0xff);     /*通过模板测试的象素置为 1*/
    glStencilOp(GL_KEEP,GL_KEEP,GL_REPLACE);
    允许写深度缓存;
    用三角形带解码绘制 X ;
}

myRenderTestStencilByZ(X)
{
    glStencilFunc(GL_EQUAL,0x01,0xff);    /*对于所有模板缓存值为 1 的象素操作*/
    glDepthFunc(GL_GREQUAL);             /* 使用深度测试, 并禁止写深度缓存 */
    glStencilOp(GL_KEEP,GL_KEEP,GL_INCR); /*深度测试和模板测试均通过时模板
                                         缓存值加 1; */
    glDisable(GL_CULL_FACE);            /*以非剔除的模式绘制 X 的前后面*/
    用三角形带解码绘制 X;
}

```

图 2-14 两凸块间基于图象碰撞检测算法的伪代码

2.5 实验结果与分析

算法采用 C++, OpenGL/GLUT 分别在 SGI ONYX2 工作站 (CPU 为 R10000) 和 PC 机(CPU PIV 1.6GHz, 内存 512M, 显卡 NVIDIA Geforce FX 5800, 显存 128M) 上实现, 其中凸包计算采用了共享软件包 QHULL (<http://www.geom.umn.edu/software/qhull/>)。

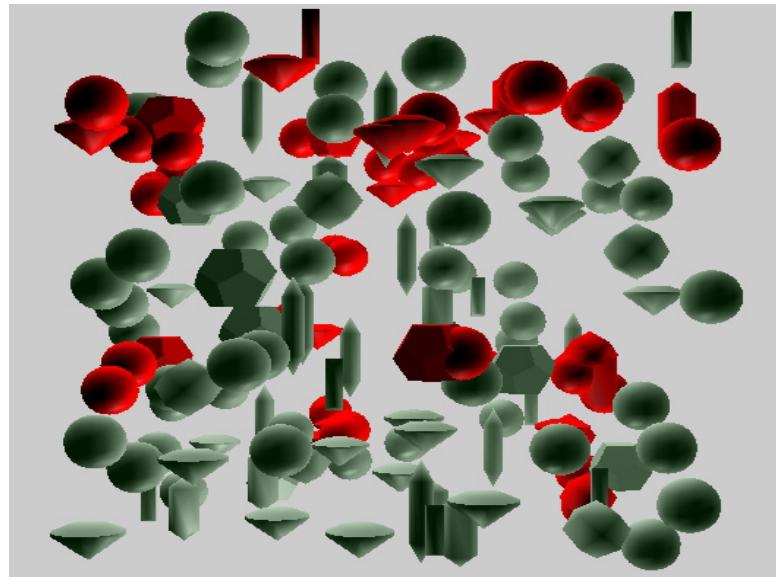


图 2-15 测试场景一

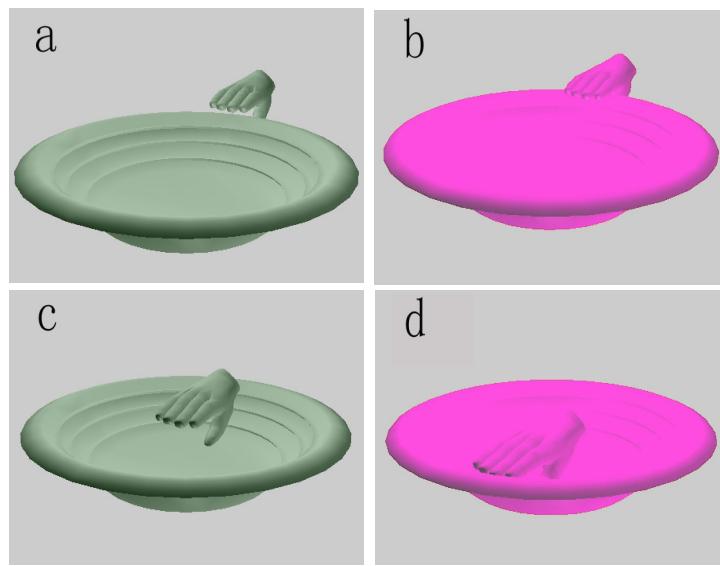


图 2-16 测试场景二

图 2-15 和图 2-16 所示为两个测试场景, 其中用红色显示的物体表示与其他物体发生碰撞。测试场景一中有 130 个物体做随机运动, 测试场景二中手在运动。测试针对每个场景和 4 种不同的碰撞检测算法 (RAPID[Baciu 1999],

RECODE[Gottschalk 1996]和本算法 IBCD) 分别进行。测试过程中记录每一步碰撞检测所需的时间 t_i 和运行 1000 步的平均时间 \bar{t} 。

图 2-17 和图 2-18 显示了在两个测试场景中随着场景复杂度的增加, 碰撞检测平均时间 \bar{t} 的变化情况。这些结果是在 SGI 工作站上测试得到的。

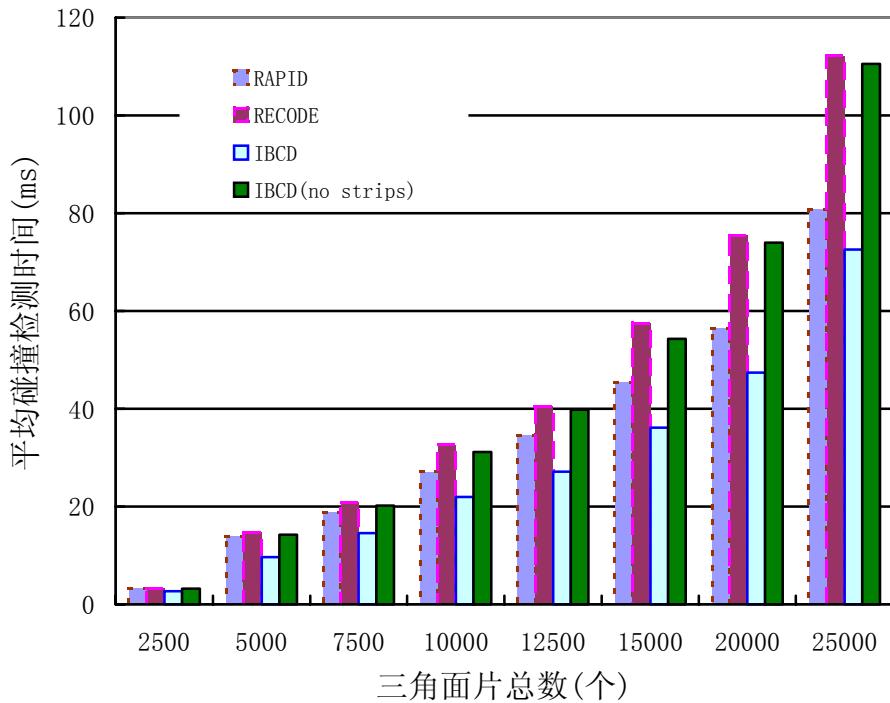


图 2-17 对场景一的测试结果的对比

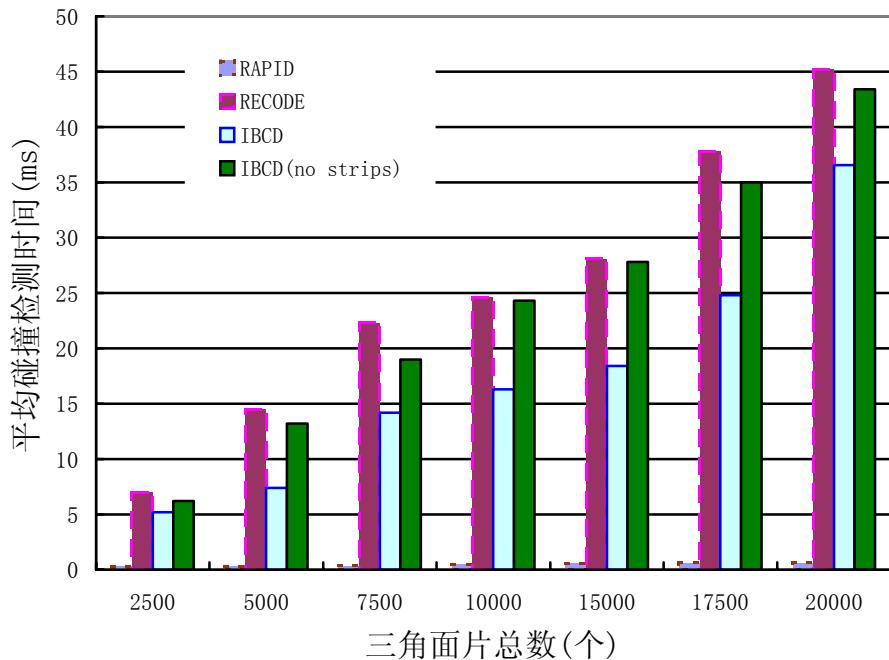


图 2-18 对场景二在 SGI 工作站下的测试结果的对比

图 2-19 是在 PC 机上用场景二对相关算法进行测试的结果。从实验结果可以看出，本算法比基于图象的碰撞检测算法 RECODE 在速度上有较大提高，其原因之一是采用了好的三角形带来加速绘制过程。

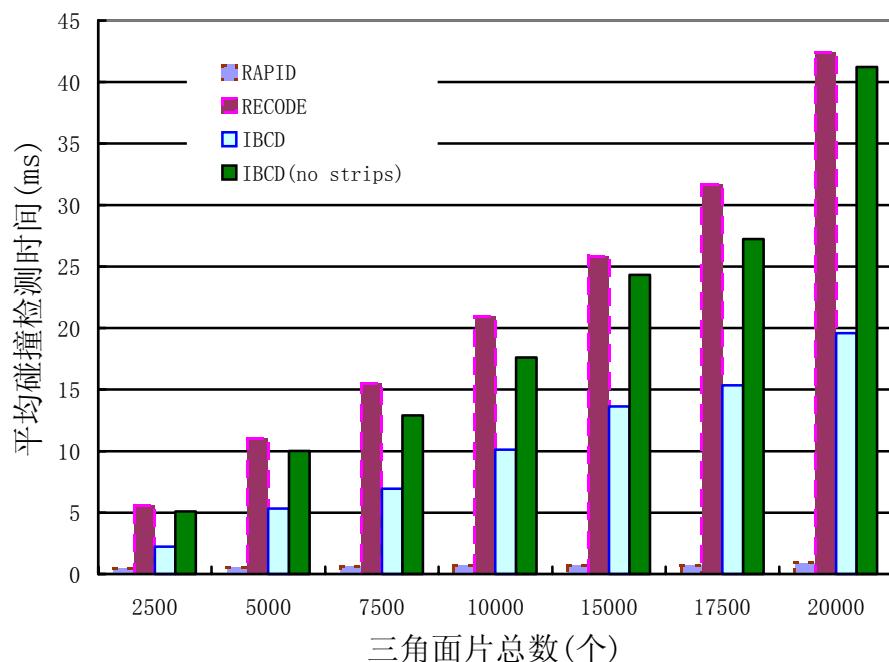


图 2-19 场景二在 PC 机下的测试结果比较

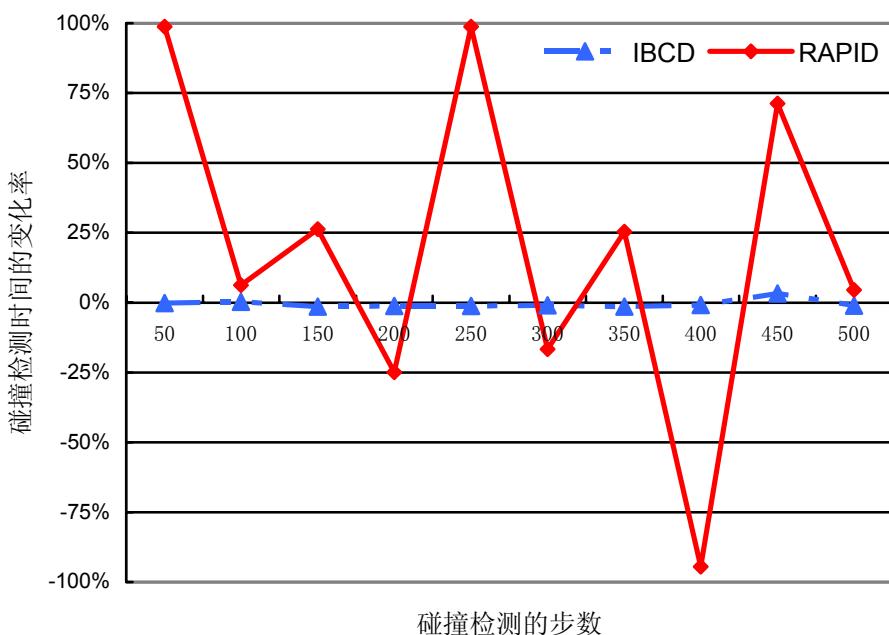


图 2-20 场景二中的碰撞检测时间变化率的比较图

图 2-20 揭示的是针对相同场景，本算法和 RAPID 算法的碰撞检测时间变化率的比较。其中水平轴是碰撞检测的步数，垂直轴是碰撞检测时间的变化率 η ：

$$\eta = \frac{t_i - \bar{t}}{\bar{t}} \times 100 \% \quad (2-12)$$

从图中不难看出，本算法继承了基于图象的碰撞检测算法的优点，具有基于物体几何空间的碰撞检测算法所不具备的时间消耗的稳定性，即对同一复杂度的场景而言，不同步数中的碰撞检测时间变化不大。这一特点有利于准确预测碰撞检测时间。

2.6 小结

本章提出了一种可处理任意形状物体间碰撞检测问题的基于图象的快速碰撞检测算法。算法结合了图形硬件的绘制加速性能和 OBB 包围盒的简化优势来提高复杂物体间碰撞检测的速度。算法巧妙地利用 OBB 包围盒之间的相交检测结果来设置图形硬件绘制时所需的视域参数，同时还融入了几何网格的相邻性压缩技术来加速图形绘制的过程，进而有效地提高了碰撞检测的效率。相对于基于物体几何空间的碰撞检测算法，该算法实现更为简单。而且与其他基于图象的碰撞检测算法相比，它能有效地处理非凸的复杂物体，具有更好的通用性。

该算法也有一定的局限性。作为一种基于图象的碰撞检测算法，由于图象本身的离散性，它的碰撞检测精度仍无法与基于物体几何空间的碰撞检测算法相比。此外，算法对内存的要求比较高，而且表面凸分解和凸包计算令算法在预处理阶段花费较多的时间。

第三章 基于流的实时碰撞检测算法

摘要

高性能可编程图形硬件的出现，正改变通用计算仅能由 CPU 完成的传统观念。本章探索性地采用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。

关键词： 碰撞检测 流计算 可编程图形硬件 通用计算

3.1 引言

图形硬件(GPU)最初仅作为一个图形绘制加速的协处理器来使用，而近些年由于其性能的急速增长，研究人员已经着手利用它来处理其他通用计算问题。在上一章中作者已详细介绍了这方面的相关工作。同时还提出了一种新的基于图象空间的碰撞检测算法，解决了图象空间碰撞检测算法较难处理非凸复杂物体的问题，并且采用绘制加速技术提高了碰撞检测的效率。但采用表面凸分解技术处理非凸复杂物体的代价较大，需要较长的预处理时间，此外，它还存在基于图象碰撞检测的算法所共有的缺点，即算法的精确度由采样图象的分辨率所决定。由图形硬件绘制的图象本身固有的离散性所带来的误差使得基于图象的碰撞检测算法在检测结果的精确度和准确性方面无法与基于物体几何空间的碰撞检测算法相比较。

近年来高性能可编程 GPU (programmable GPU, PGPU) 及其相关技术的出现，给我们解决上述问题带来了契机。随着半导体技术的迅猛发展，GPU 在性能上已经超越 CPU，并且 GPU 内在的并行结构还会使这种性能差距越拉越大；另一方面，图形硬件体系结构正发生质的变革，以前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) 和象素绘制处理器 (pixel shader/fragment program) 所取代[Lindholm 2001]。图形绘制流水线因此进化为一种通用的可编程流处理器，而不再只能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[Ati 2003]和 NVIDIA GeForce FX 系列[Nvidia 2003a]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通用的科学计算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一个活跃的研究领域。

在图形应用领域中，可编程 GPU 已用于过程纹理映射、实时绘制以及体可视化[Olano 1998][Peercy 2000][Proudfoot 2001][Cabral 1994]。

最近 Carr 等[Carr 2002]和 Purcell 等[Purcell 2002]将可编程 GPU 用于光线跟踪算法。Carr 等将所有的射线保存到纹理中，在绘制一个与视口等大小的矩形时进行纹理映射，同时由可编程的 GPU 进行光线跟踪计算。Purcell 等提出了相似的映射方法将光线跟踪算法映射到可编程 GPU 的流计算模型上，同时还比较了在有分支的可编程 GPU 和无分支采用多遍绘制的可编程 GPU 上实现光线跟踪算法的性能。他们的方法均通过可编程 GPU 的片断绘制程序来显著提高光线跟踪的效率。

在非图形领域，可编程 GPU 的应用也已经开始。Bolz 等[Bolz 2003]充分利用了可编程 GPU 的流计算模式和高度浮点运算性能处理两类数值仿真计算：一类是稀疏矩阵变化梯度求解计算；另一类是常规网格的多网格求解计算。他结合这两类问题的特点，有效地将两个问题的求解过程映射到 GPU 流水线上的片断绘制或象素绘制阶段。

Harris 等[Harris 2002]设计了一个利用可编程 GPU 处理动态物理现象的实时可视仿真器。他采用对偶映射网格（coupled map lattice, CML）来表示动态物体的状态，并以纹理来保存每帧场景状态的网格，然后通过可编程 GPU 的片断绘制来处理相关的仿真计算。具体地，他对蒸汽的物理现象用可编程 GPU 实现了实时仿真。

由此可见，可编程 GPU 的高性能和灵活性使其逐步具备了处理更为通用，在传统计算意义下难以处理的计算问题，如复杂三维物体的实时碰撞检测。所以本章探索性地采用可编程 GPU 来解决实时碰撞检测问题，将两个任意形状物体的碰撞检测过程映射到可编程 GPU 上，通过 GPU 的实时绘制过程计算出碰撞检测结果。在此基础上，还提出了两种有效优化措施以提高算法效率。

3.2 流计算模式

可编程 GPU 本质上是一种 SIMD 的并行模型，即单指令流多数据流模型。它以一种流计算模式来并行处理大量数据。为阐述如何在可编程 GPU 上设计与实现实时碰撞检测算法，首先介绍其处理数据的整个流水线过程。

3.2.1 可编程图形硬件

计算机图形硬件或 GPU 的最初设计目的是实现将多边形面片的顶点从模型坐标转换为视窗口内平面坐标的流水线。一旦完成这种转换，光栅化过程就用象素来填充这些多边形面片，同时以一定的透视方式对深度、颜色和纹理坐标等进行插值。在光栅化过程中，可通过插值后的纹理坐标检索纹理内容以及将纹理图象映射到多边形面片上。

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。如图 3-1 所示，这种可编程流水线的灵活编程性主要表现在两个部分：其一是顶点绘制阶段(vertex stage)的可编程顶点处理器 (vertex shader)；其二是片断绘制阶段(fragment stage)的可编程片断处理器或象素处理器 (fragment shader or pixel shader)。在这两个可编程处理器中执行的由用户自定义的程序分别称为顶点绘制程序(vertex program)和片断绘制程序(fragment program)。

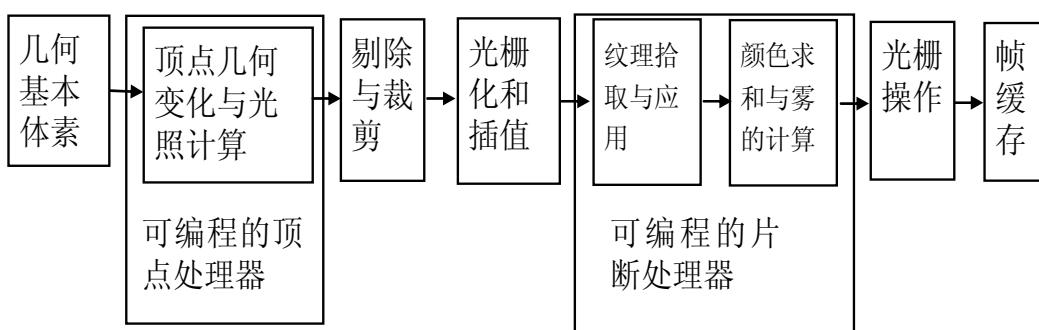


图 3-1 可编程图形流水线

可以说，可编程顶点处理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[Lindholm 2001]。而可编程的片断处理器则能够在多个纹理坐标下对单个象素进行算术操作及获取纹理采样数据，但这些单个象素上的操作相对独立，不能同时访问存储在其他象素上的数据。一般而言，片断处理器的运行速度要比顶点处理器快若干数量级。目前 NVIDIA 的一个顶点绘制程序通常由 128 个 4 路 SIMD 浮点指令组成[Nvidia 2003a]，其片断绘制程序也包括一个

4 路 SIMD 指令集，除此之外，它还增加了处理纹理数据的相关指令集。与顶点绘制程序不同的是，片断绘制程序在第四代 GPU 之前只能进行定点类型的数值计算，第四代 GPU 开始支持每个分量为 32 位的 IEEE 浮点类型的数值。

3.2.2 流计算模式

GPU 在硬件技术不断发展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大的影响。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，关键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有效地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或数组作为输入数据，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（如图 3-2）。

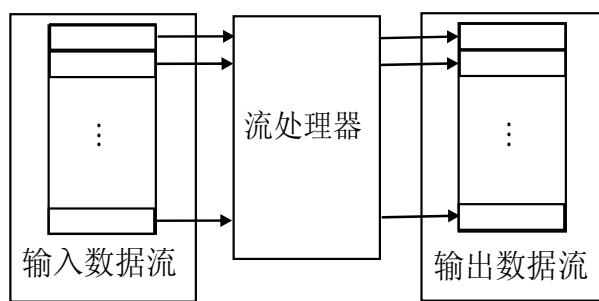


图 3-2 流计算模式

采用流计算模式进行计算有三个优势：

(1) 易于并行化。由于数据流的每个元素的计算都是独立的，就可让设计人员进一步采用流水线的计算方式来并行处理数据流的每个元素；

(2) 能够有效利用高带宽。流计算模型可对较小的数据记录进行大量的高强度计算，因此可有效使用内存的高带宽进行数据的读取；

(3) 消除潜在的内存重用危险。当图形硬件在一个片断中访问纹理时，片断寄存器以先入先出（FIFO）的方式保持获取的数值，这时片断处理器就可开始处理另外一个片断了，从而使基于流计算模式的图形硬件能够消除访问纹理时内存被重用的潜在危险。

可编程 GPU 正是以流计算模式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，两个处理过程通过数据流联系在一起。多数情况下，顶点处理程序因为不常用可以被忽略，因此可编程 GPU 又可被看作数据流的片断处理器。

3.3 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，本文采用检查一物体的所有边与另一物体的所有三角形是否有交的方法来确定它们是否发生碰撞。进一步，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。

3.3.1 边纹理生成

边纹理生成过程主要是依据物体 A 的边表生成边纹理。边纹理有两个，一个为起始点纹理，用于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 $e_ray(t) = orig + t \cdot dir$ ，则 $orig$ 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 。依据 A 的边表信息生成了起始点纹理和方向长度纹理后，其具体存放方式如下：

(1) 对于起始点纹理，图象中每个象素的三个颜色通道用于存放一起始点的位置向量 $orig$ ；

(2) 对于方向长度纹理，图象中每个象素的三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 则由该象素的 $alpha$ 通道保存。

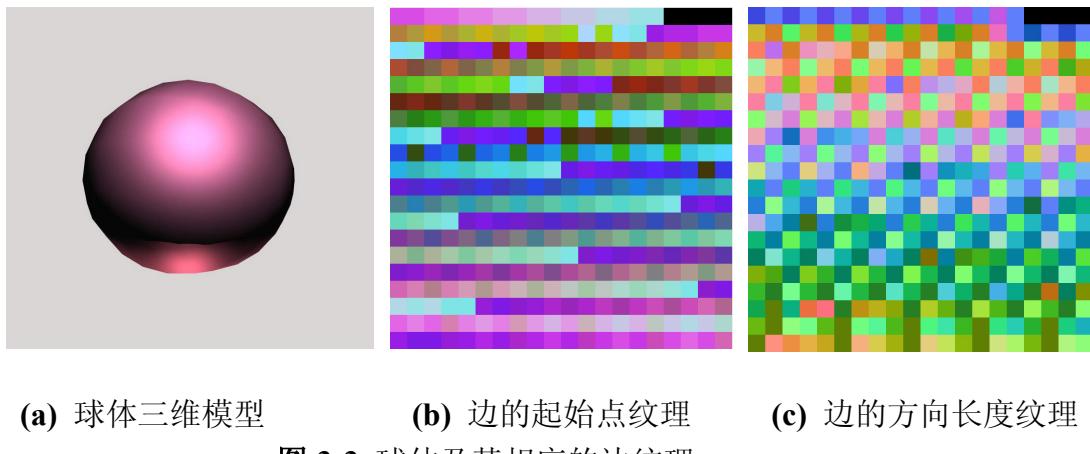
这两个边纹理大小相同，其每一纹理中相对应的象素表示 A 物体的一条边。纹理图象的大小由具体边数决定：

$$w = h = \lfloor \sqrt{n_e} \rfloor + 1 \quad (3-1)$$

其中 n_e 为 A 的边数， w, h 分别为纹理的宽度和高度。实现中，本算法采用了 NVIDIA 的矩形纹理技术[Nvidia 2003b]来生成边纹理。利用该技术，算法可灵活地存储不同大小的边纹理，并保证纹理中的非纯黑色象素和物体的边一一对应。

图 3-3 给出了一个有 396 条边的三维球体模型和由它所生成的边纹理的结果。其中图 3-3(a)为球体的三维模型，图 3-3(b)、图 3-3(c)分别为由球模型生成的边起始点纹理和边方向长度纹理。它们是由真实边纹理映射的象素颜色值到 [0,1]

区间后显示出来的结果，其分辨率均为 $20 \times 20 (=396+4)$ 。注意它们的右上角 4 个黑色象素没有和任何边相对应，这样可以保证算法检测结果的正确性。



(a) 球体三维模型 (b) 边的起始点纹理 (c) 边的方向长度纹理

图 3-3 球体及其相应的边纹理

3.3.2 边与三角形求交

边与三角形求交是指通过自定义片断绘制程序对 A 的所有边和 B 的所有三角形进行求交计算的过程。

基于流的碰撞检测算法将这一过程处理为一个循环过程，在循环的每一个迭代中，对 A 的所有边和 B 的一个三角形进行求交检测。为此，首先创建一个分辨率大小与 A 的边纹理相同的视口（viewport）专用于求交计算，并对相应的视域参数进行设定，同时建立一个大小与视口相同的矩形。在该视域中，采用平行透视投影的方式进行绘制。同时，为计算方便，视域的视线方向和相机位置均简单的指定为 $(0, 0, 1.0)$ 和 $(0, 0, -\infty)$ 。此外算法还禁用了该视口的颜色缓存以提高绘制速度，从而加速相交检测的计算过程，

在每一次迭代中，当前待处理的 B 的三角形的相关信息首先被存储到该矩形的四个顶点的多纹理坐标中，绘制该矩形的光栅化过程再将这些属性插值到视窗口的每个象素上。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象素上。最后在每个象素上用自定义的片断绘制程序对 A 的边和 B 的三角形进行求交计算。有关求交过程的具体细节将在 3.4 节中给出。

3.3.3 求交结果获取

求交结果获取是指通过硬件深度测试和遮挡查询（NV_occlusion_query）[Nvidia 2003b]快速获得碰撞检测结果的过程。一直以来，CPU 从 GPU 中读取数据的速度较慢，这已经成为一个影响 GPU 发展的瓶颈。第四代的 GPU 开始支持

通过硬件直接获取 GPU 部分信息的功能，即遮挡查询。遮挡查询通过检查视口内全部象素深度测试的结果，返回通过深度测试的象素个数。虽然遮挡查询只能获取深度测试的深度比较结果，而且只能给出满足条件的象素个数，还无法获取这些象素的位置，但它已经为解决这个瓶颈开了个好头。

由于 A 的所有边的长度值在求交计算之前已预先写入深度缓存，并且片断绘制程序在每个象素上将 A 的每条边与 B 的当前三角形的求交结果（即参数值 t ）作为对应象素的深度值输出，因此，通过简单的深度比较，即可得知 A 是否存在与 B 的当前三角形相交的边。具体地说，若存在某象素的 $t (>=0)$ 值小于等于对应边长度（对应于深度测试中的 GL_LESS 测试），则两物体相交，返回相应结果。否则继续进行算法的循环迭代过程，处理 B 的下一个三角形，判断 A 的所有边是否与该三角形相交，直到处理完 B 的所有三角形。这里，我们利用了硬件遮挡查询技术来快速直接地读取深度测试的比较结果。

作为总结，图 3-4 给出了基于流的碰撞检测基本算法的总体框架。

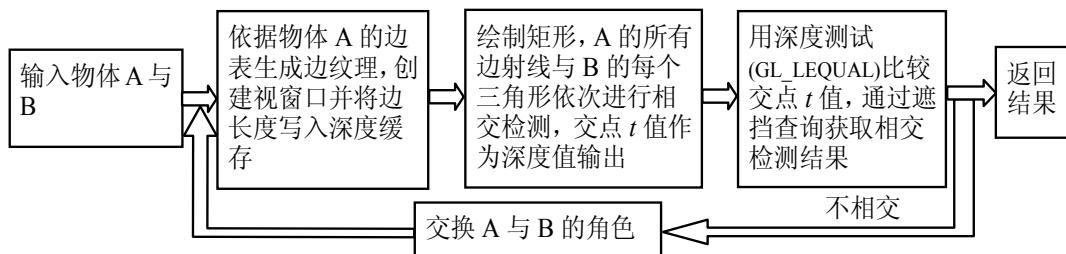


图 3-4 基于流的碰撞检测基本算法的总体框架

3.4 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。目前已有很多种实时绘制语言（Shading Language）可用于编写片断绘制程序，如北卡罗莱纳大学的 PfMan[Olano 1998]、SGI 的 Interactive Shading Language[Peercy 2000]、斯坦福大学的 Real-Time Shading System[Proudfoot 2001]、NVIDIA 的 C for Graphics(Cg)[Nvidia 2003c][Mark 2003] 和 Microsoft 的 High Level Shader Language(HLSL)[Microsoft 2003]等。实时绘制语言与 OpenGL 或 Direct3D 结合起来才可以实现具体的功能。

我们采用 NVIDIA 公司的实时绘制语言 Cg 编写边与三角形求交的片断绘制程序。算法实现中，采用了象素缓存技术[Wynn 2001]来分离用于碰撞检测的绘制和用于一般场景物体的绘制。象素缓存技术是指将绘制出的结果绘制到后台的象素缓存中的技术。利用象素缓存技术可以将场景的绘制与用于通用计算的绘制过程有效地区分开。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，算法还采用了浮点缓存技术[Nvidia 2003b]来准确处理浮点类型的数值计算。浮点缓存是在象素缓存技术基础上令图形硬件可处理浮点类型数值的技术。

下面我们分三部分具体介绍采用片断绘制程序实现边与三角形求交的过程。

3.4.1 数据流输入

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个部分：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。由于纹理图象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象素和 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。矩形的四个顶点属性设置相同，均保存了三角形的所有信息。如图 3-5 所示， v_1 ， v_2 ， v_3 表示 B 的当前三角形的三个顶点坐标， 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。

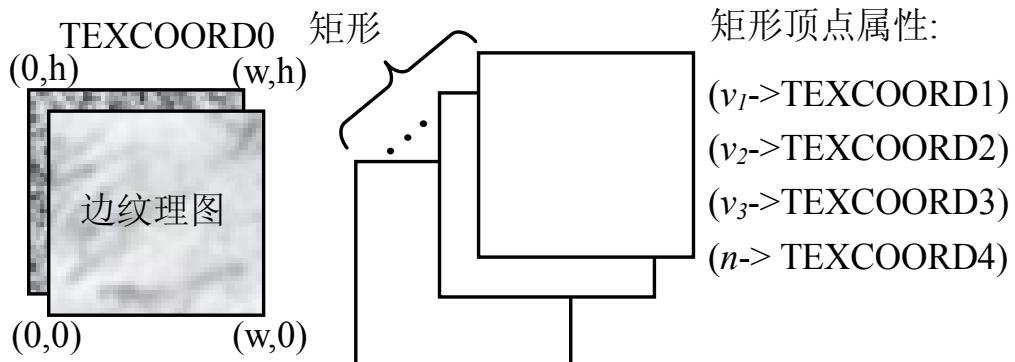


图 3-5 数据流的输入

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，只需将所建立的矩形的四个顶点的主纹理坐标分别设置为 $(0, 0)$, $(w, 0)$, (w, h) 和 $(0, h)$ 。光栅化过程将矩形的纹理坐标插值到每个象素上。片断绘制程序利用这些主纹理坐标可访问边纹理中的每个象素。具体的输入数据的数据结构在 Cg 中的定义如图 3-6 所示。

```

struct a2v : application2vertex
{
    float4 position      : POSITION;
    float2 TexCoord      : TEXCOORD0;        // the two texture coordinate
    float4 v1             : TEXCOORD1;        //the triangle first vertex
    float4 v2             : TEXCOORD2;        //the triangle second vertex
    float4 v3             : TEXCOORD3;        //the triangle third vertex
    float4 n              : TEXCOORD4;
};

struct v2f : vertex2fragment
{
    float4 position      : POSITION;        // clip space position
    float2 texCoord       : TEXCOORD0;        // the two texture coordinate

    float3 v1             : TEXCOORD1;        // first vertex
    float3 v2             : TEXCOORD2;        // second vertex
    float3 e12            : TEXCOORD3;        // e12 = v2-v1
    float3 e13            : TEXCOORD4;        // e13 = v3-v1
    float3 n              : TEXCOORD6;        // triangle normal
}

```

图 3-6 Cg 中边与三角形求交程序的输入数据结构

图中结构 a2f 为顶点绘制程序的输入数据结构，结构 v2f 为顶点绘制程序的输出数据结构，它同时又是片断绘制程序的输入数据结构。

3.4.2 相交计算

Möller 等[Möller 1997]提出了一个非常有效的射线与三角形求交算法。如前所述，顶点数据首先要通过图形流水线的顶点处理器进行坐标变换和光照等操作。作为可编程 GPU 的一部分，可编程的顶点处理器虽然速度较慢，但也可用来减少片断绘制程序的大小和计算负载。

本章的算法结合 Möller 等提出的算法，将边与三角形求交中部分较独立的计算转移到顶点处理器中执行。经过改造，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步骤如下：

(1) 仍假定边以射线的方式表示为 $e_{ray}(t) = orig + t \cdot dir$ ，其中 $orig$ 为边的一个顶点坐标， dir 为边方向矢量。 v_1, v_2, e_{12}, e_{13} 分别为三角形的两个顶点和两条边，如图 3-7 所示。

(2) 输入 $orig, dir, v_1, v_2, e_{12}, e_{13}$ 后，通过公式 (3-2) 至公式 (3-11) 的顺序计算，可得到 t, u, v, w 四个值。

$$v_{1o} = orig - v_1 \quad (3-2) \qquad v_{2o} = orig - v_2 \quad (3-3)$$

$$v_{1od} = v_{1o} \times dir \quad (3-4) \qquad v_{2od} = v_{2o} \times dir \quad (3-5)$$

$$e_{12d} = e_{12} \times dir \quad (3-6) \qquad det = e_{13} \cdot e_{12d} \quad (3-7)$$

$$t = -(n \cdot v_{1o}) / (n \cdot dir) \quad (3-8) \qquad u = e_{13} \cdot v_{1od} / det \quad (3-9)$$

$$v = -e_{12} \cdot v_{1od} / det \quad (3-10) \qquad w = u + v \quad (3-11)$$

(3) 当 u, v, w 均处于 $[0, 1]$ 区间时，则边与三角形相交， t 值为交点在射线的位置参数。

(4) 否则，二者不发生相交，将 t 值设定为最大限定值。

(5) 返回， t 参数作为深度值输出，用于深度测试。

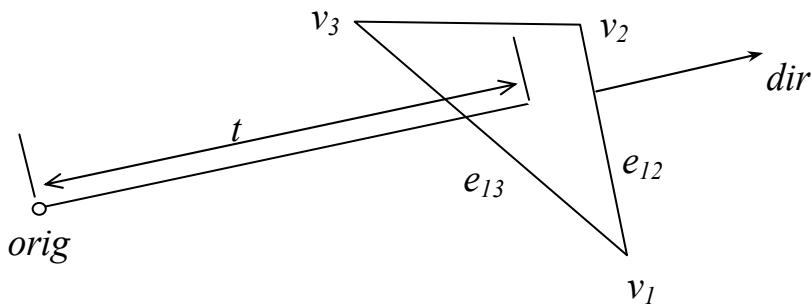


图 3-7 边与三角形相交示意图

本算法与 Möller 算法的不同之处是将三角形三个顶点的输入参数改为了输入两个顶点和两条边。其中两条边 e_{12} , e_{13} 由下面两个公式给出：

$$e_{12} = v_2 - v_1 \quad (3-12)$$

$$e_{13} = v_3 - v_1 \quad (3-13)$$

这样做的目的是尽量减少片断绘制程序相交检测的计算量。三角形两条边的计算与另一物体边集合的信息无关，不需要通过边纹理来获取，因此可将它们转移到可编程的顶点处理器上，由顶点绘制程序来完成。通过图形流水线的顶点绘制程序处理，矩形的顶点属性被更新为 v_1 , v_2 , n , e_{12} 和 e_{13} ，并被用作片断绘制程序的输入参数，然后再由光栅化过程插值均匀分布到视口的每个象素上。图 3-8 所示为边与三角形求交的片断绘制程序的 Cg 代码。

输入：射线原点、方向；三角形的法向量、两个顶点坐标向量和两个边向量；

输出： t 参数值

```
float4 Detect_Rays_with_Triangle(out float t, in float3 orig, in float3 dir, float3 v1, float3 v2, float n,
                                    float3 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;
}
```

图 3-8 边与三角形求交的片断绘制程序 Cg 代码

3.4.3 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，由于个人计算机采用异步 AGP 总线结构，导致从 GPU 中接收数据比向 GPU 发

送数据慢许多，因此通过 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以通过硬件直接获取深度测试的比较结果。我们采用上述技术快速获取碰撞检测的输出结果，从而巧妙地避免了由图形硬件读写速度不对称而导致的、相对慢得多的深度缓存读取操作。

3.5 优化的基于流的碰撞检测算法

在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绘制时每次需要进行多条线段和一个三角形的相交检测，其复杂度为：

$$Cost_w = N_r \times N_t \times Cost_{rd} + N_r \times Cost_{rb} \quad (3-14)$$

其中 N_r 和 N_t 分别为边和三角形的个数， $Cost_{rd}$ 和 $Cost_{rb}$ 则分别为 CPU 向 GPU 填充和从 GPU 中读取一个象素的时间开销。 $Cost_{rb}$ 的倒数为 GPU 的读取速率，这里读取速率是指读取字节数/秒。而 $Cost_{rd}$ 的倒数则为 GPU 的填充速率，它是指填充象素个数/秒。这里由于传输的字节数与象素个数是非线性关系，故填充速率用象素个数/秒而不是字节数/秒来度量。本章算法使用的纹理不多，一个循环中仅需进行两次纹理映射，故可以简单地用边或象素个数除以填充速率来表示片断绘制程序的时间开销。

上述公式表明，基于流的碰撞检测基本算法的时间复杂度与边的个数呈线性关系，与三角形个数呈非线性关系。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。由于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，只能尽可能减少输入数据流中的边的个数和三角形个数。为此，本章给出以下两种优化方法。

3.5.1 层次树优化法

层次树优化法的目的是尽量减少输入待检测的三角形个数。具体方法是，在基于流的碰撞检测算法的循环检测过程中引入层次树的递归遍历，从而减少循环次数。仍采用前面提到的例子，我们首先对物体 B 进行预处理，将其组织为一棵层次包围盒二叉树后，将物体 A 的所有边先与物体 B 的层次包围盒树的根节点包围盒求交。该求交过程同样采用 GPU 的片断绘制程序实现。当 A 的所有边都不与之相交，则返回两物体不交的结果；否则，递归判断其两个子节点包围盒是否与 A 的任一边相交，直到叶节点中的三角形。由于 OBB 包围物体较紧密，且其创建和求交计算不太复杂，作者仍采用 OBB 树作为层次包围盒树[Gottschalk 1996]。

采用片断绘制程序实现边与包围盒求交检测，其方法与采用片断绘制程序实现边与三角形求交检测类似，只不过所建立的矩形的顶点属性存放的是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[Williams 2001]给出了一种快速可靠的射线

与包围盒求交的算法，作者将其改写到片断绘制程序中实现了用片断绘制程序来计算边与包围盒的相交检测。具体的 Cg 代码如图 3-9 所示。

```

输入：射线原点、方向和包围盒的最大最小点坐标向量； 输出：t 参数值
float4 Detect_Rays_with_BoundingBox( out float t, in float3 orig, in float3 dir, float3 boxmin,
                                         float3 boxmax )
{
    bool bHit = true;

    float tmin = (dir.x >= 0.0f) ? ((boxmin.x-orig.x)/dir.x) : ((boxmax.x-orig.x)/dir.x);
    float tmax = (dir.x >= 0.0f) ? ((boxmax.x-orig.x)/dir.x) : ((boxmin.x-orig.x)/dir.x);
    float tymin = (dir.y >= 0.0f) ? ((boxmin.y-orig.y)/dir.y) : ((boxmax.y-orig.y)/dir.y);
    float tymax = (dir.y >= 0.0f) ? ((boxmax.y-orig.y)/dir.y) : ((boxmin.y-orig.y)/dir.y);

    bHit = (tmin > tymax) ? false : bHit;           bHit = (tmax < tymin) ? false : bHit;
    tmin = (tymin > tmin) ? tymin : tmin;          tmax = (tymax < tmax) ? tymax : tmax;

    float tzmin = (dir.z >= 0.0f) ? ((boxmin.z-orig.z)/dir.z) : ((boxmax.z-orig.z)/dir.z);
    float tzmax = (dir.z >= 0.0f) ? ((boxmax.z-orig.z)/dir.z) : ((boxmin.z-orig.z)/dir.z);

    bHit = (tmin > tzmax) ? false : bHit;           bHit = (tmax < tzmin) ? false : bHit;
    tmin = (tzmin > tmin) ? tzmin : tmin;          tmax = (tzmax < tmax) ? tzmax : tmax;
    bHit = (tmax >= 0.0f) ? bHit : false;          t = (bHit) ? tmin : MaxT;
}

```

图 3-9 边与包围盒求交的片断绘制程序 Cg 代码

上述优化算法通过使用层次包围盒树快速排除了多数与边集不相交的三角形，从而减少了所需绘制的矩形个数，提高了算法总体效率。

3.5.2 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 的所有边与物体 B 的一个节点包围盒发生相交时，保留所有相交的边，重新组织边表，并依据新的边表重新构建两个新的边子纹理。为获取物体 A 的所有边与 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象素位置。为了解决这一问题，优化算法在进行深度测试的同时增加了模板缓存测试，并用模板缓存来识别相交边对应的象素位置。这样，可将识别出来的相交边重新组织为新的边表，依据新的边表来创建两个新的更小的边子纹理。然后采用新的边子纹理，分别与 B 层

次包围盒树上的当前节点的两个子节点包围盒进行求交计算，如图 3-10。如此递归计算下去，直到 B 层次包围盒树的叶节点中的三角形。一般地，由于边纹理在求交计算中逐层变小，GPU 所需绘制的象素个数也逐步减少，从而可大大加快绘制速度，提高算法整体效率。

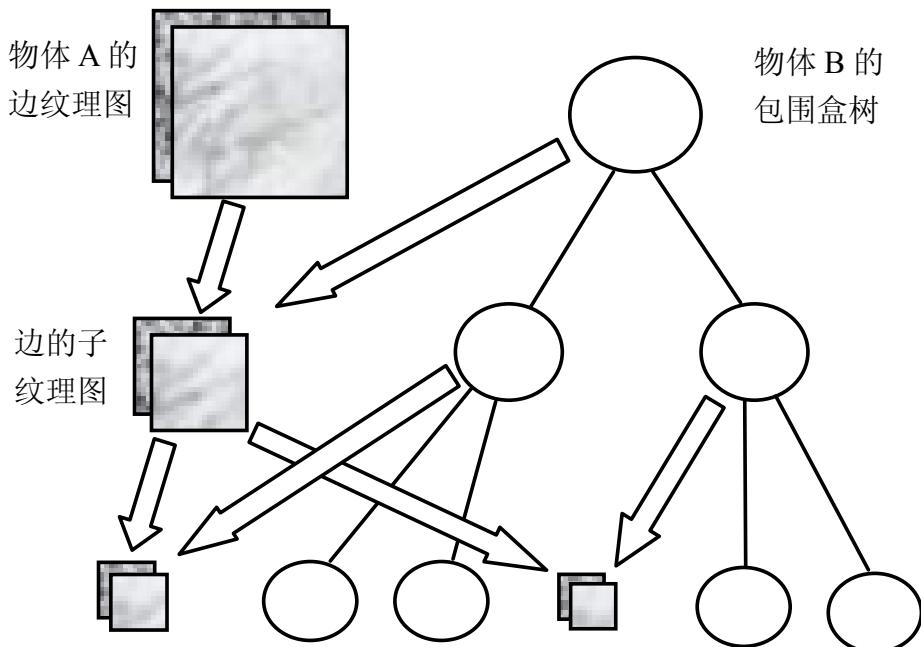


图 3-10 子纹理优化过程

3.5.3 优化的基于流的碰撞检测算法

基于上述两个优化方法，作者进一步提出了以下优化的基于流的碰撞检测算法，具体步骤如下（这里仍假定 A 与 B 的边纹理和 OBB 树都已经创立）：

首先处理 A 的边纹理和 B 的 OBB 树之间的相交检测，创建一个与 A 纹理相同大小的视口，并设定视域参数。设定 B 的 OBB 树 (T) 的根节点为当前节点 (CO)，并将 A 所有边的长度通过纹理映射一次写入深度缓存；

其次，在视口中绘制一个与视口等大小的矩形 (Q)，并用该矩形的所有顶点属性保存 CO 的各项参数。绘制中，首先用 GPU 的片断绘制程序计算 A 的边是否与 CO 相交，然后通过深度比较和模板测试，并读出模板缓存，得到与 CO 相交的所有边的集合 (E)；

再次，判断 E 是否为空，若 E 为空，则判断 CO 是否为根节点。若 CO 为根节点则直接返回未发现碰撞；否则交换 A, B 的角色重复上述过程；若 E 不为空，则由所有的相交边重新组织为新的边表，依据该边表创建边的子纹理；同时由相交边的个数重新设定视口的大小；

最后，递归计算新的边纹理是否与 CO 的两叶子节点的 OBB 包围盒相交，直到 T 的叶子节点为止。若当前 OBB 包围盒为叶子节点，则将视口中绘制的矩形的各顶点属性改为包围盒所包含三角形的各项参数，即三角形的三个顶点坐标和法向量。再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相交检测结果。

图 3-11 给出了优化的基于流的碰撞检测算法的具体流程图。

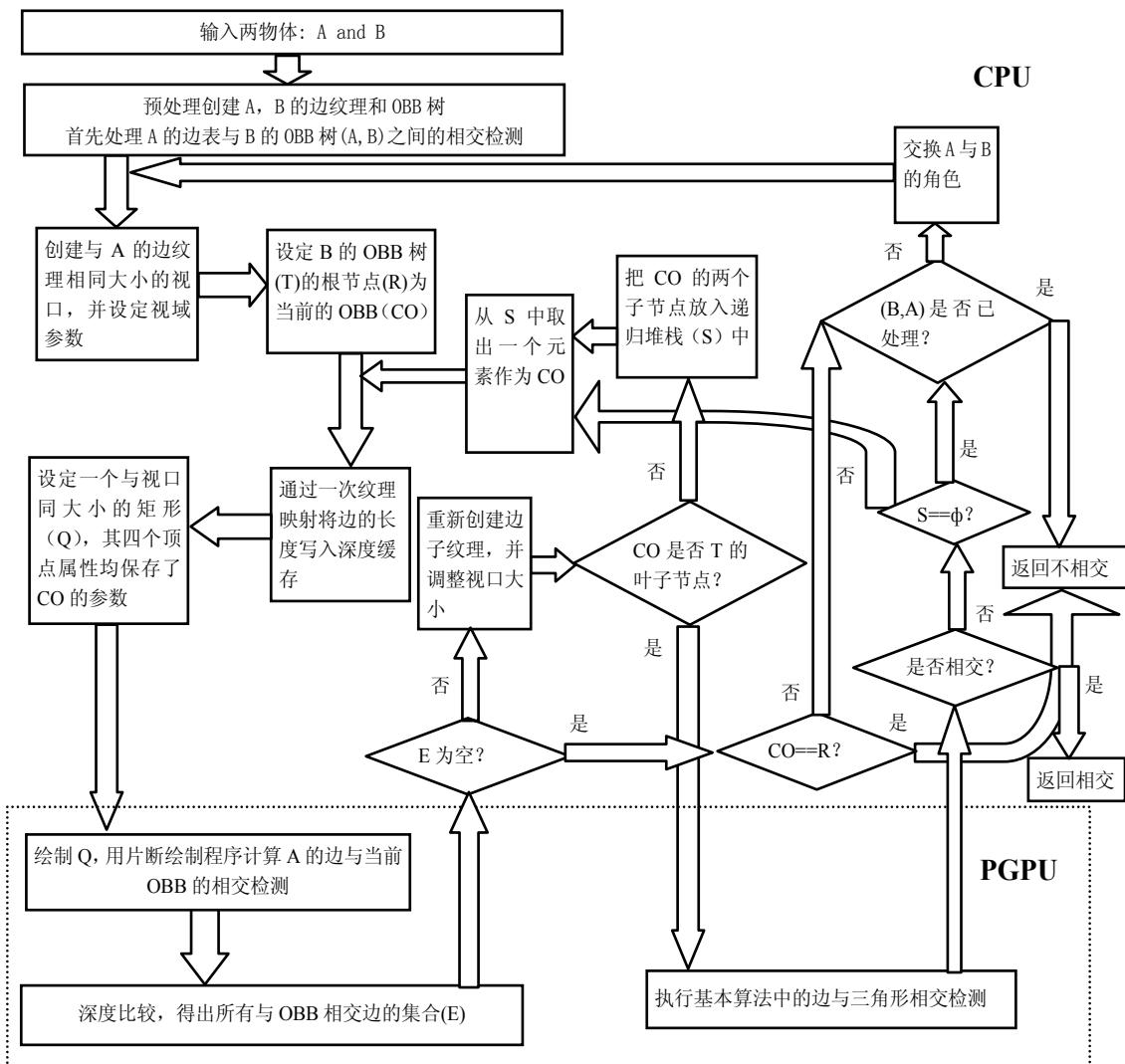


图 3-11 优化的基于流的碰撞检测算法流程图

3.6 实验与结果讨论

基于流的碰撞检测算法及其优化算法已在 PC 机(CPU P4 1.6GHz, 内存 512M, 显卡 NVIDIA Geforce FX 5800, 显存 128M)平台上实现。NVIDIA Geforce FX5800 的图形卡用于支持象素缓存、浮点缓存和顶点与片断绘制程序等。

实验中, 为全面比较基于流的碰撞检测算法的效率, 作者在多个不同复杂度的场景下对算法进行了测试, 并将其与基于图象的典型算法 RECODE[Baci 1999] 和基于物体空间的经典算法 RAPID[Gottschalk 1996]进行了比较。图 3-12 至图 3-15 所示为四个测试场景, 其三角面片总数分别为 21,292、18,217、40,419 和 18,378。在这些测试的场景中, 优化的基于流的碰撞检测算法性能不仅优于 RECODE, 而且对于复杂场景如场景 3, 性能甚至明显超过了 RAPID。

为了验证 GPU 辅助碰撞检测计算的强大性能, 作者在场景一中对不同复杂度的相同物体, 用 GPU 和 CPU 分别实现本章基于流的碰撞检测算法来测试它们相交检测的速度。从图 3-16 可以看出, 采用 GPU 实现的基于流碰撞检测算法在性能上要明显优于用 CPU 软件仿真实现的算法。从而说明了本章算法充分利用了 GPU 的并行计算的特性和高计算性能。

实验结果见表 3-1 和图 3-17 通过实验结果可看出, 基于流的碰撞检测算法在采用了两种优化方法后不同程度地提升了算法的总体效率, 更充分地利用了可编程 GPU 高性能的计算能力。而且场景越复杂, 碰撞检测的效率提升得越显著。

为了与本文上一章的算法进行比较, 作者还在场景 4 下进行不同复杂度的算法性能测试, 如图 3-18。比较图 2-18, 可以发现本算法不但在场景复杂度较高时碰撞检测速度比第二章的算法更快, 而且相交检测的时间相对更加平稳, 具有更良好的稳定性。

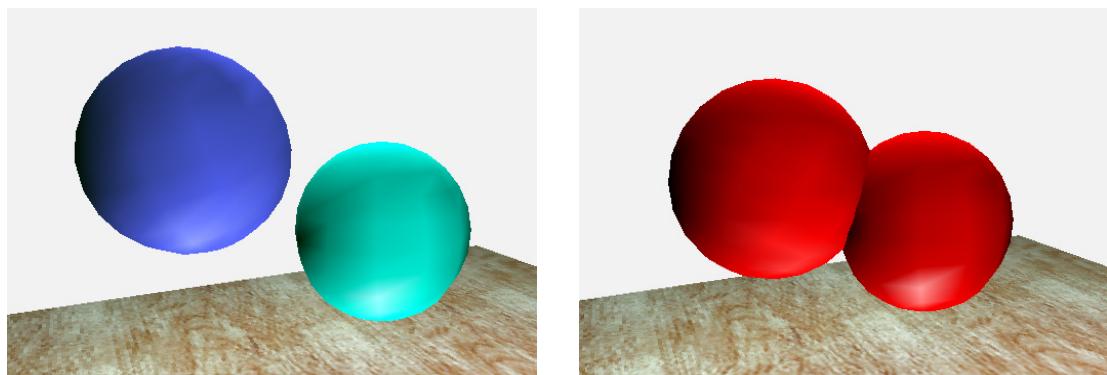


图 3-12 测试场景 1 (右图红色指示已发生碰撞)

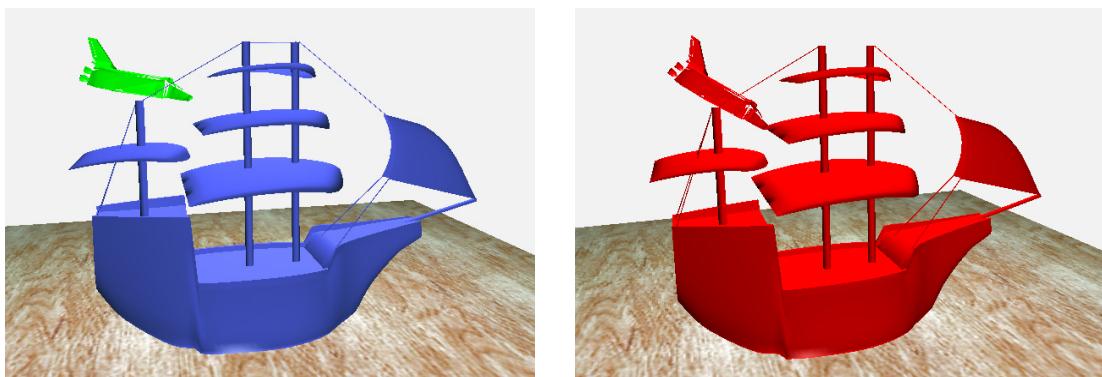


图 3-13 测试场景 2（右图红色指示已发生碰撞）

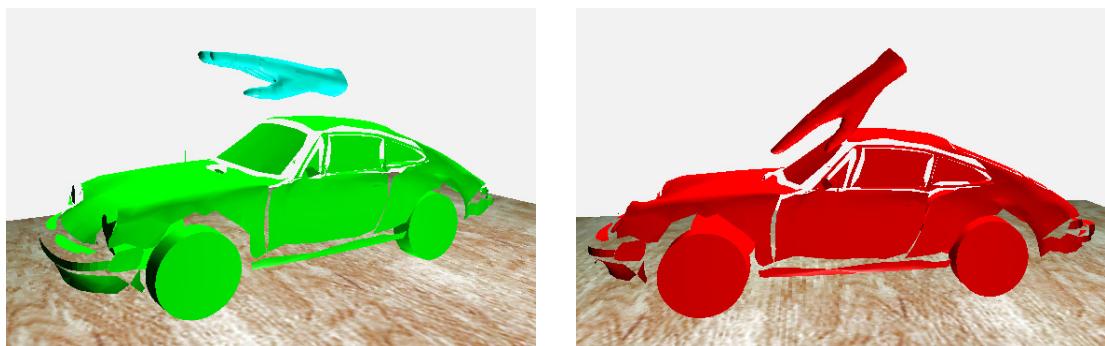


图 3-14 测试场景 3（右图红色指示已发生碰撞）

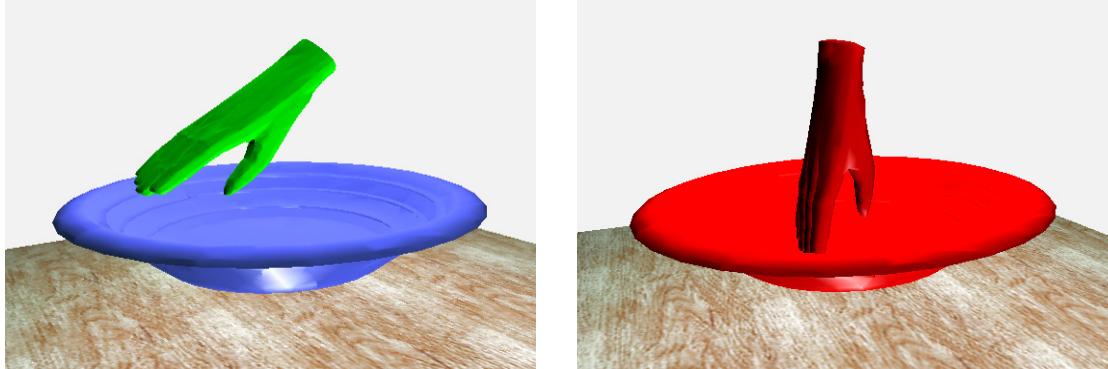


图 3-15 测试场景 4（右图红色指示已发生碰撞）

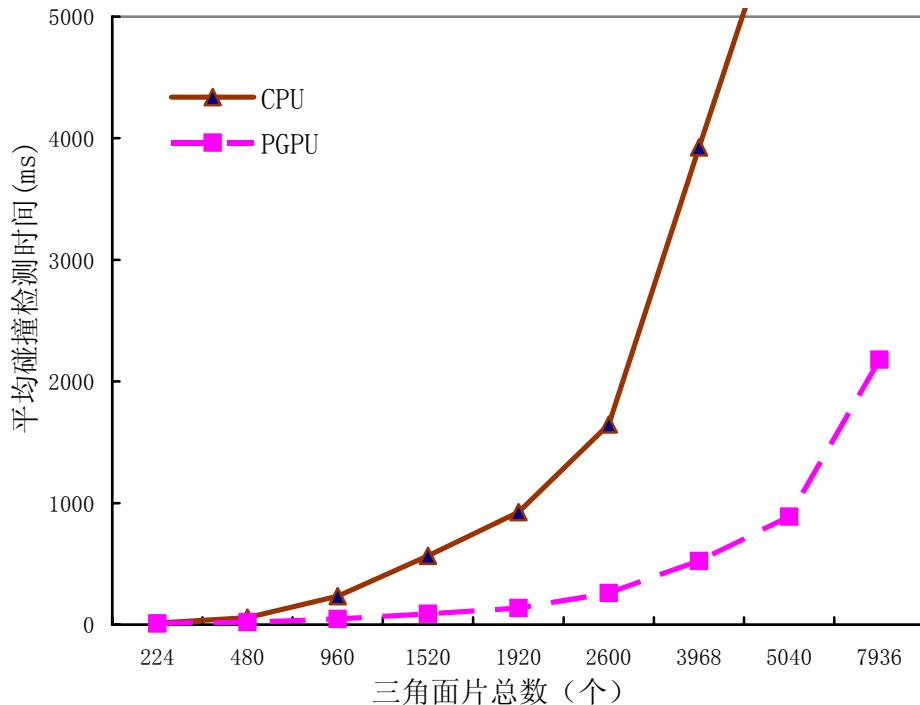


图 3-16 由 CPU 仿真实现和 PGPU 实现的基于流的碰撞检测基本算法的性能曲线

表 3-1 实验结果及比较

场景		SCD0	SCD1	SCD2	RECODE	RAPID
1	t_{cd}	6627.5	47.6	23.0	29.3	1.0
	Speedup	1	139.2	288.2	226.2	6627.5
2	t_{cd}	2256.0	53.2	40.4	Na	1.9
	Speedup	1	42.4	55.8	Na	1187.4
3	t_{cd}	15587.4	58.7	24.1	Na	161.6
	Speedup	1	265.5	646.8	Na	96.5
4	t_{cd}	9593.6	72.3	26.1	Na	10.2
	Speedup	1	132.7	367.6	Na	940.5

SCD0: 基于流的碰撞检测基本算法; SCD1: 采用了 OBB 树的基于流碰撞检测算法;
 SCD2: 优化的基于流碰撞检测算法; RECODE:[Baciu 1999] 中提出的算法;
 RAPID:[Gottschalk 1996] 中提出的算法; t_{cd} : 平均碰撞检测时间(ms); Speedup: 各种方法相对 SCD0 的加速比。

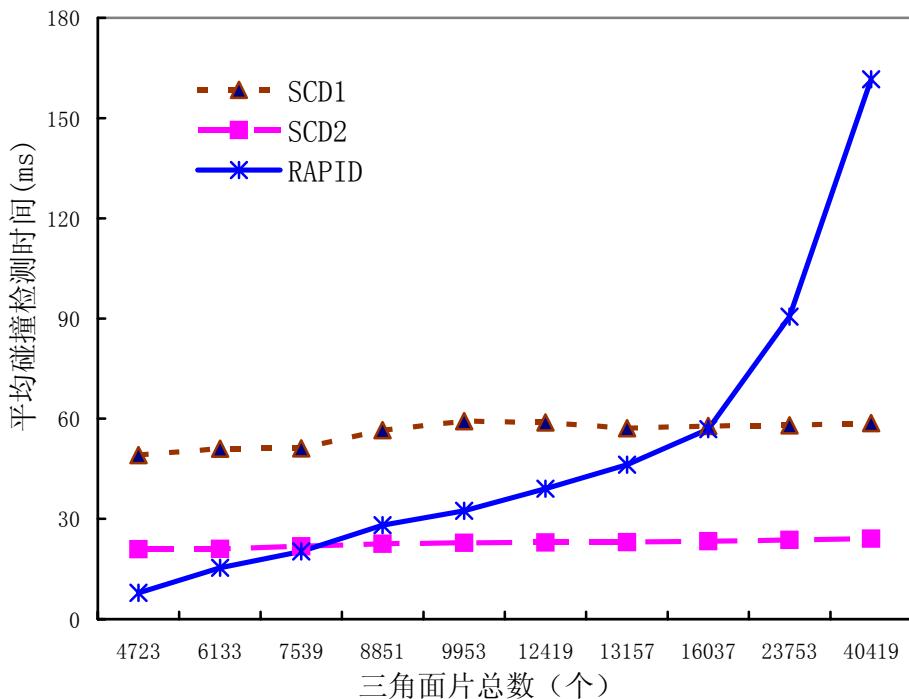


图 3-17 随场景 3 复杂度变化的算法性能曲线

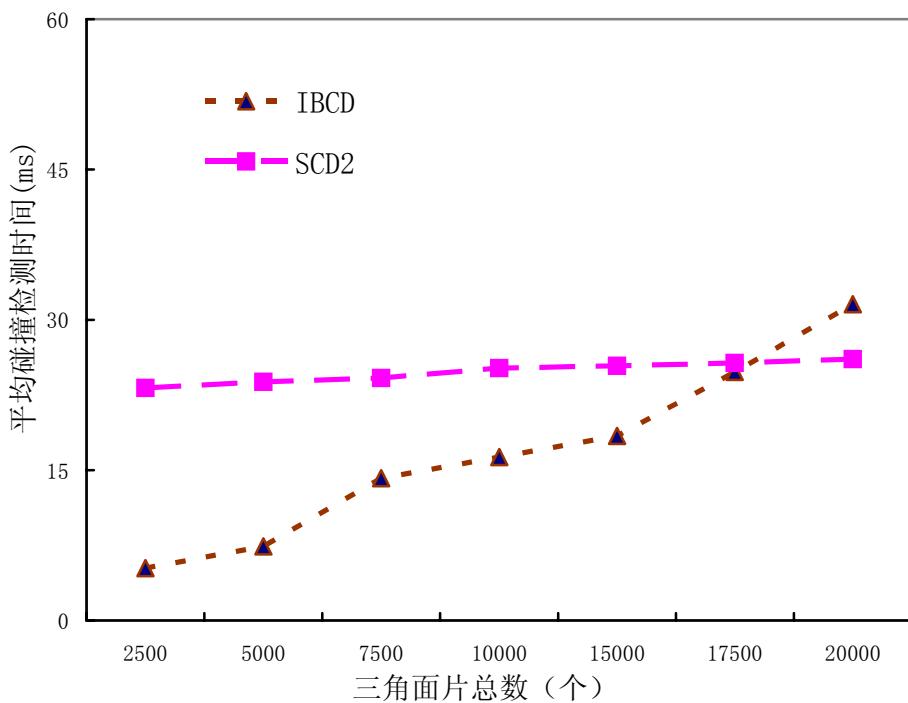


图 3-18 随场景 4 复杂度变化的算法性能曲线

3.7 小结

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。本章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 GPU 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，进一步提出了两种有效的优化方法，它们显著提高了算法的总体效率。与常规基于图象的碰撞检测算法相比，算法除了在性能上的优势之外，更具备以下两个特点：

(1) 精确性：一般基于图象的碰撞检测算法的精度取决于绘制视窗分辨率的大小，而本算法的精度由浮点数存储方式决定，与基于几何的碰撞检测算法的精度一致。

(2) 通用性：一般基于图象的碰撞检测算法通常仅能处理凸体，本算法却能够处理任意三角形网格物体。

基于流的碰撞检测算法在突破了常规基于图象的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，本算法将具有广阔的前景。

第四章 并行碰撞检测算法

摘要

本章提出了一种并行的快速碰撞检测算法。该算法在对动态复杂场景中的物体建构平衡层次包围体树的基础上，通过一种新的边包围体遍历策略有效地减少了并行碰撞检测算法并行任务个数，加大了并行的粒度，从而提高并行计算的加速比，加快了碰撞检测的速度。算法采用多线程技术实现并行计算，使其在单处理机和多处理机上均能运行，同时通过动态平衡所有 CPU 的计算负载，优化算法的整体效率。

关键词： 碰撞检测，层次包围体树，平行计算，多线程

4.1 引言

为实现实时碰撞检测，研究人员从几个方面作出了努力。首先，利用时间和空间的连贯性来对运动场景物体间进行快速的碰撞检测 [Cohen 1995][Lin 1991][Mirtich 1998]；其次，利用网络分布式计算和多处理器并行计算来提高碰撞检测的速度。王兆其等[王兆其 1998]给出了一种面向对象的碰撞检测算法，主要将其应用于分布式虚拟环境中，并讨论了在分布式环境中碰撞检测的完全性和唯一性的问题。由于以往由单机承受的繁重计算任务可通过分布式环境中多台计算机来共同完成，故算法具有较高的效率。Zachmann[Zachmann 1997]通过对 BSP 树和 K-d 树的研究提出了一种包围盒树构建方法，采用一种灵活的二分方法，所构造的包围盒树更加符合并行算法的要求。但遗憾的是，在他的算法中碰撞检测是一个递归过程，这使得并行具有一定的难度。

本章提出了一种新的并行碰撞检测算法。算法采用分治策略对任意形状的物体进行自适应的空间划分，建立其平衡包围盒树，以适应并行计算的需要。这是一种近似于二叉树的划分方法，不同之处在于划分时要求两个子包围盒内的多边形个数大致相等。在此基础之上，进一步提出了一种边包围体遍历策略用于并行遍历物体的平衡包围体树。该遍历策略能有效减少并行任务的个数，加大并行的粒度，从而提高并行计算的加速比，加快碰撞检测的速度。

本章其他各节组织如下：第二节概述并行算法的设计思想；第三节给出一种简单的并行碰撞检测算法；在第四节中，进一步提出一种基于边包围体遍历策略的并行碰撞检测算法；随后的第五节讨论算法的具体实现及实验结果；最后对本章工作进行小结。

4.2 并行算法的设计思想

随着并行处理硬件性能的迅速提高，人们对并行算法的研究也日益增加。所谓并行算法是指一次可执行多个操作的算法。对并行算法的研究现在已发展为一个独立的研究领域。很多用串行算法解决的问题已经有了相应的并行算法。本节首先介绍并行算法的一些基本概念、设计思想和算法复杂性分析，作为进一步介绍出并行碰撞检测算法的基础。

4.2.1 并行算法的定义与分类

简单地说，并行算法是适合于在各种并行计算机上求解问题和处理数据的算法[陈国良 1994]。它可定义为：

并行算法是一些可同时执行的诸进程之集合，这些进程相互作用和协调动作从而达到对给定问题的求解。

并行算法从不同角度可分类为：数值计算的和非数值计算的；同步的、异步的和分布式的； SIMD 机器上的、 MIMD 机器上的和 VLSI 模型上的，等等。

其中，同步算法（Synchronized Algorithm）是指某些进程必须等待别的进程的一类并行算法，因为一个进程的执行依赖于输入数据和系统中断，所以全部进程均必须同步于一个给定的时钟，以等待最慢的进程。异步算法（Asynchronized Algorithm）是指诸进程的执行一般不必相互等待的一类并行算法。在此情况下，进程的通信通过动态读取（修改）共享存储器的全局变量完成。分布式算法（Distributed Algorithm）是指由通信链路连接的多个场点（site）或节点（node）协同完成某项计算任务的算法。

一般而言，运行在 SIMD（单指令流多数据流）机器模型上的并行算法称为同步并行算法（Synchronized Parallel Algorithm），其中 SIMD 机器模型是指具有统一控制器多个处理器的机器模型。运行在 MIMD-SM（多指令流多数据流）机器模型上的并行算法称为异步并行算法（Asynchronized Parallel Algorithm），其中 MIMD-SM 是指每个处理器均由单独控制器控制，且拥有共享的全局存储器的机器模型。而运行在 MIMD-CL 机器模型（实质上是多计算机系统）上的算法称为分布式算法。VLSI 并行算法是指在 VLSI 计算模型上发展的一类并行算法。

4.2.2 并行计算模型

不同类型的并行算法与不同的并行计算模型相关。目前研究最多的是共享存

储器的 MIMD 模型，简记为 MIMD-SM。它是指所有处理器之间通过读/写一个全局的共享存储器来相互通信。许多关于数组、表、树和图的并行算法都可以很容易地用 MIMD-SM 模型来描述。目前一般常见的并行机大多是基于 MIMD-SM 模型的计算机，如多向量机 CRAY-2、SGI 公司的 ONYX2 等多 CPU 图形工作站。本章的并行碰撞检测算法在该模型下设计。

图 4-1 说明了 MIMD-SM 的模型结构。其中有 n 个普通的串行处理器 $CPU_0, CPU_1, \dots, CPU_{n-1}$ 共享一个全局存储器。所有处理器都可以同时从全局存储器读出信息或向全局存储器写入信息。各处理器也可以并行地执行各种算术和逻辑操作。

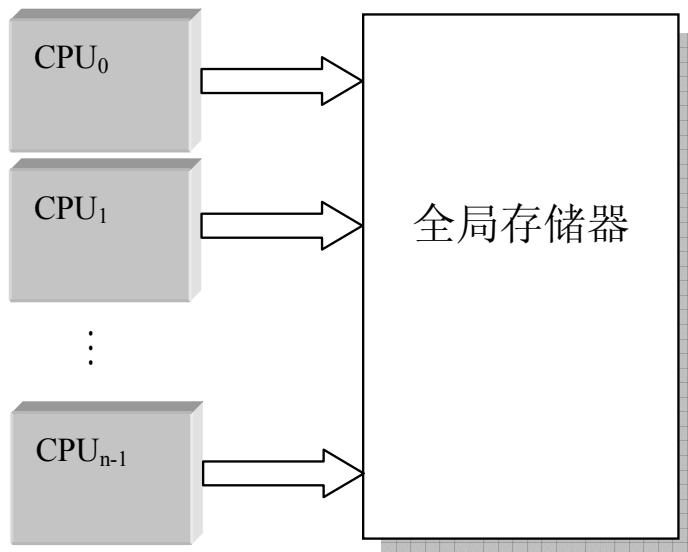


图 4-1 MIMD-SM 的并行计算模型结构

理想的 MIMD-SM 模型中，往往假设运行时间可以用算法执行的并行的存储器存取次数来衡量，这对研究并行算法性能会有很大的帮助。不过真正的并行计算机并不能做到在单位时间内对全局存储器并行地执行存取操作。而是随着并行计算机中处理器数目的增加，其对存储器进行存取操作所需的时间也相应增加。

对于可以任意方式存取数据的并行计算机来说，虽然可以证明存取操作为单位时间的假设成立，但实际中的并行计算机一般都包含一个通讯网络，用此来支持抽象的全局存储器。与算术操作等其他操作相比，通过网络之间的通讯操作速度相当慢。实际的计算机也往往与 MIMD-SM 的单位时间抽象不一致，其主要原因在于不同存储器存取模式的速度可能不同。但是，从实用角度出发，作为一种近似描述，MIMD-SM 模型的单位时间存取的假设还是可行的。

并行算法的运行时间既依赖于执行算法的处理器数目，也依赖于输入问题本

身的规模。一般来说，在分析 MIMD-SM 算法时，必须同时讨论其时间复杂度和处理器数目对算法性能的影响。这一点与串行算法主要集中于时间复杂度的分析不尽相同。处理器的数目直接与算法的加速比相关联，但并不是处理器个数越大，并行算法的加速比也越大。通常，并行算法设计中要考虑的重要问题之一是如何平衡处理器个数和算法效率之间的关系。

4.2.3 并行算法的设计

设计某个问题的并行算法一般至少有三种方法：

- (1) 检测和开拓现有串行算法中的固有并行性而直接将其并行化；
- (2) 修改已有的并行算法使其可求解另一类相似问题；
- (3) 从问题本身的描述出发，从头开始设计出一个全新的并行算法。

目前普遍使用的几种设计方法包括平衡树方法、倍增技术、分治策略、划分原理、流水线技术、加速级联策略以及破对称方法等[张德富 1992][陈国良 1994]。

4.3 简单并行碰撞检测算法

在对并行算法设计思想，尤其是分治策略进行深刻理解的基础上，结合碰撞检测算法的特点，我们采用分治策略设计并行碰撞检测算法。在分治方法中，一般通过将待处理的原问题分解成若干与原问题特性相同的子问题分而治之；若所得的子问题仍由于规模过大等原因而难以处理，则可反复使用分治策略直至将问题分解成易于求解的诸子问题为止。使用分治方法时，子问题和原问题通常类型相同，因此很自然地导致递归过程。由于碰撞检测问题的特殊性，采用分治策略进行碰撞检测所形成的递归过程从本质上讲是一个或树搜索过程，因此可用队列循环按层次遍历得到或树搜索结果，并对该层次遍历过程进行并行处理。并行算法分别对不同物体的包围盒树进行宽度优先搜索，以期尽早地排除物体间彼此不相交的部分，最终快速获得物体间是否发生碰撞的准确结果。

4.3.1 基本碰撞检测算法

在绪论中提到的碰撞检测算法一般框架中，把碰撞检测分为两个阶段、三个层次。其中局部检测阶段在整个框架中占主导地位，碰撞检测的大部分时间消耗在这个阶段，因而有必要对这部分的计算进行优化。

一般而言，对两个任意形状的多面体 A 和 B，A 和 B 发生碰撞的充要条件是

当且仅当下述三个条件至少其一成立：

- (1) 在多面体 A 中至少存在一条边与多面体 B 中的某一个面相交；
- (2) 在多面体 B 中至少存在一条边与多面体 A 中的某一个面相交；
- (3) 多面体 A 包含多面体 B 或多面体 B 包含多面体 A。

不妨假设已排除 A, B 之间彼此包含的情况，根据上述充要条件，可立即得到一个简单的基本碰撞检测算法，如图 4-2 所示。通过分析可知，若场景中面片个数为 n，则基本碰撞检测算法的算法复杂度为 $O(n^2)$ ，这显然无法适应实时碰撞检测的要求。因此引入并行计算以提高碰撞检测算法的效率，相当必要。

输入：两物体 A, B； 输出：布尔值。true 为相交，false 为不相交。

```
bool Detect_Collision_Base(A, B)
{
    for( A中的所有面片, 其中 a 为当前面片)
    {
        for ( B的各条边, 其中e为当前边 )
        {
            if (e与 a相交 )    返回发生碰撞检测真值;
        }
    }
    for( B中的所有面片, 其中 b 为当前面片)
    {
        for ( A的各条边, 其中e为当前边 )
        {
            if (e与 b相交 )    返回发生碰撞检测真值;
        }
    }
    返回不发生碰撞假值;
}
```

图 4-2 基本的碰撞检测算法的伪代码

4.3.2 建构平衡包围盒树

对待检测物体建立合理的层次包围盒二叉树是提高碰撞检测算法效率的重要手段。物体的层次包围盒树构造得是否合理会显著影响算法的效率，层次包围盒树是否平衡，即树的各节点的大小是否大致相当则直接影响并行算法的加速比。为了能够构造出大致平衡的物体包围盒树，算法采用了以下构造方法，具体分为三步：

- (1) 在物体局部坐标系中，建立整个物体的 AABB 包围盒，即平行坐标轴包围盒。该包围盒作为该物体包围盒树的根节点，包容物体中所有的多边形。

(2) 利用与局部坐标轴垂直的一个平面作为分割面将上述包围盒划分成两个子包围盒(不妨称之为左、右子包围盒),形成根节点的两个子节点。子包围盒中所包含的是物体的多边形集合。为使左、右子包围盒中的多边形个数大致相当以保证包围盒树的大致平衡,以及尽可能少地分割出需要同时放入两个包围盒中的横跨切割面的多边形(因这样的多边形需要处理两次),算法基于一个惩罚函数从三个可能的分割面中启发式地确定符合上述要求的分割面。具体做法是首先找出包围盒的三个轴向中满足式子(4-1)的 p 值点,然后将经过 p 值点,垂直 p 值点所在轴向的平面设定为切割面。

$$\min \left\{ \begin{array}{l} \min \{ f(p) \mid p_x \perp x_axis, p_x \in [x_{\min}, x_{\max}] \} \\ \min \{ f(p) \mid p_y \perp y_axis, p_y \in [y_{\min}, y_{\max}] \} \\ \min \{ f(p) \mid p_z \perp z_axis, p_z \in [z_{\min}, z_{\max}] \} \end{array} \right\} \quad (4-1)$$

这里, $x_{\min}, y_{\min}, z_{\min}$ 分别为包围盒在 x, y, z 三个坐标轴上的最小值, $x_{\max}, y_{\max}, z_{\max}$ 分别为包围盒在三个坐标轴上的最大值。 $f(p)$ 为惩罚函数,具体定义如下:

$$f(p) = |n_l - n_r| + \lambda n_c \quad (4-2)$$

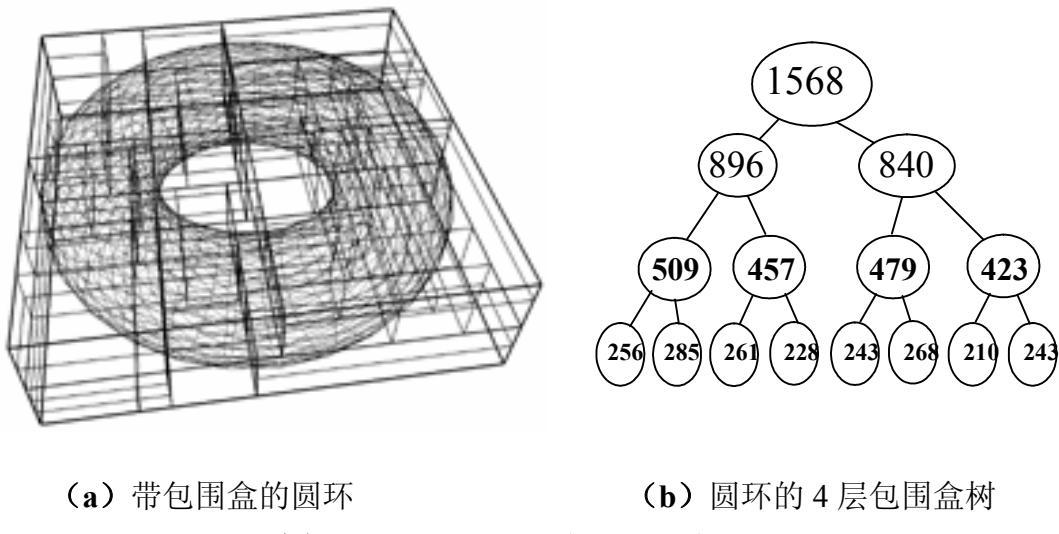
其中 p 为分割面的一个轴向的位置, n_l, n_r, n_c 分别为左包围盒内、右包围盒内和横跨分割面的多边形的个数。 λ 是 n_c 的一个影响系数。

(3) 对(2)中得到的两个子节点分别递归地执行上述包围盒的分割过程,得到最终的包围盒树。

适当的终止条件对于递归过程而言相当重要,算法为上述递归的过程设计了以下三个出口:

- 递归深度超过了预先设定的树的最大深度。树的最大深度由物体的多边形个数来确定,多边形越多,最大树深度值越大。
- 子节点所包含的多边形个数少于预先设定的叶节点所包含多边形个数的最小值。根据经验,我们设定叶节点所包含多边形个数的最小值为 10。
- 左、右子节点至少有一个包含的多边形个数接近于其父节点所包含的多边形个数。因为到这一步后再划分下去显然已失去意义。

图 4-3 给出了一个创建包围盒树的例子,其中图 4-3(a)是带有分割好包围盒的圆环,图 4-3(b)是对该圆环的包围盒递归三次后得到的 4 层平衡包围盒树。节点内的数字指是包围盒所包含的多边形的个数。



(a) 带包围盒的圆环

(b) 圆环的 4 层包围盒树

图 4-3 圆环的包围盒和包围盒树

4.3.3 简单串行碰撞检测算法

物体的层次包围盒树建好后，可通过同时遍历两个物体的包围盒树来检查两物体是否发生碰撞。然而，由于通常的树遍历算法是一个递归过程，不易直接进行并行化，因此我们首先设计了串行碰撞检测算法。为了叙述方便，我们引入任务树的概念，将同时遍历两物体包围盒树的过程定义为一棵任务树。在任务树中，每一节点为一任务，而一个任务即为两物体包围盒树中两节点之间的相交检测。

为检测两物体是否发生相交，就需要同时遍历两物体的包围盒树。在遍历过程中，若在任务节点树的一个节点中，其中一物体包围盒树的节点是非叶子节点，则只须检测包围盒树的两节点所指向的包围盒是否相交；否则，在任务节点中两节点均为两包围盒树的叶节点，则在判断出它们所指的包围盒相交后，还要判断这两包围盒内所包含的多边形是否发生碰撞。

图 4-4 为遍历过程中可能形成的任务树示例，(a) 为物体 A 的层次包围盒树，其根节点为 $BV_{A,0}^0$ ；(b) 为物体 B 的层次包围盒树，其根节点为 $BV_{B,0}^0$ ；(c) 为遍历检测两物体碰撞时形成的任务树，其中 $(BV_{A,x}^s, BV_{B,y}^t)$ 为其中的一个任务节点。任务树中每个节点的最大子任务节点数为 4，最小子任务节点数为 2。

分析该任务树，每个任务节点的子任务树之间是或的关系。于是，只要在其中一个子任务树中检测出物体发生碰撞，则两物体必发生碰撞。相反，若遍历完了整个任务树还没有出现相交情况，那么可以判定这两个物体没有发生碰撞。因此，每个任务的返回值是其所有子任务树的或，故遍历任务树的过程实质上是一个基于分治的或树搜索过程。利用任务树的概念，简单串行碰撞检测算法的伪代

码如图 4-5 所示，其中队列 LIVESET 用来存放尚未处理的任务节点。

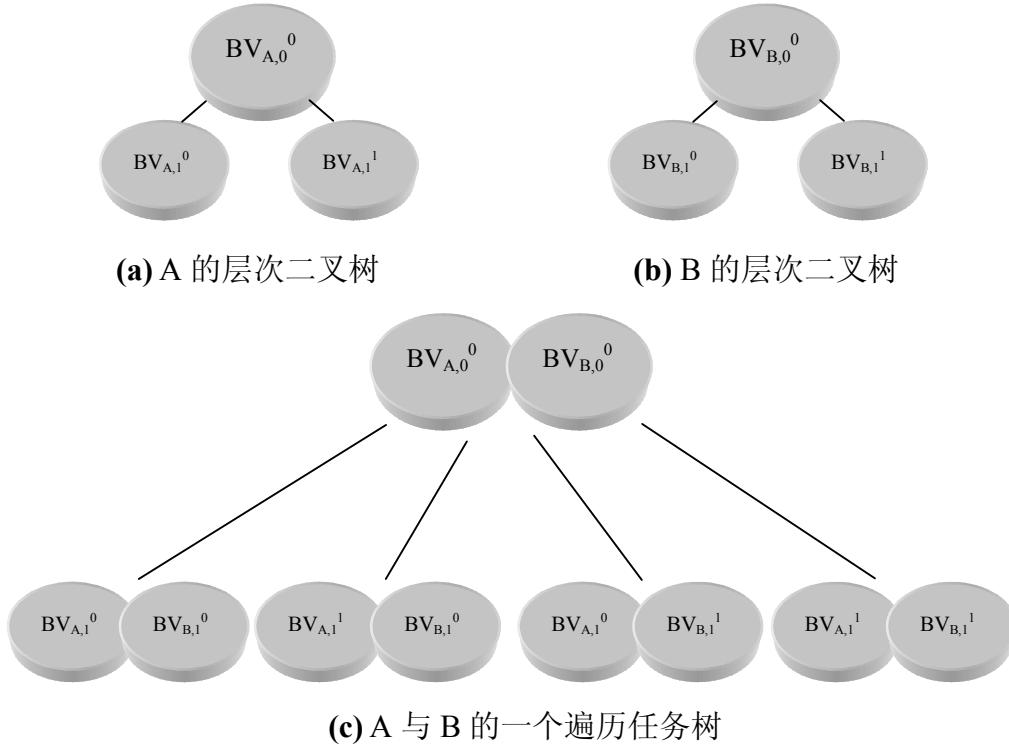


图 4-4 串行碰撞检测中遍历层次二叉树的过程

输入：两物体 A, B; 输出：碰撞检测结果。(真值为碰撞，假值为不碰撞)

```

boolean Serial_Collision_Detection_Traverse(A, B)
{
    建立 A, B 两物体的包围盒树 a,b; /*任务(a,b)作为根任务*/
    LIVESET←任务节点 (a,b); /*将根任务放入队列*/
    While(LIVESET 不为空){
        (a1,b1) ←LIVESET; /*从队列中取出一任务节点*/
        if(a1,b1 两包围盒不相交)
        {
            if(LIVESET 为空)
                直接返回假值; /*A, B 不碰*/
            }else if( a1 是叶节点) { /*a1 是叶节点*/
                if(b1 是叶节点){ /*b1 是叶节点*/
                    若对 a1,b1 用基本碰撞检测算法检测出相交, 返回真值;
                }else{ /*b1 不是叶节点*/
                    LIVESET←(a1, b1->left); /*任务 a, b->left 进入队列*/
                    LIVESET←(a1, b1->right);
                }
                else{ /*a1 不是叶节点*/
                    if(b1 是叶节点){ /*b1 是叶节点*/
                        LIVESET← (a1->left, b1);
                        LIVESET← (a1->right, b1);
                    }else{ /*b1 不是叶节点*/
                        LIVESET← (a1->left, b1->left);
                        LIVESET← (a1->left, b1->right);
                        LIVESET← (a1->right, b1->left);
                        LIVESET← (a1->right, b1->right);
                    }
                }
            }
        }
    } /* While */
    返回假值; /*A, B 未发生碰撞*/
} /* Serial_Collision_Detection_Traverse */

```

图 4-5 简单串行碰撞检测算法的伪代码

4.3.4 简单并行碰撞检测算法

按分治策略建构的包围盒树是一棵大致平衡的二叉树，这保证了各子任务的规模大致相当。加之在任务树的每一层中各子任务是相互独立的，因此将一个层次上的子任务进行并行处理是一种自然的选择。基于上述分析，在串行算法的基础上我们具体设计了一个简单的并行碰撞检测算法，如图 4-6 所示，其中仍用队列 LIVESET 来存放尚未处理的任务节点。

```

输入：两物体 A, B; 输出：碰撞检测结果。(真值为碰撞，假值为不碰撞)
boolean Parallel_Collision_Detection_Base (A,B)
{
    建立 A, B 两物体的包围盒树 a,b; /*任务(a,b)作为根任务*/
    LIVESET ← 根任务 /*根任务置入活动点队列*/
    While (LIVESET 不为空)
    {
        For (依次取出队列中的所有任务节点,并行处理)
        {
            If(当前任务检测结果是两包围盒不相交)
                中断自身任务;
            Else {
                根据任务节点的特征，若对应包围盒树中的节点不同时为叶节点，则往队列 LIVESET 中加入子任务节点(见串行碰撞检测算法)。
                反之，当对应包围盒树的节点均为叶节点，且用基本碰撞检测算法检测到发生碰撞时，则要先取消其他正并行处理的任务，然后返回真值。
            }
        }
        主过程等待所有并行处理的任务结束。
    } /* while 循环*/
    返回假值;
} /* Parallel_Collision_Detection_Base */

```

图 4-6 简单并行碰撞检测算法的伪代码

4.4 基于边包围体遍历策略的并行碰撞检测算法

上节提出的简单并行碰撞检测算法，实质上是基于包围体—包围体的遍历策略的并行算法。这里包围体泛指包括包围盒、包围球等在内的各种类型的包围体。简单的并行碰撞检测算法的问题在于其任务树过于分散，使得并行的粒度太细，并行进程之间的通讯过于频繁，影响了并行算法的加速比，从而很难得到理想的算法效率。我们认为解决这一问题的途径在于提高并行的粒度。

事实上，对于两个待检测是否发生碰撞的物体 A 与 B 而言，在建构了物体的层次包围体树之后，检测这两个物体的相交情况就可以转化为检查 A 的所有边与 B 的层次包围体树之间是否相交，以及 B 的所有边与 A 的层次包围体树是否相交的问题。

基于上述思路，我们首先设计了另一种串行的碰撞检测算法，分为以下 4 步。不失一般性，这里只考虑 A 的所有边与 B 的层次包围体树的相交检测。

- (1) 设定 B 的层次包围体树的根节点包围体为当前要处理的包围体；
- (2) 将 A 的所有边与 B 的当前包围体进行相交检测。如果不交，返回未发生碰撞检测；
- (3) 如果当前包围体节点为叶子节点，则检测 A 的所有边是否与包围体所包围的所有面相交。如果相交，则返回两物体发生碰撞；
- (4) 如果当前包围体节点为非叶子节点，分别设定其两个子节点包围体为当前包围体，递归执行步骤(2)~(4)。

我们称上述算法所采用的遍历策略为“边包围体遍历策略”。上述算法就称为基于边包围体遍历策略的串行碰撞检测算法。

用边包围体遍历策略遍历生成的层次任务树如图 4-7 所示，其中 $BV_{B, 0}^0$ 仍为 B 的包围体树的根节点， ES_A 为 A 的所有边的集合，任务节点为 $(ES_A, BV_{B, s}^t)$ ， $BV_{B, s}^t$ 为 B 包围体树上的一个节点。可以看出，采用新的遍历策略，遍历生成的层次任务树的子任务明显减少，每个任务节点的最大子任务仅为 2 个。同时由于分配每个任务节点上的计算量明显比原来任务树节点计算量要大，因此算法的并行粒度得以加大。

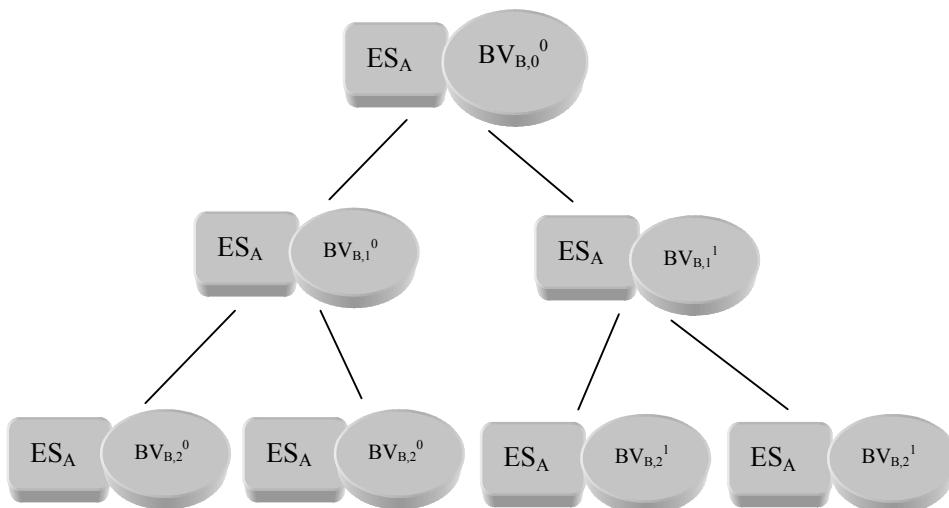


图 4-7 由边包围体遍历策略生成的层次任务树

边包围体遍历策略仍采用递归实现，为了适应并行计算，同样需要先将其转化为串行的顺序遍历算法，然后再通过串行算法设计出相应的并行算法。由于采用边包围体遍历策略的串行算法设计与简单串行算法的设计相似，本节不再重复。通过对基于边包围体遍历策略的串行算法进行并行化设计，即将遍历任务树每一层上的子任务并行处理，得到基于边包围体遍历策略的并行碰撞检测算法。图 4-8 给出了新的并行算法的伪代码，其中仍用队列 LIVESET 来存放尚未处理的任务节点。

```

输入: 两物体 A, B; 输出: 碰撞检测结果。(真值为相交, 假值为不交)
boolean Parallel_Collision_Detection_EdgeBVT_TraversalStrategy (A,B)
{
    输入 A 所有边的集合, B 物体的包围盒树 BV; /*任务(a,b)作为根任务*/
    LIVESET ← 根任务 (ESA, BVB,00); /*根任务置入活动点队列*/
    While (LIVESET 不为空)
    {
        For parallel do (依次取出队列中的前 K 个任务节点,并行处理)
        {
            If( 检测 ESA 与当前节点包围盒不发生相交)
                中断自身任务;
            Else {
                If (包围盒节点为叶子节点)
                {
                    if( 检测 ESA 与包围盒内的面片集合相交 )
                        取消其他并行任务, 然后返回真值发现碰撞;
                }else{
                    取 B 的当前包围盒节点的两个子节点与 ESA 组成两个子任务
                    加入 LIVESET 队列中;
                }
            }
        }
        主过程等待所有子任务处理结束;
        处理 LIVESET 下面 K 个子任务;
    } /* while 循环*/
    交互 A, B 位置, 重复上述过程;
    返回假值;
} /* Parallel_Collision_Detecton_EdgeBVT_TraversalStrategy */

```

图 4-8 基于边包围体遍历策略的并行碰撞检测算法的伪代码

4.5 算法实现

4.5.1 算法实现

本算法采用符合 POSIX.1c 标准的多线程技术[Lewis 1996]来实现。所谓线程就是在进程的内部执行的指令序列。多线程（multithreading）则是把一个进程分成很多可执行线程，每一个线程都独立运行。传统概念上每一个进程包含一个单线程，所以用多进程就是使用多线程。但是一个进程需要一个地址空间，创建一个新进程意味着需要创建一个新的地址空间。因此，创建一个进程是昂贵的，而在一个已经存在的进程内部创建线程则相对廉价，而且由于共享同一个地址空间，进程内部的线程间通信也较简单。多线程通过把内核级资源和用户级资源独立开来为用户提供了更多的灵活性。

总的来说，用多线程编程有如下优点：(1) 提高应用程序响应；(2) 更加有效地使用多 CPU 系统；(3) 改善程序结构；(4) 占用更少的系统资源；(5) 改善性能。

多线程之间的通信可通过共享同一地址空间来实现，并且在共享内存的多处理器上执行的一个多线程程序中，每个线程可以分别在不同的处理器上运行，因此用多线程实现并行计算简单而有效。本章采用多线程的方式来实现并行碰撞检测算法，为任务树中每个任务节点分配一个线程，由线程处理节点任务。节点任务完成后，主节点负责获取碰撞检测结果并最终返回。

一般情况下，有并发性需求的多线程应用程序不需要考虑处理器的数量。应用程序的性能在被多处理器改善的同时对用户是透明的。这也就保证了我们的并行算法不但可以在多处理机上并行执行，也可以在单处理机上并发地执行。但一个进程中可同时执行的线程个数是有限定的。本章算法在 SGI ONYX2 工作站（有 4 个 R10000 处理器）上编程实现。SGI ONYX2 工作站使用的 IRIX 操作系统缺省可同时执行的线程上限是 64 个。鉴于我们的并行碰撞检测算法中同一层子任务节点的个数可能超过这个数目，因此为保证算法的可靠性和最优性，我们在具体实现中，通过经验测试设定每次同时执行的线程个数最大为 32 个。若同层的子任务数不超过 32，则同时执行的线程数仍为该层子任务的总数。

4.5.2 实验结果

为比较各种算法的效率，我们采用三个场景对 4 种算法进行了测试。四种算

法分别是简单串行碰撞检测算法、简单的并行碰撞检测算法、基于边包围体遍历策略的串行碰撞检测算法和基于边包围体遍历策略的并行碰撞检测算法。三个测试场景分别如图 4-9、图 4-10 和图 4-11 所示。其中场景 1 中设置了 15 个物体，三角面片总数为 28,853。我们让这些物体在一个限定的空间内做随机运动，并且每一步都对所有物体两两进行碰撞检测。场景 2 和场景 3 的面片总数分别为约 10,000 和 20,000 个。在并行碰撞检测程序的运行过程中，SGI ONYX2 的 4 个 CPU 工作状态表明，4 个 CPU 一直在并行地工作，显示了良好的并行效果。对于 4 个算法测试出的具体结果见表 4-1。

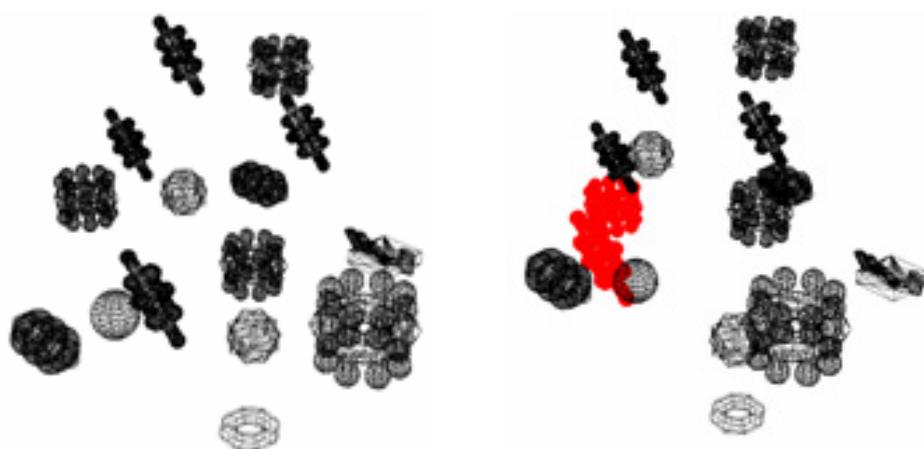


图 4-9 测试场景 1（右图红色指示已发生碰撞）



图 4-10 测试场景 2（右图红色指示已发生碰撞）

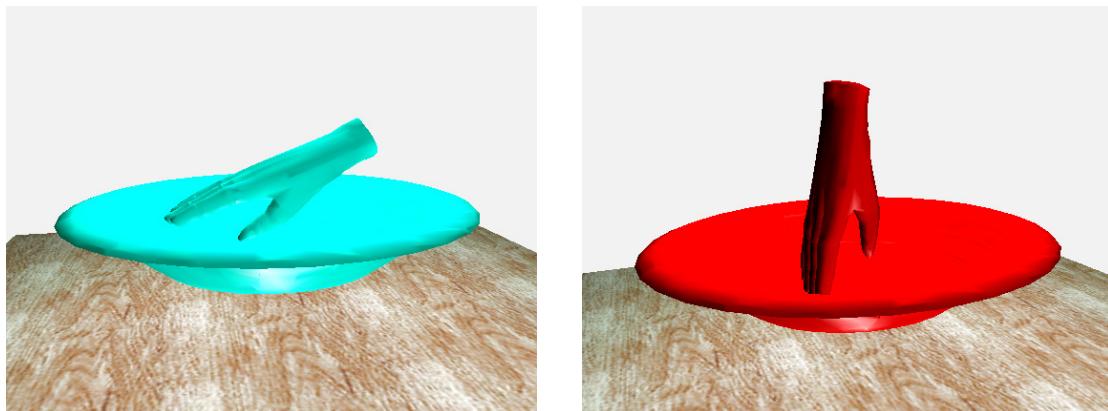


图 4-11 测试场景 3（右图红色指示已发生碰撞）

表 4-1 4 种碰撞检测算法的实验结果比较

场景		SSTCD	SPCD	ESCD	EPCD
1	t_{cd}	246.9	109.4	231.2	89.3
	Speedup	1	2.26	1	2.59
2	t_{cd}	3652.1	1443.8	349.1	125.9
	Speedup	1	2.53	1	2.77
3	t_{cd}	14832.7	7225.9	581.7	222.4
	Speedup	1	2.05	1	2.66

SSTCD: 简单串行碰撞检测算法; SPCD: 简单的并行碰撞检测算法;
 ESCD: 基于边包围体遍历策略的串行碰撞检测算法; EPCD: 基于边包围体遍历策略的并行碰撞检测算法;
 t_{cd} : 平均碰撞检测时间(ms);
 Speedup: 并行算法相对其相应串行算法的加速比。

从实验结果可以清楚看到，基于边包围体遍历策略的并行碰撞检测算法明显比简单的并行碰撞检测算法效率高，而且其相对与相应串行算法的加速比也有了部分提高，这些都表明采用基于边包围体的遍历策略有效地提高了并行碰撞检测算法的效率。

4.6 小结

本章提出了一种并行的快速碰撞检测算法。算法的主要特点表现在以下四个方面：

(1) 算法从设计开始就贯穿了并行思想，用分治策略建构物体大致平衡的包围盒树，保证了并行处理的各子问题的规模大致相等，从而能够充分利用多处理机的性能；

(2) 算法提出了一种边包围体的遍历策略，在增大每个任务计算量的同时，从总量上减少了任务树的节点个数，从而加大了并行算法的粒度，进而提高并行加速比，改善了并行碰撞检测算法的效率。

(3) 采用多线程技术实现，具有一定的通用性，在单处理机和多处理机上都能运行；

(4) 适用于对动态、复杂的场景进行实时碰撞检测。

进一步的研究工作包括：

(1) 充分并行化。目前我们仅对碰撞检测算法中的主要部分作了并行处理，进一步将对其他具有并行性的部分，如矩阵运算，包围盒之间的求交等实施并行；

(2) 采用并行虚拟机（PVM）技术或 MPI（消息传递接口）技术，使算法可利用互联网络实现网格计算。

第五章 总结与展望

5.1 全文工作总结

本文围绕实时碰撞检测问题，就如何提高复杂场景下碰撞检测的效率进行了深入研究。本文着重探讨了如何利用图形硬件的高计算性能、可编程性及多处理器计算机的并行计算能力来有效提高碰撞检测算法的效率。主要研究成果包括以下五个方面：

- 提出了一种基于流的、精确快速的碰撞检测算法。该算法探索性地将碰撞检测映射到可编程 GPU 的流计算模型上，利用可编程 GPU 的高计算性能，显著提高了碰撞检测的速度。算法首先将两物体之间的碰撞检测转化一组线段集合与三角形的求交问题，然后将线段与三角形的求交计算映射到可编程 GPU 上以有效利用图形硬件的并行架构，由实时绘制过程快速计算出相交检测结果。在对算法的时间复杂度客观分析的基础上，同时提出了两种有效的优化技术以进一步提高算法的效率。
- 突破了一般基于图形硬件的碰撞检测算法在检测精度上的局限性。虽然基于基于流的碰撞检测算法与基于图象的碰撞检测算法同样都是利用图形硬件来辅助计算，但与后者的碰撞检测精度取决于绘制视口分辨率的大小不同，基于流的碰撞检测算法通过新的流计算模式以及浮点缓存技术，令其碰撞检测精度完全由浮点数存储方式决定，与基于几何的碰撞检测算法检测精度保持一致。
- 提出了一种新的基于图象的快速碰撞检测算法。该算法利用凸表面分解技术突破了一般基于图象碰撞检测算法仅能处理凸体的局限性。同时算法结合了图形硬件的绘制加速性能、三角形带压缩编码的优化绘制方法和 OBB 包围盒的简化优势，有效地提高了复杂物体之间碰撞检测的速度。算法巧妙地利用 OBB 包围盒之间的相交检测结果来设置图形硬件绘制时所需的视域参数，同时还融入了几何网格的相邻性压缩技术来加速图形绘制的过程，进一步提高碰撞检测的效率。
- 提出了一种并行的快速碰撞检测算法。该算法在对动态复杂场景中的物体建构大致平衡的层次包围盒树的基础上，借助一种新的边包围体遍历策略并行遍历物体的包围盒树，形成更均衡、粒度更大的任务队列，以提高并行计算的加速比，更充分地利用多处理机系统的并行计算性能，从而提高了并行碰撞检测算法的效率。算法采用多线程技术实现并行计算，使其在单处理机系

统和多处理机系统上均能运行。

- 实现了以上算法，进行了相应的测试，给出了比较结果。结果表明，基于流的碰撞检测算法在算法性能上优于基于图象的碰撞检测算法，并具有更稳定的碰撞检测速度。而且，在碰撞检测的精度和适用面上，前者还取得了较显著的突破。并行碰撞检测算法由于受硬件平台的影响，碰撞检测的速度没有太多的优势，但它良好的并行策略使其具有可以发展的空间。

5.2 今后工作展望

针对现有的研究工作，我们将对其作出进一步改进和完善，同时继续探索更有效的实时碰撞检测算法。今后工作将主要集中在以下几个方面：

- 通过合理平衡 CPU 和 GPU 的计算负载，优化基于图象碰撞检测算法的性能。基于图象的碰撞检测算法利用 GPU 辅助相交检测，然而当 GPU 本身计算的负载过重时，反而会影响算法的整体速度。因此，找到有效平衡 CPU 和 GPU 计算负载的策略非常重要；
- 进一步提高基于流的碰撞检测算法的效率。通过改变可编程 GPU 的片断处理器在相交检测过程中处理的对象，如用三角形与三角形的相交计算替代边与三角形的求交计算，从而减少每次绘制的象素个数，加快相交检测中片断处理器的绘制过程，进一步提升基于流的碰撞检测算法的性能；
- 改进基于流的碰撞检测算法，使其能够有效地处理可变形物体。

参考文献

- [Agarwal 1991] Agarwal PK, Kreveld MV, Intersection queies for curved objects. In Proceedings of 7th Annual ACM Symposium on Computational Geometry, 1991:41~50.
- [Akeley 1990] Akeley K, Haeberli P, Burns D. Tomesh.c: C program on SGI developer's Toolbox CD. 1990.
- [Ati 2002] ATI, 2002. Radeon 9700. <http://www.ati.com/>
- [Baciu 1997] Baciu G, Wong WSK. Rendering in object interference detection on conventional graphics workstations. In Proceedings of the Pacific Graphics, Seoul National University, Korea, October 1997: 51-58.
- [Baciu 1999] Baciu G, Wong SKW, Sun H. RECODE: An image-based collision detection algorithm, Journal of Visualization and Computer Animation, 1999, 10(4):181~192.
- [Banerjee 1993] Banerjee R, Goel,V. and Mukherjee,A.. Efficient parallel evaluation of CSG tree using fixed number of processors. ACM Solid Modeling '93, 1993, 5: 137~146.
- [Bara 1989] Bara D. Analytical Methods for Dynamic Simulation of Non-PenetratingRigid Bodies, ACM Computer Graphics, 1989, 23(3): 223-232.
- [Bara 1990] Bara_D. Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation, ACM Computer Graphics, 1990, 24(4): 19-28.
- [Barequet 1996] Barequet G, Chazelle B, Guibas L, Mitchell J, Tal A. Boxtree: A hierarchical representation of surfaces in 3d. In Proceedings of Eurographics'96, 1996.
- [Baxter 2002] Baxter III WV, Sud A, Govindaraju NK, Manocha D. GigaWalk: interactive walkthrough of complex environment. UNC-CH Technical Report TR02-013, 2002.
- [Bergen 1997] Bergen GVD. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 1997, 2(4):1~14.
- [Bergen 1999] Bergen GVD. A fast and robust GJK implementation for collision detection of convex objects." Journal of Graphics Tools, 1999, 4(2): 7-25.
- [Bourma 1991] Bouma W, Vanecek G. Collision detection and analysis in a physically based simulation. In Proceedings Eurographics workshop on animation and simulation, 1991: 191-203.

- [Boyles] Boyles M, Fang SF, Slicing Based Volumetric Collision Detection, *Journal of Graphics Tools*, 1999, 4(4):23-32.
- [Cameron 1986] Cameron S, Culley RK. Determining the minimum translational distance between two convex polyhedra, In Proceedings of IEEE International Conference on Robotics and Automation, 1986: 591-596.
- [Cameron 1990] Cameron S. Collision detection by four-dimensional intersection testing. In Proceedings of IEEE International Conference on Robotics and Automation, 1990, 6(3): 291-302.
- [Cameron 1991] Cameron S. Approximation hierarchies and s-bounds. In the Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, 1991: 129-137.
- [Cameron 1997a] Cameron S. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation*, 1997, 13(6): 915-920.
- [Cameron 1997b] Cameron S, Enhancing GJK: Computing minimum and penetration distances between convex polyhedron, *IEEE International Conference on Robotics and Automation*, 1997.
- [Canny 1986] Canny JF. Collision detection for moving polyhedra. *IEEE Transactions on PAMI*, 1986, 8(2):200-209.
- [Chazelle 1989] Chazelle B. An Optimal algorithm for intersection three-dimensional convex polyhedra. In Proceedings of 30th annual IEEE Symposium on Foundation Computer Science, 1989: 586~591.
- [Chazelle 1997a] Chazelle B, Dobkin D, Shouraboura N, Tal A. Strategies for polyhedral surface decomposition: an experimental study. *Computational Geometry: Theory and Applications*, 7:327-342, 1997.
- [Chazelle 1997b] Chazelle B, Palios L. Decomposing the boundary of a non-convex polyhedron. *Algorithmica*, 17:245–265, 1997.
- [陈国良 1994] 陈国良,《并行算法的设计与分析》, 高等教育出版社, 1994.
- [Chung 1996] Chung K, Wang W. Quick collision detection of polytopes in virtual environments. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, Hong Kong, July 1996: 125-132.
- [Cohen 1995] Cohen J, Lin MC, Manocha D, Ponamgi M. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In Proceedings of ACM Interactive 3D Graphics Conference, Monterey, CA, USA, 1995: 189-196.
- [Cremer 1994] Cremer JF, Stewart AJ. The architecture of newton, A General Purposed Dynamic Simulator, *Proceedings of the IEEE International Conference on*

- Robotics and Automation, May 1994.
- [Dingliana 2000] Dingliana J, O'Sullivan C. Graceful degradation of collision handling in physically based animation. Computer Graphics Forum. In Proceedings of Eurographics 2000, 19(3) 239-247.
- [Dingliana 2001] Dingliana J, O'Sullivan C, Bradshaw G. Collisions and adaptive levels of detail. SIGGRAPH 2001 Sketches Program, LA, 2001.
- [Dobkin 1985] Dobkin DP, Kirkpatrick DG, A linear algorithm for determining the separation of convex polyhedra. Journal of Algorithms, 1985, 6:381~392.
- [Dobkin 1990] Dobkin DP, Kirkpatrick DG. Determining the separation of preprocessed polyhedra - A unified approach, In the Proceedings of the 17th Internat. Colloq. Automata Lang. Program., V. 443 of Lecture Notes in Computer Sciences, 1990: 400~413.
- [Ehmann 2000] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra using multi-level Voronoi marching. In Proceedings IEEE IROS, 2000.
- [Ehmann 2001] Ehmann S, Lin M C. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In Proceedings of the Eurographics Conference, Manchester, 2001: 500-510.
- [Evans 1996] Evans F, Skiena S, Varshney A. Optimizing triangle strips for fast rendering. In Proceedings of Visualization'96, pages 319-326, 1996.
- [Farouki 1989] Farouki RT, Neff CA, O'Connor M. Automatic parsing of degenerate quadric-surface intersections. ACM Transaction on Graphics, 1989, 8: 174-203.
- [Farin 1993] Farin G. Curves and surfaces for computer aided geometric design: a practical guide. Academic Press Inc. 1993.
- [Garcia 1994] Garcia-Alonso A, Serrano N, Flaquer J. Solving the Collision Detection Problem, IEEE Computer Graphics and Applications, 1994, 13 (3): 36~43.
- [Gilbert 1988] Gilbert EG, Johnson DW, Keerthi SS. A fast procedure for computing the distance between objects in three-dimensional space. IEEE Journal on Robotics and Automation, 1988, 4: 193~203.
- [Gilbert 1990] Gilbert EG, Foo CP, Computing the Distance Between General Convex Objects in 3D Space, IEEE Transactions on Robotics and Automation, 1990, 6(1): 53-61.
- [Gottschalk 1996] Gottschalk S, Lin M, Manocha D. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, the Proceedings of ACM SIGGRAPH'96, 1996: 171-180.

- [Govindar 2003] Govindaraju NK, Redon S, Lin MC, Manocha D. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Graphics Hardware*, 2003.
- [Govindar 2002] Govindaraju NK, Sud A, Yoon SE, Manocha D. Parallel occlusion culling for interactive walkthroughs using multiple GPUs. UNC Computer Science Technical Report TR02-027, 2002.
- [Gregory 1999] Gregory A, Lin M, Gottschalk S, Taylor R. H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction, In Proceedings of IEEE VR'99, 1999: 38-45.
- [Hahn 1988] Hahn JK. Realistic Animation of Rigid Bodies, *ACM Computer Graphics*, 1988, 22 (4): 299~308.
- [Hamada 1996] Hamada K, Hori K. Octree-based approach to real-time collision-free path planning for robot manipulator. In ACM96-MIE, 1996: 705~710.
- [He 1999] He T. Fast collision detection using QuOSPO trees, In Proceedings of the 1999 symposium on Interactive 3D graphics, 1999: 55-62.
- [Heidelb 2003] Heidelberger B, Teschner M, Gross M. Volumetric collision detection for deformable objects. TR395 in Computer Science Department ETH Zurich, Switzerland, April 2003.
- [Hoffmann 1989] Hoffmann CM. Geometric and Solid Modeling. Morgan Kaufmann, San Mateo, California, 1989.
- [Hoff 2001] Hoff III KE, Zaferakis A, Lin M, Manocha D. Fast and simple 2D geometric proximity queries using graphics hardware, In Proceedings of ACM Symposium on Interactive 3D Graphics, 2001:145~148.
- [Hoff 2002] Hoff III KE, Zaferakis A, Lin M, Manocha D. Fast 3D geometric proximity queries between rigid & deformable models using graphics hardware acceleration. points at which edges intersect objects. Technical Report TR02-004, Dept. of Computer Science, University of North Carolina at Chapel Hill, 2002.
- [Hubbard 1993] Hubbard PM. Interactive Collision Detection. In Proceedings of IEEE Symposium on Research Frontier in Virtual Reality, 1993.
- [Hubbard 1995] Hubbard PM. Real-time collision detection and time-critical computing. In SIVE 95, The First Workshop on Simulation and Interaction in Virtual Environments, Iowa City, Iowa. University of Iowa, informal proceedings, 1995, 1: 92~96.
- [Hudson 1997] Hudson T, Lin M, Cohen J, Gottschalk S, Manocha D. V-collide: accelerated collision detection for VRML. In Proceedings of ACM Symposium on VRML, 1997: 119-125.

- [Hughes 1996] Hughes M, Lin M, Manocha D, Dimattia C. Efficient and accurate interference detection for polynomial deformation and soft object animation. In Proceedings of Computer Animation, 1996: 155-166.
- [Isenburg 2000] Isenburg M. Triangle strip compression. In Proceedings of Graphics Interface '00 Conference, pages 197-204, 2000.
- [Johnson 1998] Johnson DE, Cohen E. A framework for efficient minimum distance computations. In Proceedings of IEEE International Conference Robotics & Automation, Leuven, Belgium, May 16-21, 1998: 3678-3684.
- [Jiménez 2001] Jiménez P, Thomas F, Torras C. Collision detection: a survey, Computers and Graphi cs, 2001, 25(2):269~285.
- [Keyser 1997] Keyser J, Krishnan S, Manocha D. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic. In ACM/Siggraph Symposium on Solid Modeling, 1997: 42-55.
- [Kim 1995] Kim HS, Ko H, Lee K, Lee C. A collision detection method for real time assembly simulation. In Proceedings of IEEE International Symposium on Assembly and Task Planning, 1995: 387~392.
- [Kim 1997] Kim HS, Ko H, Lee K, Lee C. A collision detection for interactive mechanical assembly simulation. In Proceedings of IEEE International Symposium on Assembly and Task Planning, 1997: 170~175.
- [Kim 2002] Kim YJ, Lin M, Manocha D. Fast penetration depth estimation using rasterization hardware and hierarchical refinement, In Symposium on Computational Geometry'2003, 2003: 386-387.
- [Klosowski 1996] Klosowski J, Held M, Mitchell JSB. Real time collision detection for motion simulation within complex environments. ACM SIGGRAPH Visual Proceedings, 1996: 151.
- [Klosowski 1998] Klosowski J, Held M, Mitchell JSB, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Transaction On Visualization and Computer Graphics, 1998, 4(1): 21-37.
- [Krishnan 1998] Krishnan S, Gopi M, Lin MC, Manocha D, Pattekar A. Rapid and accurate contact determination between spline models using ShellTrees. In the Proceedings of Eurographics'98, 1998.
- [Krishnan 1997] Krishnan S, Manocha D. An efficient surface intersection algorithm based on the lower dimensional formulation, ACM Transactions on Graphics, 1997, 16(1): 74-106.
- [Krishnan 1998] Krishnan S, Pattekar A, Lin MC, Manocha D. Spherical shell: A higher order bounding volume for fast proximity queries, In Proceedings of the Third International Workshop on Algorithmic Foundations of Robotics,

- 1998, 177-190.
- [Kumar 1994] Kumar V, Grama AY, Vempaty NR. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 1994, 22: 60-79.
- [Lane 1980] Lane JM, Riesenfeld RF. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1980, 2(1): 150-159.
- [Larsen 1999] Larsen E, Gottschalk S, Lin M, Manocha D. Fast proximity queries with swept sphere volumes, Technical report TR99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, 1999.
- [Larsson 2001] Larsson T, Möller TA. Collision detection for continuously deforming bodies. In *Proceedings of Eurographi cs'2001*, 2001, 325-333.
- [Latombe 1991] Latombe JC. Robot motion planning. Kluwer Academic Publishers, 1991.
- [Lewis 1996] Lewis B, Berg DJ. PthreadsPrimer - A guide to multithreaded programming, SunSoft Press, A Prentice Hall Title. 1996.
- [Lin 1991] Lin MC, Canny J. Efficient algorithms for incremental distance computation, In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991: 1008-1014.
- [Lin 1993] Lin MC. Efficient Collision detection for animation and robotics, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.
- [Lin 1994] Lin MC, Manocha D, Ponamgi K. Fast contact determination between general polyhedral models, in the *Proceedings of IEEE International Conference on Robotics and Automation*, 1994, 4: 602-208.
- [Lin 1995] Lin MC, Manocha D. Fast interference detection between geometric models, the *Visual Computer*, 11(10): pp. 542-561, 1995.
- [Lin 1998] Lin MC, Gottschalk S. Collision detection between geometric models: a survey. In the *Proceedings of IMA Conference on Mathematics of Surfaces*. 1998.
- [Lin 2002] Lin MC, Manocha D. Interactive geometric computations using graphics hardware. In *Siggraph'2002 course notes #31*, July 2002.
- [Lindholm 2001] Lindholm E, Kilgard MJ, Moreton H. A user-programmable vertex engine. In *Proceedings of SIGGRAPH*, 2001:149~158.
- [Lombardo 1999] Lombardo JC, Cani MP, Neyret F. Real-time collision detection for virtual surgery. *Computer Animation'99*, 1999: 33-39.

- [Manocha 1994] Manocha D, Demmel J. Algorithms for intersecting parametric and algebraic curves I: simple intersections, *ACM Transactions on Graphics*, 1994, 13(1): 73-100.
- [Manocha 1995] Manocha D, Demmel J. Algorithms for intersecting parametric and algebraic curves II: multiple intersections, *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, 1995: 81-100.
- [Mark 2003] Mark WR, Glanville S, Akeley K. Cg: A system for programming graphics hardware in a C-like language. In *Proceedings of SIGGRAPH'2003*, 2003, 22(3).
- [McNeely 1999] McNeely W, Puterbaugh K, Troy J. Six degree-of-freedom haptic rendering using voxel sampling. In *Proceedings of Siggraph 1999*, LosAngeles, CA.
- [McReyn 1996] McReynolds T. Programming with OpenGL: Advanced Rendering. In *SIGGRAPH'96 course notes*, 1996.
- [Mirtich 1995] Mirtich B, Canny J. Impulse-Based Simulation of Rigid Bodies, *ACM Symposium on Interactive 3D Graphics*, 1995: 181-188.
- [Megiddo 1983] Megiddo N. Linear-time algorithms for linear programming in R³ and related problems, *SIAM J. Computing*, 1983, 12: 759~776.
- [Miller 1991] Miller J, Goldman R. Combining algebraic rigor with geometric robustness for the detection and calculation of conic sections in the intersection of two quadric surfaces. In *Proceedings of ACM Solid Modeling*, 1991: 221-233.
- [Microsoft 2003] Microsoft Company. HLSL in DirectX 9.0. 2003, Website: http://msdn.microsoft.com/library/en-us/directx9_c/directx/graphics/reference/shaders/highlevelshaderlanguage.asp.
- [Mirtich 1998] Mirtich B. V-Clip: Fast and robust polyhedral collision detection, *ACM Transactions on Graphics*, 1998, 17(3): 177-208.
- [Mirtich 1995] Mirtich B, Canny J. Impulse-based simulation of rigid bodies, *ACM Symposium on Interactive 3D Graphics*, 1995: 181-188.
- [Möller 1997a] Möller T, Trumbore B. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 1997, 2(1):21~28.
- [Möller 1997b] Möller T. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 1997, 2(2):25-30.
- [Moore 1988] Moore M, Wilhelms J. Collision Detection and Response for Computer Animation, *ACM Computer Graphics*, 1988, 22 (4):289-298.

- [Myszkow 1995] Myszkowski K, Okunev OG, Kunii TL.. Fast collision detection between computer solids using rasterizing graphics hardware. *The Visual Computer*, 1995, 11:497~511.
- [Navazo 1986] Navazo I, Ayala D, Brunet P. A geometric modeler based on the exact octree representation of polyhedra. *Computer Graphics Forum*, 1986, 5(2): 91~104.
- [Naylor 1990] Naylor BF, Amanatides JA, Thibault WC. Merging BSP trees yield polyhedral modeling results. In *Proceedings of ACM SIGGRAPH*, 1990, 115-124.
- [Neider 1993] Neider J, Davis T, Woo M. *OpenGL Programming Guide*. Addison-Wesley, Menlo Park, 1993.
- [Nvidia 2003a] NVIDIA 2003, GeForce FX. <http://www.nvidia.com/>.
- [Nvidia 2003b] NVIDIA 2003, NVIDIA OpenGL Extension Specifications. http://developer.nvidia.com/object/nvidia_opengl_specs.html
- [Nvidia 2003c] NVIDIA 2003, Cg toolkit user manual. <ftp://download.nvidia.com/developer/cg>
- [Olano 1998] Olano M, Lastra A. A Shading language on graphics hardware: the PixelFlow shading system. In *Proceedings of SIGGRAPH'98*, 1998:159~168.
- [O'Sullivan 1999] O'Sullivan C, Dingliana J. Realtime collision detection and response using sphere-trees. In *Proceedings of the Spring Conference on Computer Graphics*, Bratislava, 83-92.
- [Palmer 1995] Palmer I, Grimsdale R. Collision detection for animation using Sphere-Trees. *Computer Graphics Forum*, 1995, 14(2):105–116.
- [Peercy 2000] Peercy MS, Olano M, Airey J, Ungar PJ. Interactive multi-pass programmable shading. In *Proceedings of SIGGRAPH'2000*, 2000:425~432.
- [Ponamgi 1995] Ponamgi MK, Cohen JD, Lin MC, Manocha D. Incremental algorithms for collision detection between polyhedral models. In *SIVE 95, The First Workshop on Simulation and Interaction in Virtual Environments*, 1995, 1: 84~91.
- [Poutraint 2001] Poutraint K, Contensin M. Dual Brep-CSG Collision Detection for General Polyhedra. In *Ninth Pacific Conference on Computer Graphics and Applications (PG'01)*, 2001, 124.
- [Preparata 1985] Preparata FP, Shamos MI. *Computational Geometry*, Springer-Verlag, New York.

- [Proudfoot 2001] Proudfoot K, Mark WR., Tzvetkov S, Harnrahan P. A real-time procedural shading system for programmable graphics hardware. In Proceedings of SIGGRAPH'2001, 2001:159~170.
- [Quinlan 1994] Quinlan S. Efficient distance computation between non-convex object, In Proceedings of IEEE International Conference on Robotics and Automation, 1994: 3324-3329.
- [Redon 2001] Redon S, Kheddar A, Coquillart S. CONTACT: arbitrary in-between motions for continuous collision detection. In Proceedings of IEEE ROMAN'2001, Sep. 2001.
- [Redon 2002a] Redon S, Kheddar A, Coquillart S. Fast continuous collision detection between rigid bodies. Computer Graphics Forum, 2002, 21(3): 279~287.
- [Redon 2002b] Redon S, Kheddar A, Coquillart S. Hierarchical Back-Face Culling for Collision Detection. In proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002.
- [Requicha 1992] Requicha AAG, Rossignac JR. Solid modeling and beyond. IEEE Computer Graphics and Applications, 1992: 31-44.
- [Rossignac 1992] Rossignac J, Megahed A, Schneider BO. Interactive inspection of solids: cross-section and interferences. Computer Graphics, 1992, 26(2):353~360.
- [Routh 1905] Routh EJ. Elementary Rigid Dynamics. 1905
- [Samet 1989] Samet H. Spatial data structures: quadtree, octrees and other hierarchical methods. Addison Wesley, 1989.
- [Seidel 1990] Seidel R. Linear programming and convex hulls made easy, in the Proceedings of 6th Ann. ACM Conf. On Computational Geometry, p.211-215.
- [Shene 1991] Shene C, Johnstone J. One the planar intersection of natural quadrics. In Proceedings of ACM Solid Modeling, 1991: 234-244.
- [Shinya 1991] Shinya M, Forgue M. Interference detection through rasterization. Journal of Visualization and Computer Animation, 1991, 2:131~134.
- [Smith 1995] Smith A, Kitamura Y, Takemura H, Kishino F. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In Proceedings of the IEEE Virtual Reality Annual International Symposium, 1995: 136–145.
- [Stewart 2002] Stewart N, Leach G, John S. Linear-time CSG rendering of intersected convex objects. In 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2002, (2): 437~444.

- [Su 1996] Su C, Lin F, Yen B. An adaptative bounding object based algorithm for efficient and precise collision detection of csg-represented virtual objects. In Proc. of Symposium on virtual reality in manufacturing research and education, 1996.
- [Thibault 1987] Thibault W, Naylor B. Set operations on polyhedra using binary space partitioning trees, ACM Computer Graphics, 1987, 21(4): 153-162.
- [Turnbull 1998] Turnbull C, Cameron S. Computing distances between NURBS-defined convex objects. In Proceedings of IEEE International Conference on Robotics and Automation, 1998: 3685-3690.
- [Vassilev 2001] Vassilev T, Spanlang B, Chrysanthou Y. Fast cloth animation on walking avatars. Computer Graphics Forum, 2001, 20(3):260–267.
- [Wan 2001] Wan HG, Fan ZW, Gao SM, Peng QS, A parallel collision detection algorithm based on hybrid bounding volume hierarchy, in Proceedings of CAD&Graphics'2001, Kunming, China, 2001: 521-528.
- [Wang 1987] Wang Y, Mason M. Modeling impact dynamics for robot operations, In Proceedings of the IEEE International Conference on Robotics and Automation, 1987: 678-685.
- [王兆其 1998] 王兆其, 赵沁平, 汪成为。面向对象碰撞检测方法及其在分布式虚拟环境中的应用。计算机学报, 1998, 21(11): 990~994.
- [王志强 1999] 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.
- [魏迎梅 2001] 魏迎梅, 吴泉源, 石教英. 碰撞检测中的固定方向凸包围盒的研究. 软件学报, 2001, 12(7):105 6~1063.
- [Williams 2001] Williams AL, Barrus S, Morley RK, Shirley P. An efficient and robust ray-box intersection algorithm. 2001.
Website: <http://www.cs.utah.edu/~awilliam/box/>.
- [Wilson 1999] Wilson A, Larsen E, Manocha D, Lin M. Partitioning and Handling Massive Models for Interactive Collision Detection. In the Computer Graphics Forum, September 1999.
- [Witkin 1990] Witkin A, Gleicher M, Welch W. Interactive Dynamics. ACM Computer Graphics, 1990, 24 (2): 11~22.
- [吴明华 1997] 吴明华, 余永翔, 周济。采用空间分割技术的八叉树干涉检测算法。计算机学报, 1997, 20(9): 849~854.
- [Wynn 2000] Wynn C. Opengl vertex programming on future-generation GPUs. Nvidia document, 2000.

- [Wynn 2001] Wynn C. Using p-buffers for off-screen rendering in OpenGL, NVIDIA Corporation. 2001.
http://developer.nvidia.com/docs/IO/1293/ATT/GDC01_PixelBuffers.pdf
- [Xiang 1999] Xiang X, Held M, Mitchell J. Fast and efficient stripification of polygonal surface models. In Proceedings of ACM Symposium on Interactive 3D Graphics, 1999: 71~78.
- [Zachmann 1997] Zachmann G. Real_time and exact collision detection for interactive virtual prototyping. In Proceedings of DETC'97, 1997: 1~10.
- [Zachmann 1998] Zachmann G. Rapid collision detection by dynamically aligned DOP-Trees. In Proceedings of IEEE, VRAIS'98, Atlanta, Georgia, March 1998: 90-97.
- [Zachmann 2001] Zachmann G. Optimizing the collision detection pipeline, In Proceedings of the First International Game Technology Conference (GTEC), Hong Kong, 18-21 January 2001.
- [Zeiller 1993] Zeiller M. Collision detection for objects modeled by csg. Visualization and intelligent design in engineering and architecture, 1993: 165~180.
- [张德富 1992] 张德富.《并行处理技术》, 南京大学出版社, 1992.

攻读博士学位期间发表或录用的论文情况

期刊论文

1. Zhaowei Fan, Huagen Wan, Shuming Gao. IBCD: a fast collision detection algorithm based on image space using OBB. *Journal of Visualization and Computer Animation*, 2003, 14(4): 169-181. (SCI)
2. 范昭炜, 万华根, 高曙明. 基于图象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. (EI)
3. 范昭炜, 万华根, 高曙明. 基于流的快速碰撞检测算法. *软件学报*, 2003, 已投稿.
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. 汪肇兵, 范昭炜, 朱桂林. 智能邮件系统的设计与实现. *计算机工程与应用*, 2002, 38(6): 161-163.

会议论文

1. Zhaowei Fan, Huagen Wan, Shuming Gao. Streaming collision detection using programmable graphics hardware. In *Proceedings of CAD&Graphics'2003*, Macau, Oct 2003.
2. Zhaowei Fan, Huagen Wan, Shuming Gao. IBCD: a fast collision detection algorithm based on image space using OBB, In *Proceedings of the 8th International Conference on Virtual Systems and Multimedia (VSMM'2002)*, Korea, Sep 2002: 264-272.
3. 范昭炜, 万华根, 高曙明. 基于图象的快速碰撞检测算法. In *Proceedings of Chinagraph'2002*, Beijing, China, Sep 2002.
4. Huagen Wan, Zhaowei Fan, Shuming Gao, Qunsheng Peng. A parallel collision detection algorithm based on hybrid bounding volume hierarchy. In *Proceedings of CAD&Graphics'2001*, Kunming, China, Aug 2001: 521-528. (ISTP)
5. 范昭炜, 万华根, 高曙明. 基于并行的快速碰撞检测算法. In *Proceedings of Chinagraph'2000*, Hangzhou, China, Aug 2000.

致 谢

首先要感谢我尊敬的导师高曙明研究员，在浙大五年多的学习和研究期间，高老师广博的专业知识、严谨的治学态度、敏锐的科学眼光、活跃的学术思维和高尚的人格品质激励着我不断上进。当我在学习、科研甚至是生活中遇到难题时，他的谆谆教诲和耐心的分析指点，令我端正态度，勇于直面困难。他不但是我学习与科研上的导师，也是我平时生活和做人的榜样，并将一直影响我今后的人生道路。同时还要感谢高老师夫人马景娣老师对我的关心和热心的帮助。

非常感谢我的另一位导师万华根副研究员，他细腻的观察力、活跃的思维和高效的办事能力令我深深佩服。在课题研究和论文的撰写过程中，万老师给我以细致的指导、亲力亲为的提携和无私的帮助，屡屡令我在困境中重新振作。可以说是他让我学会了如何面对科研中所遇到的困难以及如何逐步解决科研难题。能在他的指导下开始我的科研工作之路实在是我最大的幸事。

感谢叶修梓教授及其夫人给我在人生道路上的指点与启发。与君一席话，胜读十年书。叶老师渊博的知识和敏锐的洞察力令我深感佩服。感谢他在关键的时候给予我大力的支持和帮助。

感谢彭群生教授、鲍虎军研究员、潘志庚研究员、金小刚研究员、于金辉研究员和冯结青副研究员在学术上的指点和帮助。

感谢郑文庭博士、陈为博士、王章野博士、杨建博士、秦绪佳博士、纪永革博士、陈正鸣博士、张凤军博士、刘玉生博士、闫丽霞博士、刘新国博士、周昆博士和曹卫群博士，与他们一起在学术上的讨论使我深受裨益。

感谢我的同学赵友兵、林生佑、姜忠鼎、任利峰、王青、潘纲、黄益明、汪肇兵、樊海明、楼程辉、梁荣华等在平时生活和学习上的关心和帮助。

感谢实验室的各位师弟师妹周勋、李洁、马骥、李建华、朱振华、骆阳、周广平、杨文珍、杨友东、崔秀芬、李珉、唐冰等的帮助、讨论、合作与启发。

感谢实验室的工作人员为我提供了良好的学习和科研环境，他们是罗国明老师、吕思超老师、胡敏老师、金叶英女士、汤阿姨和傅师傅。

深深感谢我的父母、哥哥和嫂子，他们在关键时刻给予我无私的帮助、大力的支持和深切的关怀，令我深深感受到了亲情的温暖。谁言寸草心，报得三春晖。

特别感谢我的女友雍圆媛在生活中对我无微不致的照顾与关怀以及在科研工作上对我的鼓励与支持，她是我完成本文工作的动力源泉。

范昭炜 敬上

二零零三年十月于求是园