凸包围多面体生成算法及应用

（申请清华大学工学硕士学位论文）

培 养 单 位：软 件 学 院
学　　科：软 件 工 程
研　究　生：唐　磊
指 导 教 师：雍　俊　海　教　授

二〇一五年四月

Convex Bounding Polyhedron Construction and its Application

Thesis Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Software Engineering

by

Tang Lei

Thesis Supervisor : Professor Yong Junhai

April, 2015

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘　要

包围盒在计算机图形学和计算几何领域中应用广泛, 常用于加速几何求交、光线跟踪和碰撞检测等多种算法. 凸包围多面体是包围盒的推广, 对于一般不规则形体, 可达到比包围盒更好的紧致程度.

本文提出一种快速构造给定点集的紧致凸包围多面体的方法. 该方法首先根据点集的近似凸包, 通过 k-means 算法生成 k 个截面法向, 然后利用 GPU 沿各法向搜索切点构成截面, 最后求交构成多面体. 实验结果表明, 与同类算法相比, 该方法能够更快地构造给定点集更紧致的凸包围多面体.

[TODO] 并能有效加速碰撞检测算法.

关键词：凸包围体；近似凸包；并行计算；碰撞检测

Abstract

[TODO] An abstract of a dissertation is a summary and extraction of research work and contributions. Included in an abstract should be description of research topic and research objective, brief introduction to methodology and research process, and summarization of conclusion and contributions of the research. An abstract should be characterized by independence and clarity and carry identical information with the dissertation. It should be such that the general idea and major contributions of the dissertation are conveyed without reading the dissertation.

An abstract should be concise and to the point. It is a misunderstanding to make an abstract an outline of the dissertation and words “the first chapter”, “the second chapter” and the like should be avoided in the abstract.

Key words are terms used in a dissertation for indexing, reflecting core information of the dissertation. An abstract may contain a maximum of 5 key words, with semi-colons used in between to separate one another.

Key words: Convex bounding volume; Approximate Convex hull; Parallel computing; Collision detection

目 录

主要符号对照表

BV	包围体 （Bounding Volume）
CH	凸包 （Convex Hull）
τ	包围体的紧致程度
rtt	渲染到纹理 （Render To Texture）
k-DOP	离散方向 k 面体（k Discrete Orientation Polytope）
k-CBP	凸包围 k 面体（k Convex Bounding Polyhedron）
t	时间

第 1 章　基于 *k*-CBP 碰撞检测算法

碰撞检测算法是计算机图形学、计算机动画等领域里必不可少的。本章提出了基于 *k*-CBP 的碰撞检测算法，算法首先对输入的网格模型进行预处理，构造模型的 AABB 包围体、*k*-CBP，当进行碰撞检测时，首先判断 AABB 是否相交，若相交再进行 *k*-CBP 之间的相交测试，再次相交再进行实际模型的相交测试。再进行实际模型的相交测试时，利用了 AABB 树形结构进行判断剪枝。在凸包围 *k* 面体之间分别用 AABB 树的方式和基于 GJK 算法两种方式进行，实验结果表明本文提出的方法能够有效加速碰撞检测算法。

本章后续部分的内容组织如下：第一小节介绍 *k*-CBP 之间的相交测试算法，第二小节介绍三角网格的相交测试算法，第三节介绍总体的算法流程，最后一节为实验结果的分析。

1.1　*k*-CBP 的相交测试算法

在所有基于包围体的碰撞检测算法中，都是利用了包围体的相交测试比直接用原始模型相交测试更简单以提升算法的整体效率，包围体的相交测试是非常重要的一个步骤。与其他基于包围体的碰撞检测算法一样，本文基于 *k*-CBP 的算法也是先进行 *k*-CBP 的相交测试，若 *k*-CBP 相交，再进行原始模型的相交测试。*k*-CBP 之间的相交测试以两种方法实现，一种是将构造的 *k*-CBP 进行空间划分，构造 *k*-CBP 的 AABB 树，再基于 AABB 树进行相交测试，详细划分原则等算法见第 3.1.1 ；另一种是基于凸多面体的相交测试算法 GJK ，详细算法见第 3.1.2节。

1.1.1　基于 AABB 树的算法

AABB 包围体是碰撞检测算法过程中一种常用的包围体，构造模型的 AABB 包围体树能够有效提高模型的求交或碰撞检测过程。本文将构造得到的 *k*-CBP 视为普通的三角网格模型，采取一种自上而下的构造 AABB 包围体树的方法，顶层包围体为所有三角网格的包围体的并集，然后按照包围体跨度最大的维度进行划分成两个子节点，然后再对每个子节点进行递归划分。具体算法如算法 6 所示。

为了使生成的 AABB 包围体树更加平衡，因此划分策略为两个子节点包含相同数量的三角网格。划分轴采用沿着坐标轴方向节点跨度最大的轴进行划分，在对三角网格进行排序时，通常可选择三角网格中心位置的某轴向坐标值进行排序，因为本文的策略为平衡二叉树策略，两个孩子节点包含三角网格数量一致，因此

算法 1 AABB 树的构造

输入: 原始三角网格 $primitives$ 和下标 $first, last$
输出: AABB 树的根节点 $root$

```
1:  function ConstructAABBTree(primitives, first, last)
2:      root ← constructNode()
3:      root.primitives ← primitives
4:      for i = first → last do
5:          root.box ← root.box ∪ primitives[i].box  // 求每个三角网格的 AABB 的并集
6:      end for
7:      size ← primitives.size
8:      if size = 1 then
9:          return root
10:     end if
11:     axis ← longestAixs(root.box)  // 计算包围体跨度最大的轴
12:     sort(primitives, axis)  // 按照 axis 对 primitives 排序
13:     if size = 2 then
14:         root.left ← constructNode()
15:         root.left.primitives ← primitives[0]
16:         root.left.box ← primitives[0].box
17:         root.right ← constructNode()
18:         root.right.primitives ← primitives[1]
19:         root.right.box ← primitives[1].box
20:         return root
21:     end if
22:     half ← size/2
23:     root.left ← ConstructAABBTree(primitives, first, half)
24:      // 前一半作为其中一个叶子节点，继续递归构造
25:     root.right ← ConstructAABBTree(primitives, half + 1, size)  // 后一半继续递归构造
26:     return root
27: end function
```

该点的选择不会对总体划分产生较大影响，只会影响划分轴边缘的三角网格，因此本文仅仅简单选择三角网格第一个坐标点的轴向坐标轴进行排序。

假设算法 6 的时间复杂度为 $T(n)$，则有

$$T(n) = O(n\log n) + 2T(\frac{n}{2}), \tag{1-1}$$

公式 3-1 中 $O(n\log n)$ 为算法中根据某坐标轴排序的耗费，根据主定理得此构造包围体树的整体算法时间复杂度 $T(n) = O(n\log^2 n)$，文献 [2] 中提到可以用一种 $O(n)$ 的算法替代其中的排序操作，使得整体复杂度为 $O(n\log n)$。

生成两个 *k*-CBP 的 AABB 包围体树后，当进行碰撞检测时，将采用算法 7 的迭代算法进行遍历判断。从顶层 AABB 节点开始，若两个节点的 AABB 包围体相交，则进行深度优先遍历其孩子节点，当到达叶子节点时，再进行原生几何（本文中的 *k*-CBP 多边形网格，为了统一转化成与输入模型一致的三角网格）进行相交测试的判断，*k*-CBP 相交后，进行真实模型的相交测试也通过此方法进行。

算法 7 中，底层叶子节点包含的原始三角网格数量与树的高度相关，对于相

算法 2 基于 AABB 树碰撞检测迭代算法

输入: 两棵 AABB 树的根节点 $rootA, rootB$
输出: 是否相交

```
1: function TraverseDetection(rootA, rootB)
2:     p ← initStack(), q ← initStack()  // 初始化两个栈，用于记录待判断的 AABB 节点对
3:     p.Push(rootA), q.Push(rootB)
4:     while !p.empty()  and  !q.empty() do
5:         nodeA ← p.Pop()
6:         nodeB ← q.Pop()
7:         c ← Intersect(nodeA.box, nodeB.box)  // 判断两个节点的 AABB 包围体是否相交
8:         if c = False then
9:             Continue  // 节点 AABB 不相交，则过滤到这两个节点及其孩子节点
10:        end if
11:        if nodeA.isLeaf() then
12:            if nodeB.isLeaf() then  // 两个叶子节点的原始几何进行相交测试
13:                for all p1 ∈ nodeA.primitives do
14:                    for all p2 ∈ nodeB.primitives do
15:                        if Intersect(p1, p2) = True then
16:                            // 按照第 3.2 节中的算法进行原始三角网格相交测试
17:                            return True  // 若相交就直接返回 True
18:                        end if
19:                    end for
20:                end for
21:            else // nodeB 节点有孩子节点
22:                p.push(nodeA), q.push(nodeB.left)
23:                p.push(nodeA), q.push(nodeB.right)
24:            end if
25:        else // nodeA 节点有孩子节点
26:            if nodeB.isLeaf() then  // nodeB 是叶子节点
27:                p.push(nodeA.left), q.push(nodeB)
28:                p.push(nodeA.right), q.push(nodeB)
29:            else // nodeA 和 nodeB 都有叶子节点
30:                p.push(nodeA.left), q.push(nodeB.left)
31:                p.push(nodeA.left), q.push(nodeB.right)
32:                p.push(nodeA.right), q.push(nodeB.left)
33:                p.push(nodeA.right), q.push(nodeB.right)
34:            end if
35:        end if
36:    end while
37:    return False  // 遍历完毕也没有检测到原始三角网格相交，则返回 False
38: end function
```

同的模型，树的高度越低，叶子节点包含三角网格数量也就越多，且叶子节点测试的时间复杂度为 $O(m^2)$，m 为叶子节点包含的三角网格数量，一般而言在存储允许的情况下都尽量使得最底层叶子节点仅包含 1 个或少数几个三角形，以减少底层叶子节点三角网格两两测试的时间复杂度。

当在运动场景中的模型进行碰撞检测时，需要对 k-CBP 及模型的 AABB 包围体树进行更新，本文采用一种近似的算法进行计算，详细将在第 3.3 节中介绍。

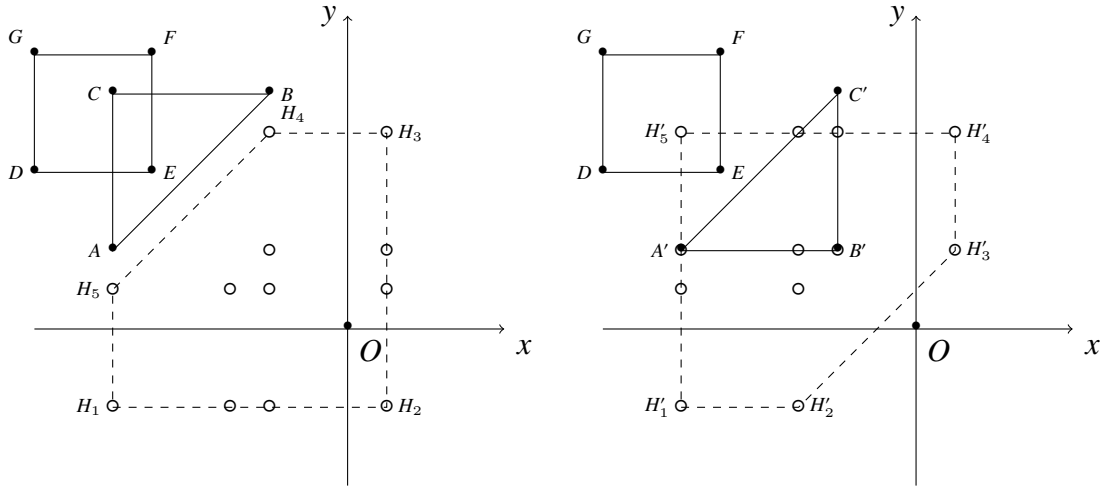
1.1.2　基于 GJK 的算法

GJK 算法是 E.G.Gilbert，D. W. Jonhson 和 S.S. Keerthi（取三位作者姓名首字母作为算法名称的缩写）等人在 1988 年发表的一种用于计算三维欧式空间中两个凸集合点之间的距离[46]，后常用于凸体之间的碰撞检测算法[47]。本文也利用此算法用于计算两个凸包围多面体 k-CBP 之间的相交测试。

GJK 算法需要用到闵科夫斯基和（Minkowski Sum）如下定义：两个点集 A 和 B，其 Minkowski 和为

$$A + B = \{a + b | a \in A, b \in B\}, \tag{1-2}$$

将相应的加号改为减号得到 Minkowski 差，即 $A - B = \{a - b | a \in A, b \in B\}$，GJK 算法的核心基础在于若两个凸体相交，则其 Minkowski 差必包含原点，因为若 A 和 B 相交即 A 和 B 必含有公共交集，即至少含有一点同时属于 A 和 B，该点的 Minkowski 差即为原点 $(0, 0)$，更加详细的证明过程可参考文献 [46,47]。下面将以一个二维例子阐述此思想。



(a) 相交　　(b) 不相交

图 1.1　二维 GJK 算法示例

如图 3.1 所示，图 3.1(a) 中，四边形 *DEFG* 四个点坐标分别为 $D(-4,2), E(-2.5,2), F(-2.5,3.5), G(-4,3.5)$，三角形 *ABC* 的坐标分别为 $A(-3,1), B(-1,3), C(-3,3)$，其 Minkowski 差如式 3-3 所示，

$$\square_{DEFG} - \triangle_{ABC} = \left\{ \begin{array}{l} (0.5,1),(-1.5,0.5),(0.5,2.5),(-1,2.5),(-3,-1),(-1,-1), \\ (0.5,0.5),(-1,1),(-3,0.5),(0.5,-1),(-1.5,-1),(-1,0.5) \end{array} \right\} \tag{1-3}$$

四边形 *DEFG* 与三角形 *ABC* 相交，该 Minkowski 差构成的多边形 $H_1H_2H_3H_4H_5$ 包含原点 ***O***。而图 3.1(b) 中，四边形 *DEFG* 坐标不变，三角形坐标变为 $A(-3,1), B(-1,1), C(-1,3)$，其 Minkowski 差如式 3-4 所示，

$$\square_{DEFG} - \triangle_{A'B'C'} = \left\{ \begin{array}{l} (0.5,1),(-1.5,1),(0.5,2.5),(-1.5,-1),(-1,2.5),(-3,2.5), \\ (-1.5,2.5),(-3,1),(-1.5,0.5),(-3,0.5),(-1,1),(-3,-1) \end{array} \right\} \tag{1-4}$$

四边形 *DEFG* 与三角形 *A′B′C′* 不相交，该 Minkowski 差构成的多边形 $H'_1H'_2H'_3H'_4H'_5$ 不包含原点 ***O***。

通过该方法可将两个凸多边形或多面体是否相交转化为其 Minkowski 差是否包含原点，假设两个凸多边形或多面体分别有 *m* 和 *n* 个点，则直接计算其 Minkowski 差的时间复杂度为 $O(m \cdot n)$。GJK 算法并不直接计算其 Minkowski 差，而是采取一种迭代的算法计算其 Minkowski 差一步一步向逼近原点，在有限步内若包含原点则原凸多边形或凸多面体相交反之不相交，而该迭代算法能在有限步内完成。

GJK 算法中定义支持映射（Support Mapping）函数为将给定的方向 ***d*** 映射为凸多边形或多面体中沿着 ***d*** 方向最远的点，该点称为支持点（Support Point），用符号 $Support(\boldsymbol{d})$ 表示，该概念与本文第 2.2节中搜索截面中的最大投影值点一致。设 $\mathbb{A}$ 和 $\mathbb{B}$ 的 Minkowski 差为 $\mathbb{D} = \mathbb{A} - \mathbb{B}$，则 $Support_{\mathbb{D}}(\boldsymbol{d}) = Support_{\mathbb{A}}(\boldsymbol{d}) - Support_{\mathbb{B}}(-\boldsymbol{d})$，这样能使得 D 围成的区域更大并尽可能的包含原点。

GJK 算法采用如下的流程进行迭代计算两个凸多面体 $\mathbb{A}$ 和 $\mathbb{B}$ 是否相交：

(1) 任选方向 ***d*** 计算 $Support_{\mathbb{D}}(\boldsymbol{d}) = Support_{\mathbb{A}}(\boldsymbol{d}) - Support_{\mathbb{B}}(-\boldsymbol{d})$，即任意从 $\mathbb{A}$ 和 $\mathbb{B}$ 的 Minkowski 差中选择一点，加入集合 ***S***；

(2) 计算 $ConvxHull(S)$ 中，具有最小二范数的点（离原点最近的点） *D*；

(3) 如果 *D* 是原点本身或 $ConvxHull(S)$ 包含原点，则表明 $\mathbb{A}$ 和 $\mathbb{B}$ 相交，停止迭代；

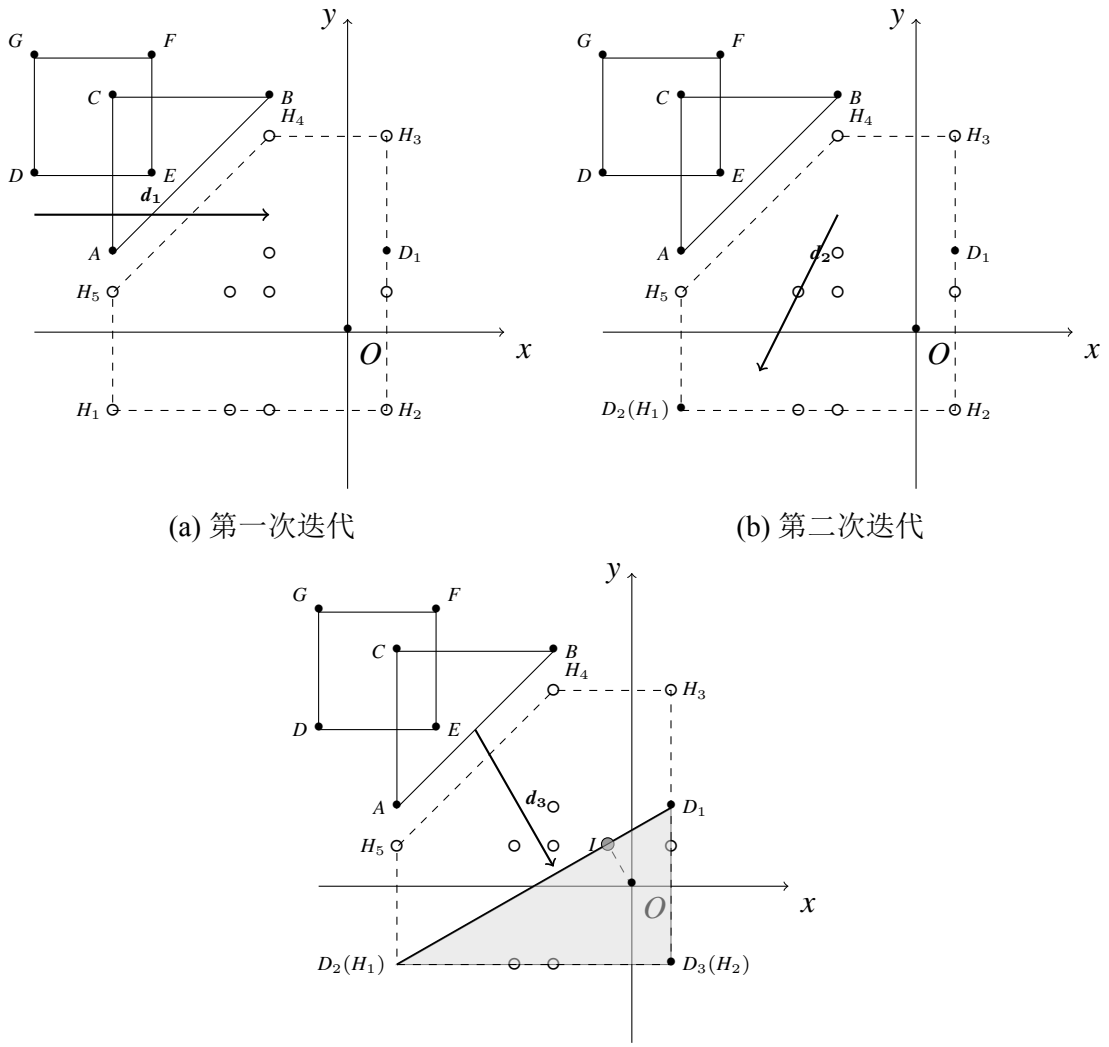
(4) 规约集合 $\boldsymbol{S}$，使得 $\boldsymbol{S}$ 中是满足 D 属于其凸包的最小的集合；

(5) 更新方向 $\boldsymbol{d} = -\boldsymbol{D}$，并计算 $Support_{\mathbb{D}}(\boldsymbol{d}) = Support_{\mathbb{A}}(\boldsymbol{d}) - Support_{\mathbb{B}}(-\boldsymbol{d})$；

(6) 如果有新的 $Support_{\mathbb{D}}(\boldsymbol{d})$ 产生，则添加至集合 $\boldsymbol{S}$ 并从第 (2) 步继续迭代，否则结束迭代，且 $\mathbb{A}$ 和 $\mathbb{B}$ 不相交。

具体而言，仍以图 3.1(a) 为例，令四边形 $DEFG$ 为 $\mathbb{A}$，三角形 ABC 为 $\mathbb{B}$，第一次迭代令 $\boldsymbol{d_1} = (1, 0)$，如图 3.2(a) 所示，

$$\begin{aligned} Support_{\mathbb{D}}(d_1) &= Support_{\mathbb{A}}(\boldsymbol{d_1}) - Support_{\mathbb{B}}(-\boldsymbol{d_1}) \\ &= E(-2.5, 2) - A(-3, 1) \\ &= D_1(0.5, 1) \end{aligned} \tag{1-5}$$



(a) 第一次迭代　　(b) 第二次迭代

(c) 第三次迭代

图 1.2　二维 GJK 算法迭代示例（相交情况）

此时，$\boldsymbol{S}=\{D_1(0.5,1)\}$，第二次迭代取 $\boldsymbol{d_2}=-\boldsymbol{D_1}=(-0.5,-1)$，

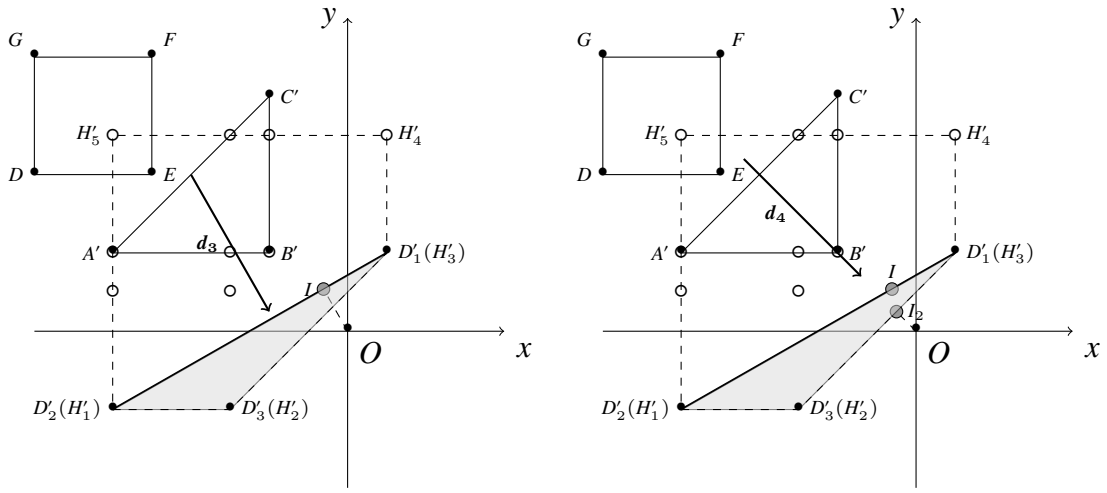
$$\begin{aligned} Support_{\mathbb{D}}(d_2) &= Support_{\mathbb{A}}(\boldsymbol{d_2})-Support_{\mathbb{B}}(-\boldsymbol{d_2}) \\ &= D(-4,2)-B(-1,3) \\ &= D_2(-3,-1) \end{aligned} \tag{1-6}$$

此时，$\boldsymbol{S}=\{D_1(-0.5,1),D_2(-3,-1)\}$，第三次迭代，计算 $ConvexHull(S)$ 中最小二范数的点为 $I(-4/13,7/13)$，并令 $\boldsymbol{d_3}=-\boldsymbol{I}=(4/13,-7/13)$，

$$\begin{aligned} Support_{\mathbb{D}}(d_3) &= Support_{\mathbb{A}}(\boldsymbol{d_3})-Support_{\mathbb{B}}(-\boldsymbol{d_3}) \\ &= E(-2.5,2)-C(-3,3) \\ &= D_3(0.5,-1) \end{aligned} \tag{1-7}$$

此时 $\boldsymbol{S}=\{D_1(0.5,1),D_2(-3,-1),D_3(0.5,-1)\}$，如图 3.2(c) 所示，$ConvexHull(S)$ 为阴影部分包含原点，结束迭代，$\mathbb{A}$ 和 $\mathbb{B}$ 相交。

再以图 3.1(b) 为例，设三角形 $\mathbb{B}'=A'B'C'$，该例前两次迭代与上例相同，此处省略直接从第三次迭代起，



(a) 第三次迭代　　(b) 第四次迭代

图 1.3　二维 GJK 算法迭代示例（不相交情况）

第三次迭代，$\boldsymbol{S}=\{(D'_1,D'_2)\}$，此时 $\boldsymbol{d_3}=-\boldsymbol{I}=(4/13,-7/13)$，

$$\begin{aligned} Support_{\mathbb{D}}(d_3) &= Support_{\mathbb{A}}(\boldsymbol{d_3})-Support_{\mathbb{B}'}(-\boldsymbol{d_3}) \\ &= E(-2.5,2)-C'(-1,3) \\ &= D'_3(-1.5,-1) \end{aligned} \tag{1-8}$$

此时 $\boldsymbol{S}=\{D'_1(0.5,1),D'_2(-3,-1),D_3(-1.5,-1)\}$，如图 3.3(a) 所示，$ConvexHull(S)$ 并不包含原点，继续迭代，

计算 $\boldsymbol{S}$ 中二范数最小的点为 $I_2(-1/4,1/4)$，并规约集合 $\boldsymbol{S}$，点 D_2' 可去掉，变为 $\boldsymbol{S}=\{(D_1'(0.5,1),D_3'(-1.5,-1)\}$，并另 $\boldsymbol{d}_4=-I_2=(1/4,-1/4)$，

$$\begin{aligned} Support_{\mathbb{D}}(d_4) &= Support_{\mathbb{A}}(\boldsymbol{d_4})-Support_{\mathbb{B}'}(-\boldsymbol{d_4}) \\ &= E(-2.5,2)-C'(-1,3)\ \textbf{or}\ A'(-3,1) \\ &= D_3'(-1.5,-1)\ \textbf{or}\ D_1'(0.5,1) \end{aligned} \tag{1-9}$$

此时，没有新的 $Support_{\mathbb{D}}(\boldsymbol{d})$ 产生，结束迭代，$\mathbb{A}$ 和 $\mathbb{B}$ 不相交。

在实际计算过程中，并不需要计算离原点最近（二范数最小）的点的坐标，只需选择垂直已有方向再进行判断选择方向即可，以图 3.2(c) 为例，此时 $\overrightarrow{D_1D_2}=D_2(-3,-1)-D_1(0.5,1)=(-3.5,-2)$，此时只需令 $\boldsymbol{d_3'}=D_1D_2\times D_1O\times D_1D_2=-(D_1D_2\cdot D_1O)D_1D_2+(D_1D_2\cdot D_1D_2)D_1O=(5,-8.75)$（$\boldsymbol{a}\times\boldsymbol{b}\times\boldsymbol{c}=-(\boldsymbol{c}\cdot\boldsymbol{b})\boldsymbol{a}+(\boldsymbol{c}\cdot\boldsymbol{a})\boldsymbol{b}$），

$$\begin{aligned} Support_{\mathbb{D}}(d_3') &= Support_{\mathbb{A}}(\boldsymbol{d_3'})-Support_{\mathbb{B}'}(-\boldsymbol{d_3'}) \\ &= E(-2.5,2)-C(-3,3) \\ &= D_3(0.5,-1) \end{aligned} \tag{1-10}$$

得到的 $Support_{\mathbb{D}}(d_3')$ 仍为 $D_3(0.5,-1)$，与选择二范数最小点 I 所代表的方向达到相同的结果，因为选择二范数最小点的坐标代表方向与该垂直方向相同，因此结果一致。

(1) 通过 $\boldsymbol{d_3'}$ 得到的支持点 $\boldsymbol{D_3}$ 中，$\boldsymbol{d_3'}\cdot OD_3=(5,-8.75)\cdot(0.5,-1)=11.25>0$ 表明通过 $\boldsymbol{d_3'}$ 方向到支持点的方向覆盖了原点，即原点在线段 D_1D_2 线段的右下方；

(2) $\overrightarrow{D_3D_1}$ 垂直并指向阴影部分面积方向为 $\overrightarrow{P_{D3D1}}=\overrightarrow{D_3D_1}\times\overrightarrow{D_3D_2}\times\overrightarrow{D_3D_1}=(-14,0)$，此时，$\overrightarrow{P_{D3D1}}\cdot D_3O=(-14,0)\cdot(-0.5,1)=7<0$ 表明原点 $\boldsymbol{O}$ 在 $\overrightarrow{P_{D3D1}}$ 的正方向即线段 $D3D1$ 的左边；

(3) 同时，$\overrightarrow{D_3D_2}$ 垂直并指向阴影部分面积方向为 $\overrightarrow{P_{D3D2}}=\overrightarrow{D_3D_2}\times\overrightarrow{D_3D_1}\times\overrightarrow{D_3D_2}=(0,24.5)$ 且 $\overrightarrow{P_{D3D1}}\cdot D_3O=(0,24.5)\cdot(-0.5,1)=24.5>0$ 表明原点 $\boldsymbol{O}$ 在 $\overrightarrow{P_{D3D2}}$ 的正方向即线段 $D3D2$ 的上边；

以上三个条件全部满足，即原点 $\boldsymbol{O}$ 被阴影部分所覆盖，即 $\mathbb{A}$ 和 $\mathbb{B}$ 的 Minkowski 差包含原点即可确定 $\mathbb{A}$ 和 $\mathbb{B}$ 相交，结束迭代。

相应的以图 3.3(a) 为例，其结果仍一样。

$$\begin{aligned} Support_{\mathbb{D}}(d_3') &= Support_{\mathbb{A}}(\boldsymbol{d_3'})-Support_{\mathbb{B}'}(-\boldsymbol{d_3'}) \\ &= E(-2.5,2)-C'(-1,3) \\ &= D_3'(-1.5,-1) \end{aligned} \tag{1-11}$$

图 3.3(a) 中，上一步通过 $\boldsymbol{d}'_3$ 得到的支持点为 $D'_3(-1.5,-1)$，且 $\overrightarrow{D'_3D'_1}$ 垂直并指向阴影部分面积方向为 $\overrightarrow{P_{D'_3D'_1}}\times\overrightarrow{D'_3D'_2}\times\overrightarrow{D'_3D'_1}=(-2,-2)*(-3)+8*(-1.5,0)=(-6,6)$，此时 $\overrightarrow{P_{D3'D1'}}\cdot D'_3O=(-6,6)\cdot(1.5,1)=-3<0$ 表明下一次迭代方向需要以 $\overrightarrow{P_{D3'D1'}}$ 的反方向开始，此时点 D'_2 可以规约掉，即 $\boldsymbol{d}'_4=(6,-6)$，以此方向得到的支持点 $D'_1(0.5,1)$，此时，$\boldsymbol{d}'_4\cdot OD_1=(6,-6)\cdot(0.5,1)=-3<0$ 表明通过 $\boldsymbol{d}'_4$ 方向到支持点覆盖的区域没有包含原点，即可排除相交。

具体的算法如算法 8 所示，文献 [47] 证明了集合 $\boldsymbol{S}$ 中在三维空间最多只包含 4 个点，且算法最终会收敛。在实际实现过程中可以根据需求设定最大迭代次数，达到后或者认为没找到相交点或者采取保守态度进行下一步验证。

算法 3 基于 GJK 的 k CBP 相交检测算法

输入: k-CBP $k\text{-}CBP_1, k\text{-}CBP_2$
输出: 是否相交

```
1:  function KCBPDetectionBasedOnGJK(k-CBP_1, k-CBP_2)
2:      d ← initNormal()
3:      D ← Support(k-CBP_1, k-CBP_2, d)
4:      S ← {p}
5:      iter ← 1, d ← −d
6:      while inter++ < MaxIter do
7:          D ← Support(k-CBP_1, k-CBP_2, d)
8:          if D · d < 0 then
9:              return False
10:         end if
11:         S ← S ∪ D
12:         contains ← CheckContainUpdate(S, d) // 检测是否包含原点，对集合 S 进行规约，
        并获取下一次迭代的方向 d
13:         if contains then
14:             return True // 包含原点，直接返回相交，否则继续迭代
15:         end if
16:     end while
17:     return False // 达到最大迭代次数，根据需求返回相交或者不相交
18: end function
```

算法 8 第 12 行 *CheckContainUpdate* 子过称需检测 $\boldsymbol{S}$ 是否包含原点，必要时进行规约并获取下一次迭代方向，以上两个例子已经对二维情况进行分析，具体三维实现可以参考文献 [47]。

1.2　三角网格的相交测试算法

所有针对三角网格模型的碰撞检测算法最终都离不开三角网格的相交测试，不管模型是用何种包围体采用几何或者代数的方法都需要进行三角网格的相交测试。本文将利用一种几何代数相结合的方法进行三角网格之间的相交测试。具体

而言，假设两个不同面的三角形所在平面交于直线 $\boldsymbol{L}$，两个三角形位置关系分为相交或者不相交两种情况，如图 3.4所示。

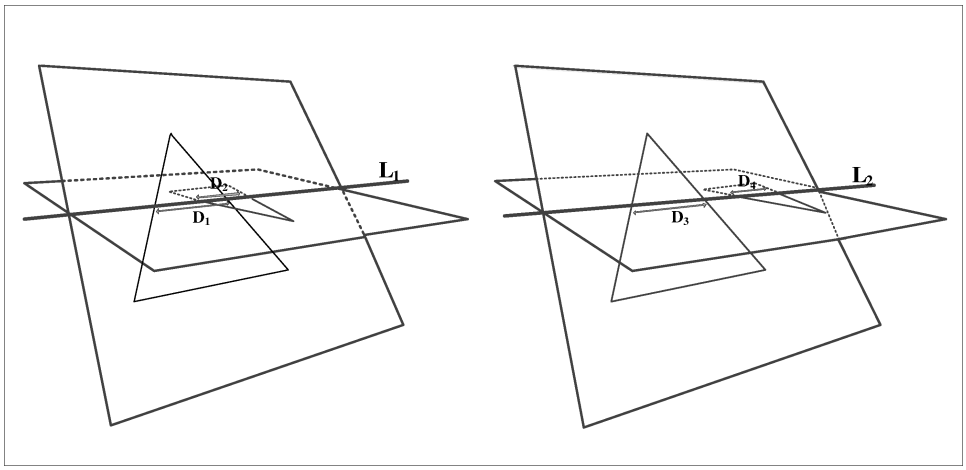


图 1.4　空间中两个非共面三角形的位置关系[62]

图 3.4左图中，两个三角形所在平面交于直线 $\boldsymbol{L}_1$ 且与三角形公共部分为线段 $\boldsymbol{D}_1$，另一个三角形与 $\boldsymbol{L}_1$ 交于线段 $\boldsymbol{D}_2$，相应的右图两个三角形分别交公共交线 $\boldsymbol{L}_2$ 于 $\boldsymbol{D}_3$ 和 $\boldsymbol{D}_4$，其中 $\boldsymbol{D}_1$ 和 $\boldsymbol{D}_2$ 相交可以推出两个三角形相交，反之 $\boldsymbol{D}_3$ 和 $\boldsymbol{D}_4$ 不相交因此三角形不相交，因此只需要判断两个三角形与平面交线的线段是否相交即可[62]，以下方法都基于此结论。

假设两个三角形 $T_1(\boldsymbol{U}_1,\boldsymbol{U}_2,\boldsymbol{U}_3), T_2(\boldsymbol{V}_1,\boldsymbol{V}_2,\boldsymbol{V}_3)$ 所在平面分别为 Π_1,Π_2，假设平面方程

$$\Pi_1 = \boldsymbol{n}_1 \cdot \boldsymbol{x} + d_1 = (U_2 - U_1) \times (U_3 - U_1) \cdot \boldsymbol{x} + d_1, \tag{1-12}$$

其中将任意一点 $U_i, i \in \{1,2,3\}$ 带入公式 3-12 得 d_1，同理得到 Π_2。

然后将 T_2 三个顶点带入方程 3-12 得到 T_2 到 Π_1 的有向距离 $\boldsymbol{l}_{1i}, i \in \{1,2,3\}$，根据 $\boldsymbol{l}_{1i}$ 的值共下面三种情况：

(1) 若 $\forall i \in \{1,2,3\}, l_{1i} = 0$，即 T_2 的三个顶点到 Π_1 的距离都为 0，则两个三角形共面；

(2) 若 $\forall i \in \{1,2,3\}, l_{1i} > 0$ 或 $\forall i \in \{1,2,3\}, l_{1i} < 0$，即有向距离同号，则 T_2 在 Π_1 的同侧，可立即排除相交；

(3) 其他情况，T_2 必交 Π_1 于一条线段。

同理可以根据 T_1 到 Π_2 得到类似的情况。针对情况（1），共面的两个三角形求交可以通过两个三角形中三条线段两两判定是否相交最多 9 次线段线段求交判定可得，或者通过我们在文献[63] 中提出的方法进行，该方法对线段三角形的位置做了详细的分类可通过不超过 6 次线段线段求交判定；针对情况（2），计算

出有向距离同号后即可排除相交立即返回；针对情况（3），如图 3.5 所示，不妨设 T_1 中点 $\boldsymbol{V}_1, \boldsymbol{V}_2$ 在平面 Π_2 的一侧，$\boldsymbol{V}_3$ 在另外一侧，且点 $\boldsymbol{V}_1, \boldsymbol{V}_2$ 在平面上投影点分别为 $\boldsymbol{K}_1, \boldsymbol{K}_2$，线段 $\boldsymbol{V}_1\boldsymbol{V}_2, \boldsymbol{V}_2\boldsymbol{V}_3$ 与 $\boldsymbol{L}$ 分别交于点 $\boldsymbol{I}_1, \boldsymbol{I}_2$，点 $\boldsymbol{V}_1, \boldsymbol{V}_2$ 向直线 $\boldsymbol{L}$ 的投影点分别为 $\boldsymbol{P}_1, \boldsymbol{P}_2$，三角形 T_1 交于直线 $\boldsymbol{L}$ 与线段 $\overline{I_1 I_2}$。

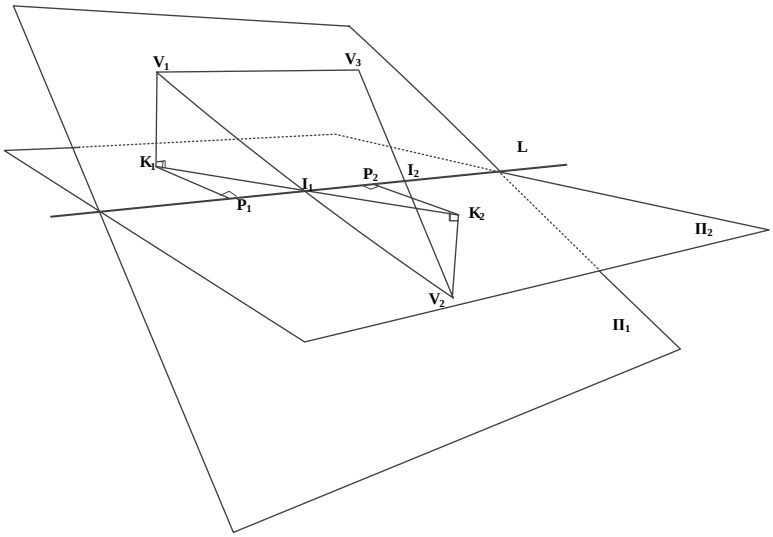


图 1.5　求非共面三角形区间线段示意图[62]

已知直线 $\boldsymbol{L}$ 的方向为 $\boldsymbol{n}_L = \boldsymbol{n}_1 \times \boldsymbol{n}_2$，设 $\boldsymbol{O}$ 为 $\boldsymbol{L}$ 上一点，则直线的参数方程为 $\boldsymbol{L} = t \cdot \boldsymbol{n}_L + \boldsymbol{O}$，假设交点 $I_1 = t_1 \cdot \boldsymbol{n}_L + \boldsymbol{O}$，投影点满足 $p_1 = \boldsymbol{n}_L \cdot (\boldsymbol{V}_1 - \boldsymbol{O}), p_2 = \boldsymbol{n}_L \cdot (\boldsymbol{V}_2 - \boldsymbol{O})$，由图 3.5 可知，$\triangle K_1 P_1 I_1 \sim \triangle K_2 P_2 I_1, \triangle V_1 K_1 I_1 \sim \triangle V_2 K_2 I_1$，可得

$$\frac{t_1 - p_1}{p_2 - p_1} = \frac{l_{11}}{l_{11} - l_{12}} \Rightarrow t_1 = (p_2 - p_1)\frac{l_{11}}{l_{11} - l_{12}} + p_1 \tag{1-13}$$

同理可得 $\boldsymbol{I}_2$ 的参数 t_2，三角形 T_2 用相同的方法也能得到区间的参数，即可判断两个区间线段是否相交，进而可得该非共面的两个三角形是否相交。完整的算法如 9 所示。

算法 9 第 10 行在进行边边测试子过程中，可通过点与有向线段的位置关系确定，如线段两个端点都在另外一个有向线段的一边说明两线段不相交，反之相交，不需要求解出实际的交点。在实际实现过程中，往往需要引入容差以提高算法的稳定性。文献 [62] 介绍了更多的优化技巧。

1.3　基于 k-CBP 的碰撞检测算法

凸包围多面体可应用于加速相关几何算法的整体效率, 图 3.6 为利用 Bunny 模型进行碰撞检测的示例, 图中模型 1 与 2、2 与 3 的包围盒分别相交, 而其 16-

算法 4 三角形求交算法

输入: 两个三角形的 6 个顶点 $T_1(\boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3), T_2(\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3)$
输出: 是否相交

function TriangleTriangleDetection($\boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3, \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$)
    $\Pi_1 \leftarrow \boldsymbol{n}_1 \cdot \boldsymbol{x} + d_1$ // 按照公式 3-12 计算 T_1 所在平面方程
    for $i = 1 \rightarrow 3$ **do**
        $l_{1i} \leftarrow \boldsymbol{n}_1 \cdot \boldsymbol{V}_i + d_1$ // 计算 T_2 到 Π_1 的有向距离
    end for
    if ($l_{11} > 0$ **and** $l_{12} > 0$ **and** $l_{13} > 0$)**or**($l_{11} < 0$ **and** $l_{12} < 0$ **and** $l_{13} < 0$) **then**
        return False // T_2 在 Π_1 的同侧，排除
    end if
    if $l_{11} = 0$ **and** $l_{12} = 0$ **and** $l_{13} = 0$ **then**
        return EdgeEdgeTest($\boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3, \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$)
        // T_2 与 T_1 的共面，普通的边边相交测试
    end if
    $\Pi_2 \leftarrow \boldsymbol{n}_2 \cdot \boldsymbol{x} + d_2$ // 按照公式 3-12 计算 T_2 所在平面方程
    for $i = 1 \rightarrow 3$ **do**
        $l_{2i} \leftarrow \boldsymbol{n}_2 \cdot \boldsymbol{U}_i + d_2$ // 计算 T_1 到 Π_2 的有向距离
    end for
    if ($l_{21} > 0$ **and** $l_{22} > 0$ **and** $l_{23} > 0$)**or**($l_{21} < 0$ **and** $l_{22} < 0$ **and** $l_{23} < 0$) **then**
        return False // T_1 在 Π_2 的同侧，排除
    end if
    $t_1 \leftarrow$ CalParam, $t_2 \leftarrow$ CalParam() // 按照公式 3-13 计算线段 $\boldsymbol{D}_1$ 的参数区间
    $t_3 \leftarrow$ CalParam, $t_4 \leftarrow$ CalParam() // 类似的方法计算线段 $\boldsymbol{D}_2$ 的参数区间
    if Overlap(t_1, t_2, t_3, t_4) **then**
        return True // 区间交叉，表明三角形相交返回 **True**
    else
        return False
    end if
end function

CBP 仅 1 与 2 相交, 实际模型仅 1 与 2 相交. 用 16-CBP 可排除模型 2 与 3 之间的碰撞检测, 而仅用包围盒算法则无法排除, 显然检测模型 2 与 3 的 16-CBP 是否相交比直接通过检测模型 2 与 3 是否相交更省时间.

1.3.1　静止场景中的碰撞检测算法

模型的 k-CBP 相交后，会用模型的 AABB 树进一步对模型进行碰撞检测，模型的 AABB 树构造方法如第 3.1.1 节所述，图 3.7 是按照本文所采用的构造方法针对 Bunny 模型构造的 AABB 树形结构的顶上 4 层。

整体的碰撞检测算法流程图如图 3.8所示，首先扫描所有输入模型点集，对每个模型计算其包围盒，然后计算要参与碰撞检测的模型的包围盒对进行相交测试，假设要计算参与碰撞检测的模型的包围盒相交的对数为 n_1，再对这 n_1 对模型计算其 k-CBP 并进行初始化，如构造 AABB 树、初始化 GJK 算法，并计算 k-CBP 是否相交，此步骤后剩余模型对数为 n_2，最后再对这 n_2 对模型进行构造 AABB 树进而进行相交测试，真实模型相交对数为 n_3。整个流程中，包围盒的命中率为 n_1/n_3，

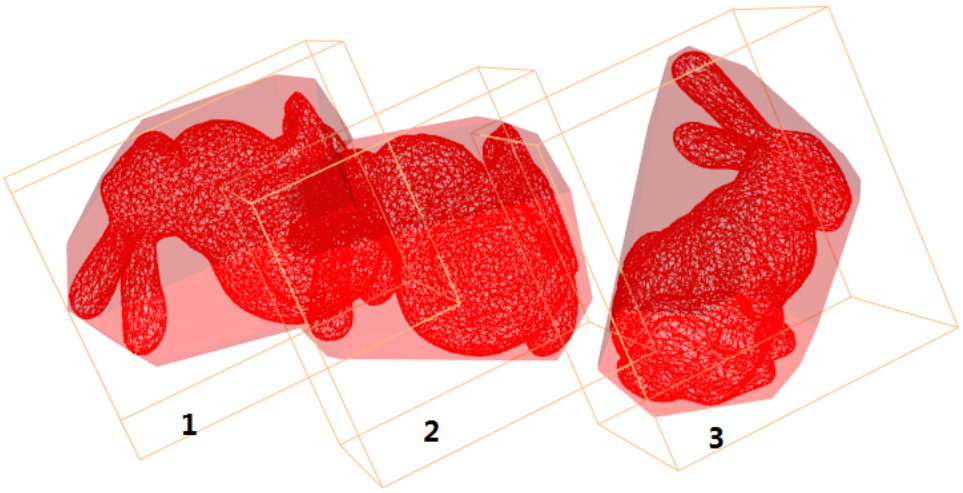


图 1.6　k–CBP 应用于碰撞检测示例

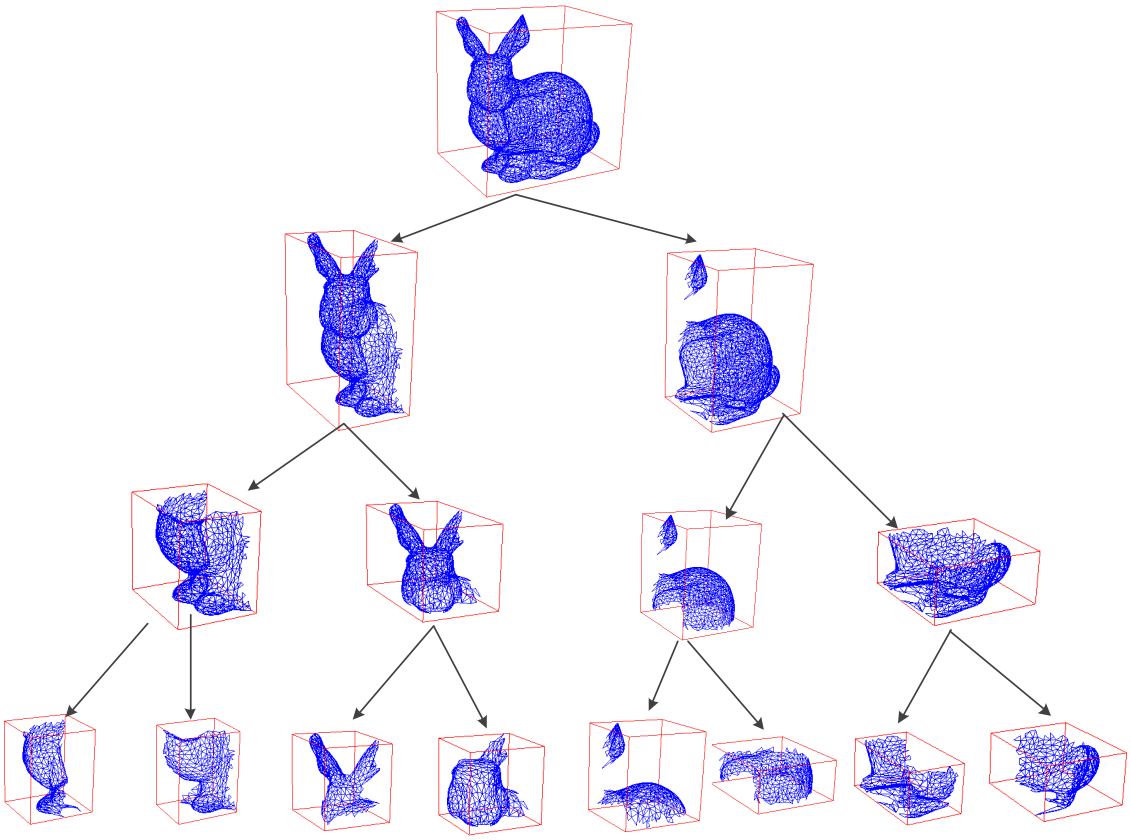


图 1.7　Bunny 模型的 AABB 树形结构 (部分)

k-CBP 的命中率为 n_2/n_3。

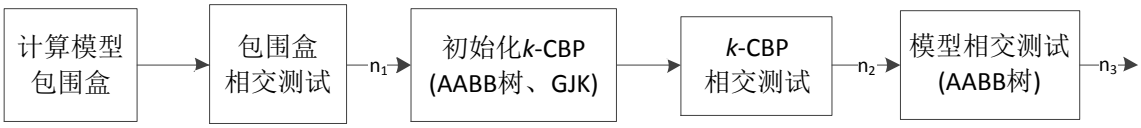


图 1.8　碰撞检测算法流程图

1.3.2　运动场景中的碰撞检测算法

当在运动场景中的模型进行碰撞检测时，模型中的点坐标会更新，一种方法是重新计算模型中的所有点再对模型做碰撞检测，但当模型点数量较大时，此方法不可取；另外一种算法是仍然利用静态场景中的 AABB 树形结构，仅重新计算将要碰撞的节点的坐标值，进而进行相交检测。节点包围盒坐标在运动过程中发生变化，精确的包围盒是重新计算该节点包含原始模型的点在变化后的点坐标值的包围盒，本文利用一种近似算法即仅对包围盒的 8 个顶点进行转换然后计算这 8 个顶点的包围盒，用这个近似包围盒进行遍历剪枝，当到叶子节点后，再重新计算网格模型的点的新坐标值用同样的方法进行相交检测。

模型围绕任意过原点的轴 $\boldsymbol{n}(x,y,z)$ 旋转任意角度 θ 的变换矩阵如公式 3-14 所示，其中 $\lambda = 1 - \cos\theta$，详细的推导过程可以参考文献 [64]。

$$\begin{cases} \boldsymbol{R}(\boldsymbol{n},\theta) & = \cos\theta \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \lambda \begin{pmatrix} \boldsymbol{n}_x^2 & \boldsymbol{n}_x\boldsymbol{n}_y & \boldsymbol{n}_x\boldsymbol{n}_z \\ \boldsymbol{n}_x\boldsymbol{n}_y & \boldsymbol{n}_y^2 & \boldsymbol{n}_y\boldsymbol{n}_z \\ \boldsymbol{n}_x\boldsymbol{n}_z & \boldsymbol{n}_y\boldsymbol{n}_z & \boldsymbol{n}_z^2 \end{pmatrix} + \sin\theta \begin{pmatrix} 0 & \boldsymbol{n}_z & -\boldsymbol{n}_y \\ -\boldsymbol{n}_z & 0 & \boldsymbol{n}_x \\ \boldsymbol{n}_y & -\boldsymbol{n}_x & 0 \end{pmatrix} \\ & = \begin{pmatrix} \cos\theta + \boldsymbol{n}_x^2\lambda & \boldsymbol{n}_x\boldsymbol{n}_y\lambda + \boldsymbol{n}_z\sin\theta & \boldsymbol{n}_x\boldsymbol{n}_z\lambda - \boldsymbol{n}_y\sin\theta \\ \boldsymbol{n}_x\boldsymbol{n}_y\lambda - \boldsymbol{n}_z\sin\theta & \cos\theta + \lambda\boldsymbol{n}_y^2 & \boldsymbol{n}_y\boldsymbol{n}_z\lambda + \boldsymbol{n}_x\sin\theta \\ \boldsymbol{n}_x\boldsymbol{n}_z\lambda + \boldsymbol{n}_y\sin\theta & \boldsymbol{n}_y\boldsymbol{n}_z\lambda - \boldsymbol{n}_x\sin\theta & \cos\theta + \boldsymbol{n}_z^2\lambda \end{pmatrix} \end{cases} \tag{1-14}$$

当模型平移时，点 $\boldsymbol{P}(x,y,z)$ 平移 $\boldsymbol{t}(t_x,t_y,t_z)$ 后的点坐标为 $\boldsymbol{P'}(x+t_x,y+t_y,z+t_z)$，可用 4×4 的变换矩阵 $\boldsymbol{T}$ 表示，即

$$\boldsymbol{T}(\boldsymbol{t}) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1-15}$$

因此当模型平移旋转时产生的变换矩阵 $\boldsymbol{M}$ 为

$$\boldsymbol{M} = \boldsymbol{R}(\boldsymbol{n}, \theta) \cdot \boldsymbol{T}(\boldsymbol{t}), \tag{1-16}$$

AABB 节点包围盒更新后的算法如算法 10 所示，算法输入为平移向量 $\boldsymbol{t}$ 及旋转方向和角度 $\boldsymbol{n}, \theta$ 或者直接传入变化矩阵 $\boldsymbol{M}$ 即可。

算法 5 AABB 节点包围盒更新算法

输入: AABB 包围盒 box，平移旋转变换矩阵 $\boldsymbol{M}$
输出: 变换之后的包围盒 box'

```
1: function TransformBox(box, M)
2:     vertices ← GetAABBVertices(box)
3:     box' ← ∅
4:     for all v ∈ vertices do  // 遍历 box 的 8 个顶点
5:         v' = M · v  // 计算变换后的点坐标
6:         update(box', v')  // 根据变换后的顶点 v 更新 box'
7:     end for
8:     return box'
9: end function
```

运动场景中模型的碰撞检测算法如算法 11 所示，假设参与碰撞检测的两个模型中第一个运动且变换矩阵为 $\boldsymbol{M}$，第二个静止，两个模型都运动时，可视第二个模型相对静止，将 $\boldsymbol{M}$ 设置为第一个相对于第二个模型的相对变换矩阵。

算法 11 是一个递归算法，从 AABB 树的根节点起向底层进行深度优先遍历，当检测到运动后的模型的某两个叶子节点的包围盒相交时，遍历其三角网格的相交检测算法，在实际实现过程中，本文的叶子节点仅含一个三角网格。运动模型的三角网格应用变换矩阵得到一个新的三角网格，同样用 3.2 的算法对新的三角网格进行相交测试。除了用该递归算法外，也可对算法 7 进行稍许改动（只需改变包围盒检测和三角网格检测的函数）的迭代算法，此处不在赘述。

在基于 GJK 的 k-CBP 碰撞检测算法中，只需要改变算法 8 中 $Support$ 子过程，将变换矩阵应用到原始顶点进行计算得到新的支持点即可，而在连续运动的模型碰撞检测过程中，利用爬山法可以将支持点的搜索优化到常数时间复杂度[47]。

1.4　实验结果

1.4.1　与包围盒过滤算法对比

本文实验通过生成不同数量的模型 (模型位置和旋转角度随机生成), 碰撞检测时首先判断包围盒是否相交, 然后判断凸包围多面体是否相交, 最后再判断实际模型是否相交. 凸包围多面体之间的碰撞检测时可采用文献 [27] 中提到的方法, 本文案例中模型和凸包围多面体是否相交都采用了普通 AABB 树的方式进行判断, 从

算法 6 运动场景中基于 AABB 树碰撞检测算法

输入: 两个模型 AABB 树的根节点 $root_0, root_1$ 及平移旋转变换矩阵 $\boldsymbol{M}$
输出: 模型是否相交

```
1: function MovingTraverseDetection(root_0, root_1, M)
2:     mBox ← TransformBox(M, root_0.box) // 按照算法 10 计算运动后的包围盒
3:     c ← Intersect(mBox, root_1.box)
4:     if c = False then
5:         return False // 包围盒不相交，直接返回 False
6:     end if
7:     if root_0.IsLeaf() then
8:         if root_1.IsLeaf() then // 两个叶子节点的原始几何进行相交测试
9:             for all p_1 ∈ root_0.primitives do
10:                for all p_2 ∈ root_1.primitives do
11:                    return Intersect(M, p_1, p_2) // 将 p_1 应用于变换矩阵 M 后采用算法3.2中
   进行三角网格相交测试
12:                end for
13:            end for
14:        else
15:            if MovingTraverseDetection(root_0, root_1.left) or
                  MovingTraverseDetection(root_0, root_1.right) then
16:                return True
17:            end if
18:        end if
19:    else
20:        if root_1.IsLeaf() then
21:            if MovingTraverseDetection(root_0.left, root_1) or
                  MovingTraverseDetection(root_0.right, root_1) then
22:                return True
23:            end if
24:        else // 两个节点都有孩子节点
25:            if MovingTraverseDetection(root_0.left, root_1.left) or
                  MovingTraverseDetection(root_0.left, root_1.right) or
                  MovingTraverseDetection(root_0.right, root_1.left) or
                  MovingTraverseDetection(root_0.right, root_1.right) then
26:                return True
27:            end if
28:        end if
29:    end if
30:    return False
31: end function
```

如表 3.1 的实验结果可看出含有凸包多围体的模型之间的碰撞检测算法能显著提高整体应用的效率.

表 1.1　*k*-CBP 和包围盒应用于碰撞检测结果对比

n	CT(Box) (ms)	CT(16-CBP) (ms)	DT(Box) (ms)	DT(16-CBP) (ms)	r(Box) (%)	r(*k*-CBP) (%)	DP(Model)
10	0.1	1.8	26.0	0.1	0.00	100.00	0
30	0.2	2.9	134.0	70.0	45.45	83.33	5
50	0.5	4.8	506.0	255.2	46.34	86.36	19
70	0.4	4.8	901.1	492.5	44.16	80.95	34
90	0.7	5.7	1324.0	734.7	41.82	73.02	46
100	0.7	7.8	1481.0	870.7	43.31	75.34	55
150	1.0	9.8	4153.1	2473.0	42.98	70.75	150
200	1.6	12.8	8049.3	4430.9	41.02	71.32	281

如表 3.1 所示, 其中 n 表示场景中模型的数量, CT(Box), CT(16-CBP) 分别表示模型包围盒的构造时间（Construct Time，简称 CT）和凸包围 16 面体的构造时间（单位 ms）①, DT(Box), DT(16-CBP) 分表表示用包围盒进行碰撞检测和利用凸包围 16 面体进行碰撞检测所耗费的时间, 其中 r(Box), r(16-CBP) 分别表示包围盒、16-CBP 的命中率（即用实际模型相交的数量除以包围体检测出来相交的数量）, DP(Model) 检测到的模型实际相交的对数（Detection Pair，简称 DP），显然计算模型包围盒所耗费的时间要明显少于计算凸包围多面体的时间, 但由于凸包围多面体比包围盒紧致, 因而命中率比包围盒高, 能排除更多本不相交的模型进而节省碰撞检测总时间, 提高算法效率。该部分工作已经发表，详细内容见参考文献 [65]。

1.4.2　静止场景中与 *k*-DOP 算法对比

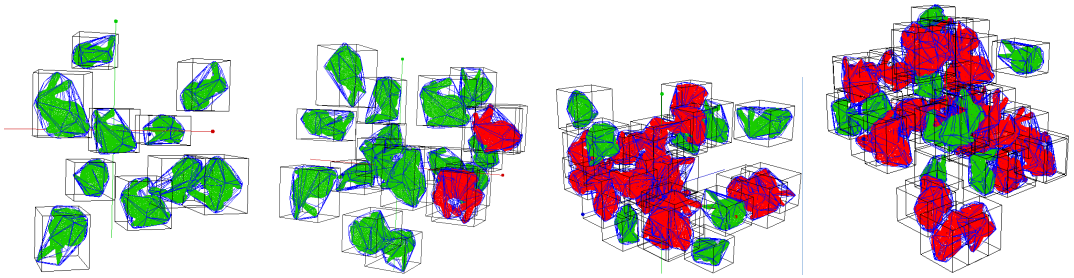
CollDet ① 是 Gabriel Zachmann 等人实现的一个碰撞检测库[66]，内含基于 *k*-DOP 的实现，本文将从静态和动态两个角度对其进行对比实验分析，为了和 *k*-DOP 进行公平实验对比，本文在实验过程中均只通过 CPU 进行计算，并未采用 GPU 加速，针对碰撞检测实验环境中多个相同的实验模型，*k*-CBP 与 *k*-DOP 的构造均采用重新扫描扫描点集构造。

在静态场景中的碰撞检测实验中，本文生成不同数量的模型，其位置和旋转角度随机生成（*k*-DOP 和 *k*-CBP 采用相同的随机生成的数据），图 3.9 为 Bunny 模

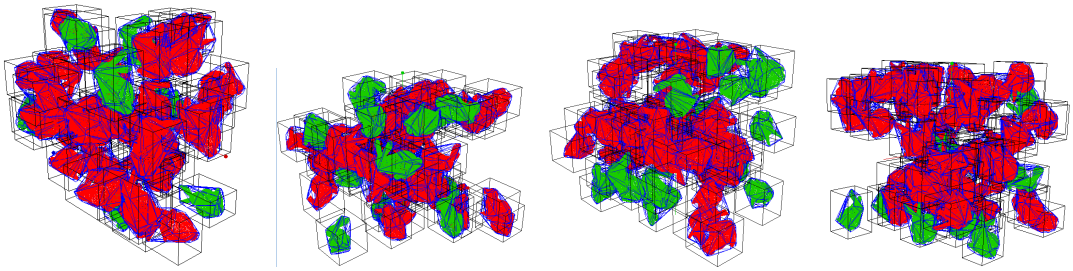
① 此处的时间为得到一个 *k*-CBP 后直接通过应用变换矩阵得到新的 *k*-CBP 总时间且是利用 GPU 搜索截面的时间，后文与 *k*-DOP 对比均采用 CPU 算法实现

① 其源码和文档均可通过 http://cgvr.cs.uni-bremen.de/research/colldet/ 下载得到

型在模拟碰撞检测的实验实例。



(a) n=10　(b) n=20　(c) n=30　(d) n=40



(e) n=50　(f) n=60　(g) n=70　(h) n=80

图 1.9　静态场景下 Bunny 模型碰撞检测示例

表 3.2 为本文算法与 *k*-DOP 均采用 CollDet 中默认值 $k = 24$ 针对 Bunny 模型的实验结果。其中，n 表示场景中模型的数量，n(Box) 表示通过包围盒过滤后构造碰撞检测对的有效模型数量，$n - n(\mathtt{Box})$ 的值表示场景中模型的包围盒不与任何其他模型包围盒相交的模型数量，CT(*k*-DOP) 和 CT(*k*-CBP) 分别表示初始化的时间（单位 ms），CT(*k*-DOP) 包括计算模型包围盒和计算 *k*-DOP 树，CT(*k*-CBP) 包括计算模型包围盒，*k*-CBP 以及用于实际模型碰撞检测算法的 AABB 树，DT(*k*-DOP)、DT(*k*-CBP)(*AABB*) 和 DT(*k*-CBP)(*GJK*) 均表示碰撞检测所耗费的时间，DT(*k*-CBP)(*AABB*) 为通过第 3.1.1 节中介绍的 AABB 方法对 *k*-CBP 进行相交检测（包括构造 *k*-CBP 的 AABB 树所耗费的时间），而 DT(*k*-CBP)(*GJK*) 表示通过第 3.1.2 节中介绍的 GJK 方法对 *k*-CBP 进行相交检测，相应的时间都包括 *k*-CBP 相交后对模型进行 AABB 相交检测时间。

表 3.3 记录了更加详细的实验结果，其中 CT(*k*-CBP) 和 CT(AABB) 分别表示本文算法中构造 *k*-CBP 的时间和构造用于具体模型碰撞检测算法的 AABB 树的时间，DP(Box) 和 DP(Model) 表示包围盒相交的对数和实际模型碰撞的对数，这两组数据与 *k*-DOP 算法中相同，DP(*k*-CBP) 表示 *k*-CBP 相交的对数。结合二表可知，在前期初始化碰撞检测环境中，本文算法优势明显，当模型为 100 时，*k*-DOP 算法构造 *k*-DOP 树需要近 9 秒的时间，而本文算法不到 1 秒，初始化环境后，进入碰撞检测环节，本文算法较 *k*-DOP 相差不大，而本文提出的基于 AABB 树算法和

表 1.2　静态场景下本文算法与 *k*-DOP 结果对比（Bunny）

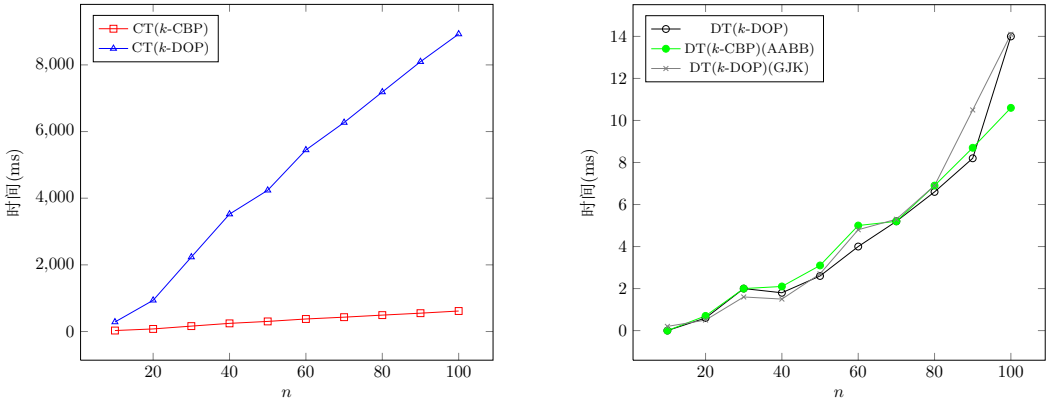
n	*n*(Box)	CT(*k*-DOP) (ms)	CT(*k*-CBP) (ms)	DT(*k*-DOP) (ms)	DT(*k*-CBP) (*AABB*) (ms)	DT(*k*-CBP) (*GJK*) (ms)
10	3	285.00	29.30	0.01	0.01	0.20
20	10	937.20	77.30	0.60	0.70	0.50
30	24	2233.60	162.20	2.00	2.00	1.60
40	38	3522.20	244.70	1.80	2.10	1.50
50	46	4238.20	302.90	2.60	3.10	2.70
60	59	5447.60	375.80	4.00	5.00	4.80
70	68	6271.00	430.40	5.20	5.20	5.30
80	78	7186.20	492.10	6.60	6.90	6.90
90	88	8096.20	550.30	8.20	8.70	10.50
100	97	8925.80	615.00	14.00	10.60	14.10

基于 GJK 算法中，对于较少三角网格的 Bunny 模型而言，AABB 算法在静态环境中较 GJK 更优，而从后文第 3.4.3 节也可以看出基于 GJK 算法在动态碰撞检测的环境中更优。

表 1.3　静态场景下本文算法实验结果（Bunny）

n	CT(*k*-CBP)(ms)	CT(AABB)(ms)	DP(Box)	DP(*k*-CBP)	DP(Model)
10	13.60	16.60	2	2	0
20	23.80	53.40	10	4	2
30	35.00	128.00	42	31	23
40	42.40	199.40	42	21	19
50	51.00	244.20	66	43	35
60	58.00	313.40	121	62	49
70	66.80	360.20	144	77	61
80	72.20	412.80	167	101	85
90	81.00	463.80	261	119	84
100	92.00	514.20	335	200	161

图 3.10 为本文的两种算法和文献 [60,66] 中基于 *k*-DOP 的实现的实验结果的曲线展示图，从图 3.10(a) 可以看出，初始化构造 *k*-DOP 树和本文算法初始化所耗费时间与参与碰撞检测的模型的数量近线性关系，但明显本文算法所耗费时间更少，图 3.10(b) 为初始化碰撞检测环境后进行碰撞检测所耗费的时间，本文的两种算法和基于 *k*-DOP 的算法相比，碰撞检测所耗费的时间相差不大，总体来说针对 Bunny 模型而言，*k*-DOP 所耗费时间稍微较少，但三者相差不大，整体在 1–4 毫



(a) 初始化时间对比　　(b) 碰撞检测时间对比

图 1.10　静态场景下本文算法与 k-DOP 实验结果对比（Bunny）

秒范围内。

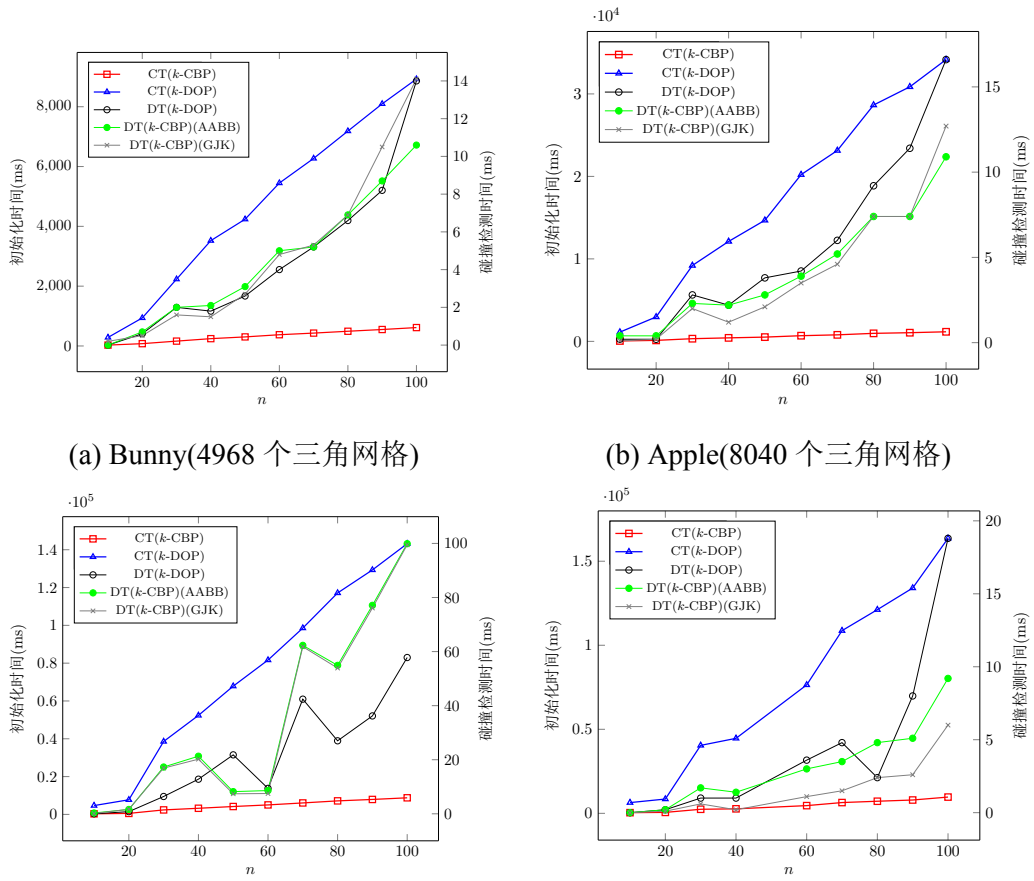
图 3.11 记录了更多模型的实验结果，其中曲线三角形和正方形记录的是碰撞检测环境中的初始化时间，另外三条曲线分别是本文的两种算法和基于 k-DOP 的算法碰撞检测所耗费的时间，左边纵坐标刻度为初始化时间，右边纵坐标刻度为碰撞检测的时间。从中可得，在静态碰撞检测实验中，本文算法与基于 k-DOP 的算法相比，在初始化环境过程中占用更少的时间，而在碰撞检测过程中针对不同的模型可能有不同的结果。从整体来看，本文算法优于基于 k-DOP 的算法。

对于更大的 k 值，构造层次结构的 k-DOP 树时会耗费更多的时间和存储空间，在模型较大且模型数量较多时，例如当对 70 个 Dinosaur 模型进行构造 k-DOP 树时，32 位程序已经会报内存不足的异常，且构造时间也更久，图 3.12 为 k-DOP 实现中的最大值 $k = 46$ 的实验结果。

更大的 k 值不一定能够使碰撞检测的效率提高，这与具体的碰撞检测环境有关，以 Bunny 模型和 Apple 模型为例，如图 3.13 为 $k = 24$ 和 $k = 46$ 的结果对比，其中，虚线的结果为 $k = 46$ 的结果，实现为 $k = 24$ 的结果，看出三种算法在 $k = 24$ 时碰撞检测所耗费的时间更少。因此在实际应用环境中应该根据具体的碰撞检测环境选择不同的 k 值。

1.4.3　运动场景中与 k-DOP 算法对比

对于运动场景中的碰撞检测实验，本文采取随机读入单个模型，并随机生成多个平移旋转变换构造多个模型，然后另其中一个模型产生随机平移旋转变换进行运动，检测该运动模型与其他所有的模型进行碰撞检测，图 3.14 为随机产生的 10 个模型，并让其中一个随机运动 7 步的示意图，示例中，对于没有和运动模型产生碰撞的模型，图中仅显示了其包围盒和 k-CBP。



(a) Bunny(4968 个三角网格)　(b) Apple(8040 个三角网格)

(c) Budda(60878 个三角网格)　(d) Dinosaur(64546 个三角网格)

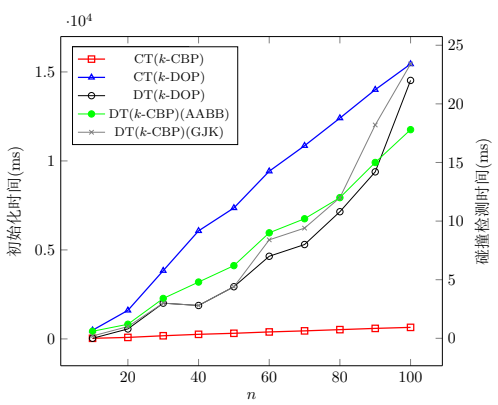
图 1.11　静态场景下本文算法与 k-DOP 实验结果对比 (k=24)

表 3.4 为本文算法和 k-DOP 算法均采用 k-DOP 实现中 $k = 24$ 默认值的初始化时间对比，init(k-CBP) 所代表的时间表示构造 k-CBP 及模型的 AABB 树所耗费的时间，init(k-DOP) 为构造 k-DOP 所耗费的时间，n 的数量为动态碰撞检测场景中模型的总数量。

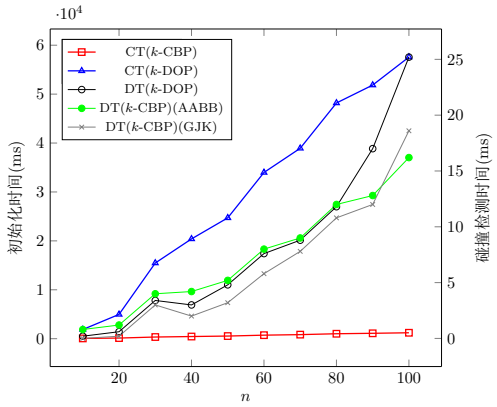
表 1.4　本文算法与 k-DOP 初始化时间对比 (k=24)

Model	$n = 2$		$n = 10$	
	init(k-CBP)	init(k-DOP)	init(k-CBP)	init(k-DOP)
Bunny	17.63	189.78	116.59	926.10
Apple	30.87	749.48	226.20	3732.43
Budda	193.91	3134.78	1713.13	15677.02
Dinosaur	250.55	4281.15	2230.88	21493.08

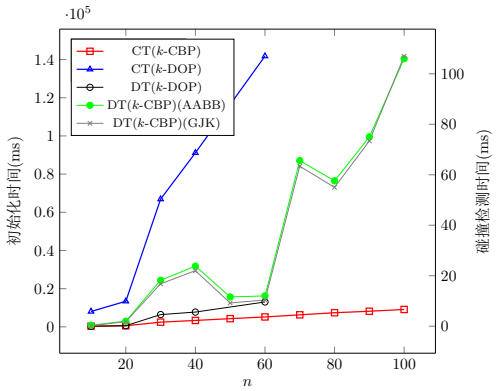
在运动环境中，本文算法与 k-DOP 算法初始化过程与在静态环境中一样，k-DOP 需要构造 k-DOP 树也因此耗费很多时间，本文需要构造 AABB 树，也是



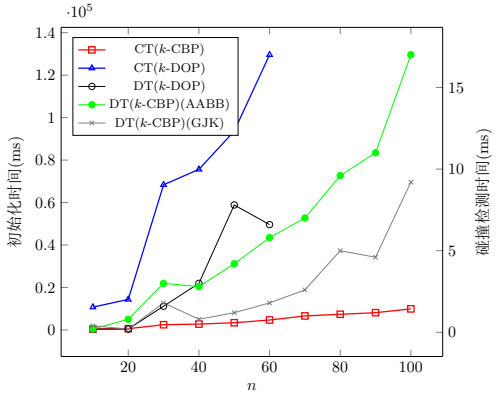
(a) Bunny(4968 个三角网格)



(b) Apple(8040 个三角网格)

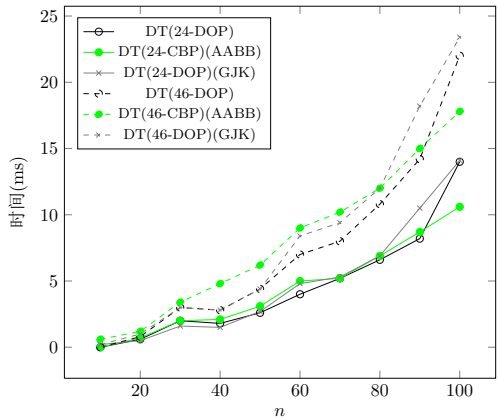


(c) Budda(60878 个三角网格)

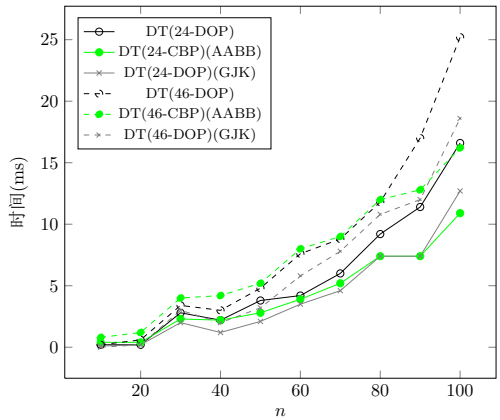


(d) Dinosaur(64546 个三角网格)

图 1.12　静态场景下本文算法与 *k*-DOP 实验结果对比 (k=46)

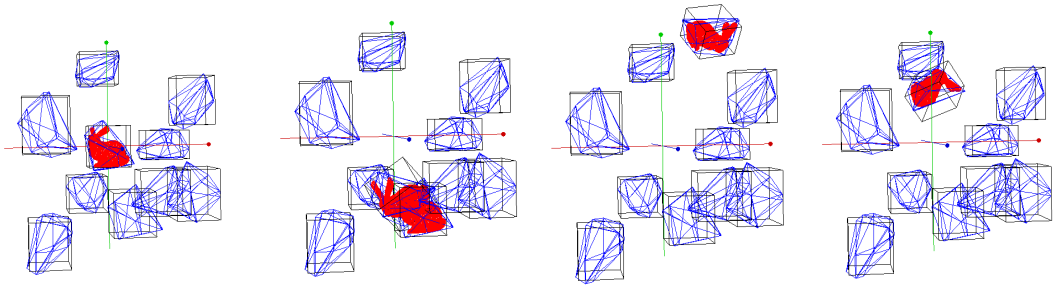


(a) Bunny(4968 个三角网格)

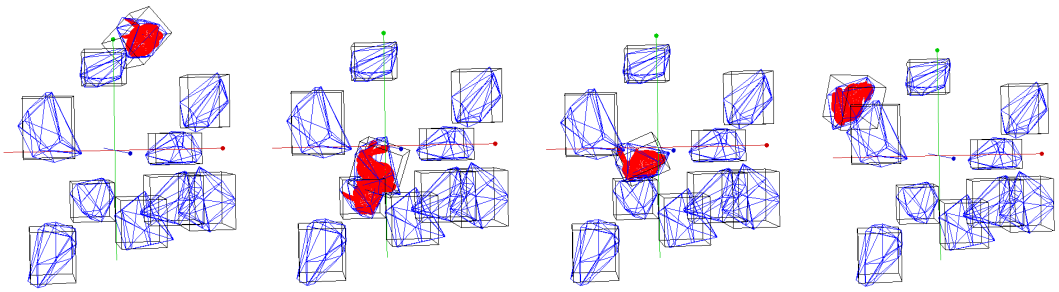


(b) Apple(8040 个三角网格)

图 1.13　静态场景下不同 *k* 值实验结果对比



(a) move=0　(b) move=1　(c) move=2　(d) move=3



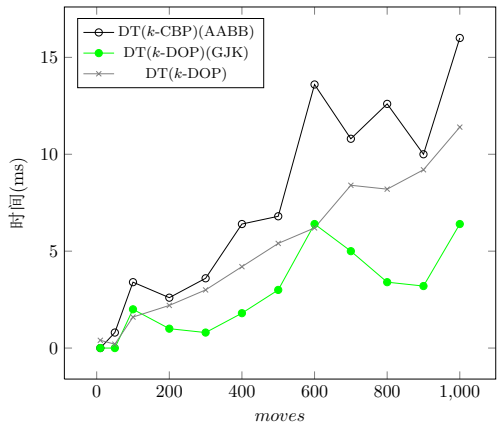
(e) move=4　(f) move=5　(g) move=6　(h) move=7

图 1.14　运动场景下 Bunny 模型碰撞检测示例

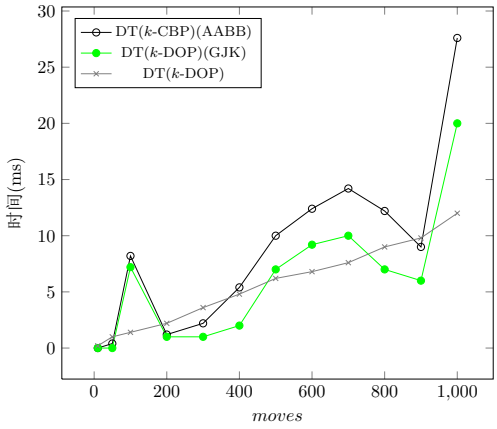
初始化过程中占用时间较多的步骤。图 3.15 为运动场境下本文算法与 *k*-DOP 实验结果对比，其中凸包围多面体面数 $k = 24$。横坐标表示其中一个模型运动的步数 *moves*，纵坐标表示随机运动 *moves* 步碰撞检测所耗费的时间，实心圆所代表曲线为前文提出的基于 GJK 算法实现，空心圆为基于 *k*-DOP 实现，另外的为前文提出的基于 AABB 树实现。

在 Bunny 模型中，本文基于 GJK 算法耗时最少，基于 *k*-DOP 算法耗时最久，本文基于 AABB 算法居中，总体趋势为 *moves* 越大，耗时也越久，表 3.5 为 2 个 Bunny 模型在随机运动 *moves* 步和 *k*-DOP 算法的比较结果，当真实模型相交数量较多时，三种算法所耗费的时间也更多。

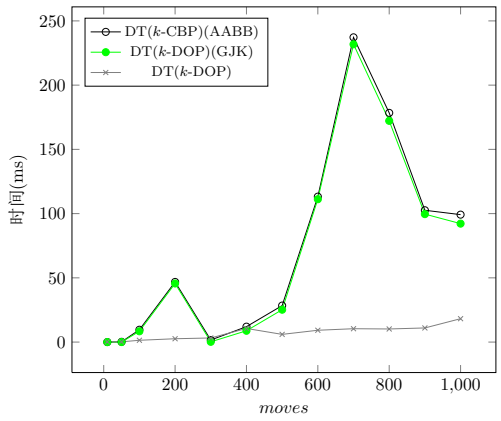
随着三角网格增多，三种算法所耗费时间均有所增加，总体而言，基于 *k*-DOP 的算法比较稳定，而本文算法随着模型不同波动较大。如图 3.15(c) 所示的 Budda 模型，本文的两种算法耗时较一致，因为两种算法在 *k*-CBP 相交后均依赖于模型基于 AABB 树的碰撞检测算法。实际上，最终碰撞检测耗时与在运动过程中相交的次数有关，且同时也与相交位置有关。图 3.15(c) 所示的 Budda 模型在 $moves = 700$ 时，均耗时 200 多毫秒，因为此时 *k*-CBP 相交对数为 DP(*k*-CBP)= 10，且实际模型相交对数为 5，且在这 5 次模型真实相交过程中，调用单独包围盒和旋转包围盒碰撞检测算法最大达 46044 次，且三角网格相交测试数量也达到 8040 次，因此总体耗时很久，深究原因发现 Budda 模型的底部包含有很多极小的共面三角形，导致很多这些极小的共面三角形与其他三角形发生多次碰撞



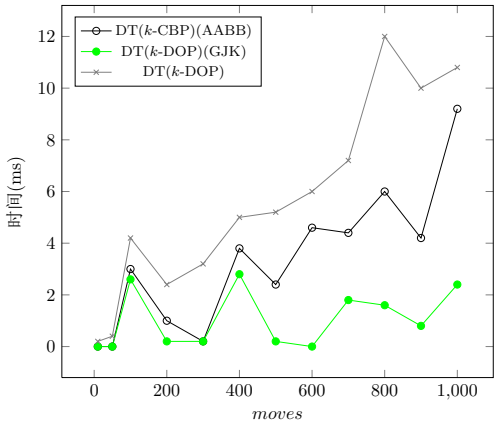
(a) Bunny(4968 个三角网格)



(b) Apple(8040 个三角网格)



(c) Budda(60878 个三角网格)



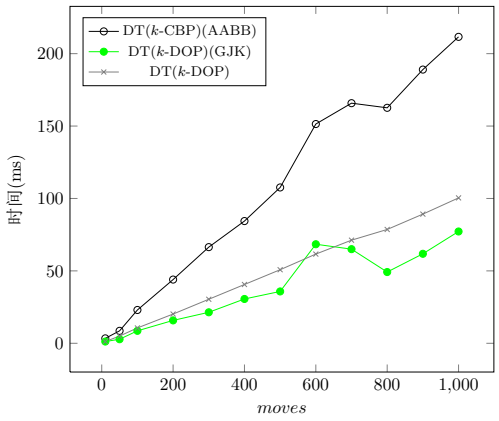
(d) Dinosaur(64546 个三角网格)

图 1.15　运动场境下本文算法与 k-DOP 实验结果对比 ($k = 24, n = 2$)

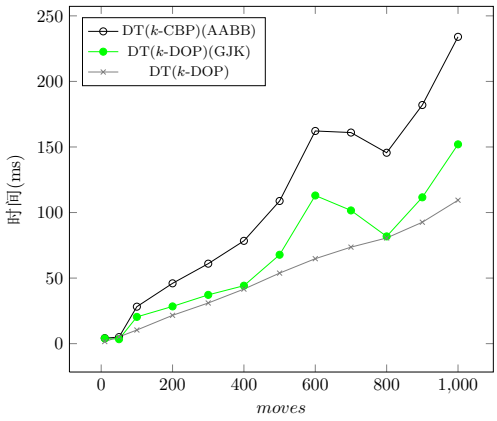
表 1.5　运动场景下本文算法与 k-DOP 实验结果（Bunny）

moves	DT(k-CBP)(AABB)	DT(k-DOP)	DT(k-CBP)(GJK)	DP(k-CBP)	DP(Model)
10	0.00	0.40	0.01	0	0
50	0.80	0.20	0.01	0	0
100	3.40	1.60	2.00	4	4
200	2.60	2.20	1.00	1	1
300	3.60	3.00	0.80	2	2
400	6.40	4.20	1.80	8	6
500	6.80	5.40	3.00	11	8
600	13.60	6.20	6.40	10	8
700	10.80	8.40	5.00	14	8
800	12.60	8.20	3.40	10	8
900	10.00	9.20	3.20	9	7
1000	16.00	11.40	6.40	22	17

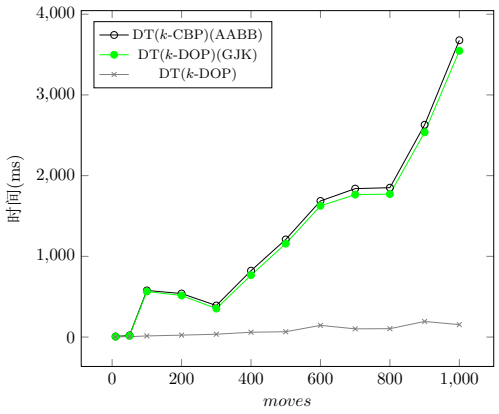
检测，这表明本文算法存在一定鲁棒性问题。



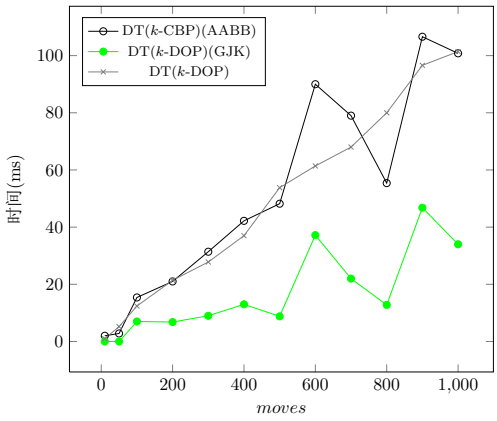
(a) Bunny(4968 个三角网格)



(b) Apple(8040 个三角网格)



(c) Budda(60878 个三角网格)



(d) Dinosaur(64546 个三角网格)

图 1.16　运动场境下本文算法与 k-DOP 实验结果对比 ($k = 24, n = 10$)

当模型数量 n 的值增大时，三种算法整体趋势仍能保持一致，如图 3.16 所示为相同 k 值，相同运动路径（包括平移距离旋转角度以及运动步数）下 10 个模型下的碰撞检测结果。对于 Budda 模型，本文算法仍耗时较久，原因与 $n = 2$ 时一致，即 k-CBP 相交，用 AABB 树进行模型的碰撞检测时，树形结构遍历较多且到叶子节点三角网格测试的次数也较多。整体来看，本文基于 k-CBP 结合 GJK 的算法在一定程度上有一定优势，而基于 k-DOP 算法相对较稳定，但其初始化时间很长。

k24–46 kdop 横向对比 k24–46 kcbp 横向对比

第 2 章　总结与展望

算法基本概念及框架

2.1　总结

2.2　展望

参考文献

[1] Bergen G v d. Efficient collision detection of complex deformable models using aabb trees. Journal of Graphics Tools, 1997, 2(4):1–13.

[2] Ericson C. Real-time collision detection. Taylor & Francis US, 2005.

[3] Gottschalk S, Lin M C, Manocha D. Obbtree: a hierarchical structure for rapid interference detection. the 23rd annual conference on Computer graphics and interactive techniques. ACM, 1996. 171–180.

[4] O'Rourke J. Finding minimal enclosing boxes. International journal of computer & information sciences, 1985, 14(3):183–199.

[5] Barequet G, Har-Peled S. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. Journal of Algorithms, 2001, 38(1):91–109.

[6] Chan C, Tan S. Determination of the minimum bounding box of an arbitrary solid: an iterative approach. Computers & Structures, 2001, 79(15):1433–1449.

[7] Welzl E. Smallest enclosing disks (balls and ellipsoids). Results and New Trends in Computer Science. Springer-Verlag, 1991. 359–370.

[8] Larsson T. Fast and tight fitting bounding spheres. The Annual Swedish Computer Graphics Association Conference(SIGRAD), 2008. 27–30.

[9] Kay T L, Kajiya J T. Ray tracing complex scenes. ACM SIGGRAPH Computer Graphics, volume 20. ACM, 1986. 269–278.

[10] Klosowski J T, Held M, Mitchell J S, et al. Efficient collision detection using bounding volume hierarchies of k-dops. IEEE Transactions on Visualization and Computer Graphics, 1998, 4(1):21–36.

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

[12] 邓俊辉. 计算几何-算法与应用, 2005.

[13] Donald R Chand S S K. An Algorithm for Convex Polytopes. Journal of the Association for Computing Machinery, 1970, 17(1):78–86.

[14] Preparata F P, Hong S J. Convex hulls of finite sets of points in two and three dimensions. Communications of the ACM, 1977, 20(2):87–93.

[15] Hossain M Z, Amin M A. On constructing approximate convex hull. American Journal of Computational Mathematics, 2013, 3:11–17.

[16] Bentley J L, Preparata F P, Faust M G. Approximation algorithms for convex hulls. Communications of the ACM, 1982, 25(1):64–68.

[17] Kavan L, Kolingerova I, Zara J. Fast approximation of convex hull. the 2nd IASTED international conference on Advances in computer science and technology(ACST), 2006. 101–104.

[18] Zunié J. AN OUTER APPROXIMATION OF THE CONVEX HULL FOR FINITE GRID POINT SETS. Novi Sad Journal of Mathematics, 1992, 22(2):177–185.

[19] Crosnier A, Rossignac J. Tribox bounds for three-dimensional objects. Computers & Graphics, 1999, 23(3):429–437.

[20] Larsen E, Gottschalk S, Lin M C, et al. Fast proximity queries with swept sphere volumes. Technical report, Department of Computer Science, University of North Carolina, 1999.

[21] Krishnan S. Spherical shell: A higher order bounding volume for fast proximity queries. Third International Workshop on Algorithmic Foundations of Robotics, 1997.

[22] Guibas L J, Nguyen A, Zhang L. Zonotopes as bounding volumes. the fourteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2003. 803–812.

[23] Schömer E, Sellen J, Teichmann M, et al. Smallest enclosing cylinders. Algorithmica, 2000, 27(2):170–186.

[24] Held M. Erit - a collection of efficient and reliable intersection tests. Journal of Graphics Tools, 1997, 2(4):25–44.

[25] Wang W, Choi Y K, Chan B, et al. Efficient collision detection for moving ellipsoids using separating planes. Computing, 2004, 72(1-2):235–246.

[26] Assarsson U, Moller T. Optimized view frustum culling algorithms for bounding boxes. Journal of Graphics Tools, 2000, 5(1):9–22.

[27] Wald I, Boulos S, Shirley P. Ray tracing deformable scenes using dynamic bounding volume hierarchies. ACM Transactions on Graphics (TOG), 2007, 26(1):6.

[28] 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.

[29] Larsson T, Akenine-Möller T. A dynamic bounding volume hierarchy for generalized collision detection. Computers & Graphics, 2006, 30(3):450–459.

[30] Madera F, Day A, Laycock S D. A hybrid bounding volume algorithm to detect collisions between deformable objects. Second International Conferences on Advances in Computer-Human Interactions(ACHI). IEEE, 2009. 136–141.

[31] Vogiannou A, Moustakas K, Tzovaras D, et al. Enhancing bounding volumes using support plane mappings for collision detection. Computer Graphics Forum, volume 29. Wiley Online Library, 2010. 1595–1604.

[32] Chang J W, Wang W, Kim M S. Efficient collision detection using a dual obb-sphere bounding volume hierarchy. Computer-Aided Design, 2010, 42(1):50–57.

[33] Tang M, Manocha D, Tong R. Fast continuous collision detection using deforming non-penetration filters. the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. ACM, 2010. 7–13.

[34] Zhigang F, Jianxun J, Jie X. Efficient collision detection using a dual k-dop-sphere bounding volume hierarchy. 2010 International Forum on Information Technology and Applications (IFITA), volume 3. IEEE, 2010. 185–189.

[35] Huebner K, Ruthotto S, Kragic D. Minimum volume bounding box decomposition for shape approximation in robot grasping. IEEE International Conference on Robotics and Automation(ICRA). IEEE, 2008. 1628–1633.

[36] Hubbard P M. Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics (TOG), 1996, 15(3):179–210.

[37] Lien J M, Amato N M. Approximate convex decomposition of polygons. Computational Geometry, 2006, 35(1):100–123.

[38] Lien J M, Amato N M. Approximate convex decomposition of polyhedra. the 2007 ACM symposium on Solid and physical modeling. ACM, 2007. 121–131.

[39] Lien J M, Amato N M. Approximate convex decomposition of polyhedra and its applications. Computer Aided Geometric Design, 2008, 25(7):503–522.

[40] Lien J M. Approximate convex decomposition and its applications[D]. Texas A&M University, 2006.

[41] Attene M, Mortara M, Spagnuolo M, et al. Hierarchical convex approximation of 3d shapes for fast region selection. Computer graphics forum, volume 27. Wiley Online Library, 2008. 1323–1332.

[42] Karlsson M, Winberg O, Larsson T. Parallel construction of bounding volumes. The Annual Swedish Computer Graphics Association Conference(SIGRAD), 2010.

[43] Lauterbach C, Garland M, Sengupta S, et al. Fast bvh construction on gpus. Computer Graphics Forum, volume 28. Wiley Online Library, 2009. 375–384.

[44] Melax S. Dynamic Plane Shifting BSP Traversal. Graphics Interface, 2000, c:213–220.

[45] Zeiller M, Purgathofer W, Gervautz M. Efficient collision detection for general csg objects. In: Terzopoulos D, Thalmann D, (eds.). Computer Animation and Simulation ’95, Eurographics. Springer Vienna, 1995: 66–79.

[46] Gilbert E G, Johnson D W, Keerthi S S. A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation, 1988, 4(2):193–203.

[47] Bergen G v d. A fast and robust gjk implementation for collision detection of convex objects. Journal of Graphics Tools, 1999, 4(2):7–25.

[48] Lin M, Gottschalk S. Collision detection between geometric models: A survey. Proc. of IMA Conference on Mathematics of Surfaces, volume 1, 1998. 602–608.

[49] Jiménez P, Thomas F, Torras C. 3d collision detection: a survey. Computers & Graphics, 2001, 25(2):269–285.

[50] Klein J, Zachmann G. Point cloud collision detection. Computer Graphics Forum, volume 23. Wiley Online Library, 2004. 567–576.

[51] Figueiredo M, Oliveira J, Araújo B, et al. An efficient collision detection algorithm for point cloud models. 20th International conference on Computer Graphics and Vision, volume 43, 2010. 44.

[52] Xinyu Zhang Y. Interactive collision detection for deformable models using streaming aabbs. IEEE Transactions on Visualization and Computer Graphics, 2007, 13(2):318–329.

[53] Bing H, Yangzihao W, Jia Z. An improved method of continuous collision detection using ellipsoids. International Conference on Computer-Aided Industrial Design and Conceptual Design, 2009.

[54] Jain A K. Data clustering: 50 years beyond K-means. Pattern Recognition Letters, 2010, 31(8):651–666.

[55] Wong T, Luk W, Heng P. Sampling with Hammersley and Halton points. Journal of graphics tools, (2):9–24.

[56] 仇德元. GPGPU 编程技术. 机械工业出版社, 2011: 1056–1063.

[57] Harris M. Optimizing parallel reduction in cuda. NVIDIA Developer Technology, 2007. 1–37.

[58] Preparata F P, Shamos M I. Computational geometry: an introduction, volume 47. 1985: 763.

[59] Preparata F, Muller D. Finding the intersection of n half-spaces in time O(n log n). Theoretical Computer Science, 1979, 8(1):45–55.

[60] Zachmann G. Rapid collision detection by dynamically aligned dop-trees. IEEE Transactions on Virtual Reality Annual International Symposium. IEEE, 1998. 90–97.

[61] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[62] Moller T. A Fast Triangle-Triangle Intersection Test. Journal of Graphics Tools, 1997, 2(2):25–30.

[63] 林建立, 唐磊, 雍俊海. 多边形网格的非流形封闭三角形网格正则化. 计算机辅助设计与图形学学报, 2014, 26(10):1557–1566.

[64] Dunn F, Parberry I. 3D Math Primer for Graphics and Game Development. Wordware Publishing, 2002.

[65] 唐磊, 施侃乐, 雍俊海, 等. 模型适应的凸包围多面体并行生成算法. 中国科学: 信息科学, 2014, 44(22):1515–1526.

[66] Sven Trenkel G Z. A benchmarking suite for static collision detection algorithms. Inter’l Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG), Plzen, Czech Republic. Union Agency, 2007..

致　谢

衷心感谢导师 xxx 教授和物理系 xxx 副教授对本人的精心指导。他们的言传身教将使我终生受益。

在美国麻省理工学院化学系进行九个月的合作研究期间，承蒙 xxx 教授热心指导与帮助，不胜感激。感谢 xx 实验室主任 xx 教授，以及实验室全体老师和同学们的热情帮助和支持！本课题承蒙国家自然科学基金资助，特此致谢。

感谢 THUTHESIS，它的存在让我的论文写作轻松自在了许多，让我的论文格式规整漂亮了许多。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

附录 A 基于着色器的并行算法关键代码

A.1 基于深度缓冲的算法

```
//顶点着色器
layout(location = 0) in vec3 inPosition;
flat out vec4 result;
uniform vec4 normal; //xyz: 坐标值, w: 法向索引
uniform float length;
uniform int normal_count;

void main()
{
    float distance = dot(inPosition, normal.xyz);
    float depth = distance / length; //放缩到 [-1, 1]
    depth = depth * 0.5 + 0.5; //映射到 [0,1]
    vec2 coord = vec2(-1 + normal.w * 2.0 / normal_count, 0.5);
    gl_Position = vec4(coord, depth, 1.0);
    result = vec4(gl_VertexID, 0, 0, 0);
}

//片段着色器
flat in vec4 result;
out vec4 output;

void main()
{
    output = vec4(result.x, 0, 0, 0);
}
```

A.2 基于乒乓技术的算法

```
//片段着色器
uniform sampler2DRect  imageSampler;
uniform sampler2DRect  indexSampler;
uniform int curPointCount;
uniform int curPointStart;

layout(packed) uniform Points
{
    //$MAX_POINT_SIZE$编译前会被替换
   vec4 curPoints[$MAX_POINT_SIZE$];
};

void main()
{
```

```
    vec4 normal_t = texture2DRect(imageSampler, gl_TexCoord[0].
        xy);
    float index_float = texture2DRect(indexSampler, gl_TexCoord
        [0].xy).w;
    int index = int(index_float);
    vec3 normal = normal_t.xyz;
    float max = normal_t.w;

    for(int i = 0; i < curPointCount; i++)
    {
        float t = dot(curPoints[i].bgr, normal); //BRGA格式
        if(t > max)
        {
            max = t;
            index = curPointStart + i;
        }
    }
   gl_FragData[0] = vec4(normal, max);
   gl_FragData[1] = vec4(0, 0, 0, index);//index为全局索引
}
```

个人简历、在学期间发表的学术论文与研究成果

个人简历

1989 年 12 月 30 日出生于重庆市石柱土家族自治县。

2008 年 9 月考入中南大学软件学院软件工程专业，2012 年 7 月本科毕业并获得工学学士学位。

2012 年 9 月免试进入清华大学软件学院攻读工学硕士学位至今。

发表的学术论文

[1] 唐磊, 李春平, 杨柳. 统计策略序列模式挖掘及其在软件缺陷预测中的应用 [J]. 计算机科学, 2013, 40(5): 164-167.

[2] Shi KanLe, Yong JunHai, Tang Lei, et al. Polar NURBS surface with curvature continuity[C]//Computer Graphics Forum. 2013, 32(7): 363-370.

[3] 唐磊, 施侃乐, 雍俊海等. 模型适应的凸包围多面体并行生成算法 [J]. 中国科学：信息科学, 2014, 44(12): 1515-1526.

[4] 林建立, 唐磊, 雍俊海等. 多边形网格的非流形封闭三角形网格正则化 [J]. 计算机辅助设计与图形学学报,2014,26(10):1557-1566.