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答辩报告)

学　　生：唐　　磊

指导教师：雍 俊 海 教授

计算机辅助设计图形学与可视化研究所

二〇一五年六月

目录

我将按照下面如下的次序来介绍本人的工作，首先是课题背景和相关工作，然后介绍凸包围体生成算法和基于凸包围体的碰撞检测算法，最后进行总结。

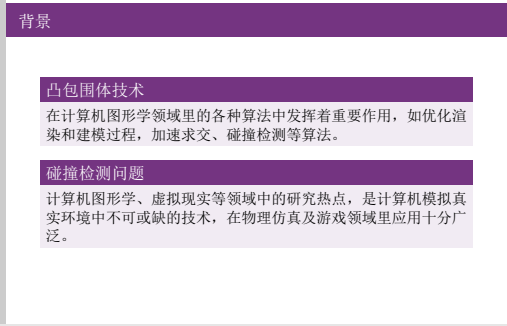
背景

凸包围体技术

在计算机图形学领域里的各种算法中发挥着重要作用，如优化渲染和建模过程，加速求交、碰撞检测等算法。

碰撞检测问题

计算机图形学、虚拟现实等领域中的研究热点，是计算机模拟真实环境中不可或缺的技术，在物理仿真及游戏领域里应用十分广泛。

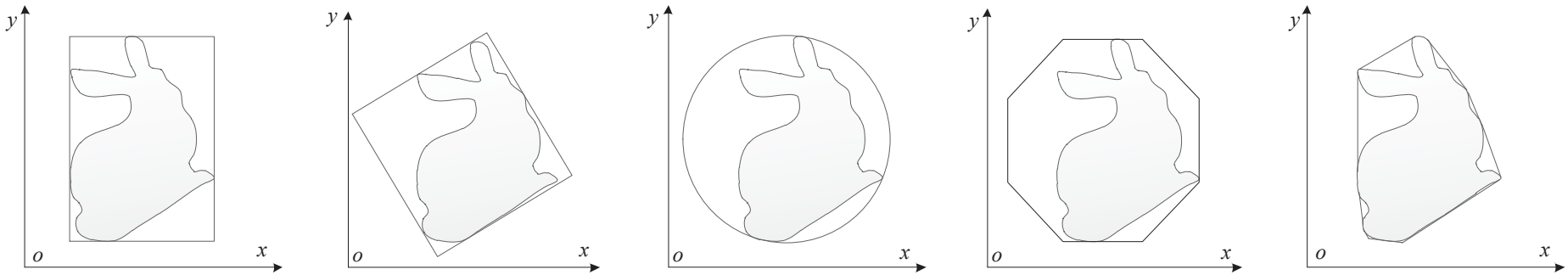


凸包围体技术在计算机图形学领域里的各种算法中发挥着重要作用，如优化渲染和建模过程，加速求交、碰撞检测等算法。主要是用在原始模型之间的相关计算（遮挡测试、相交测试等）之前进行预处理判断和剪枝，以求交算法为例，如果两个模型相交，则对应的凸包围体一定相交，若凸包围体不相交则其对应的原始模型一定不相交。而一般来讲，判断凸包围体是否相交比判断原始模型相交更简单，因此可以提升效率。

碰撞检测问题是计算机图形学、虚拟现实等领域中的研究热点，是计算机模拟真实环境中不可或缺的技术，在物理仿真及游戏领域里应用十分广泛。例如在游戏中，碰撞检测技术增强了游戏的真实性，游戏中的角色行走不可穿墙、角色中弹而亡等等都离不开碰撞检测技术。

目录

凸包围体的种类

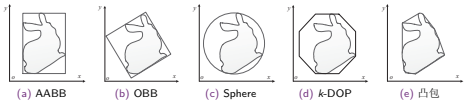


(a) AABB (b) OBB (c) Sphere (d) k-DOP (e) 凸包

Figure 1: 不同种类的包围体

其他：Tribox、Swept-sphere、Sphere-shell、Zonotopes、圆柱形、圆锥、椭球形等等。

凸包围体的种类



(a) AABB (b) OBB (c) Sphere (d) *k*-DOP (e) 凸包

Figure 1: 不同种类的包围体

其他：Tribox、Swept-sphere、Sphere-shell、Zonotopes、圆柱形、圆锥、椭球形等等。

上面这几个就是最常见的凸包围体。最常见的沿坐标轴方向的 AABB 包围盒，带方向的包围盒 OBB，包围球，k 面的凸包围体（k-DOP），和凸包，还有一些比较特定领域用的圆柱、圆锥形、椭球形等等。其中 k-DOP 是采用 k/2 对固定方向的半空间相交构成的凸包围体，是综合比较较好的包围体，因为可以通过 k 来调节包围体的简单性和紧致性来满足不同应用的需求。

本文目标

k-DOP[1]的局限性：方向**固定**且为**有限的偶数**，不同模型其截面方向一致，**不够紧致**；
而凸包很 (最) 紧致，但面片数量太多，构造复杂度 $O(n \log n)$。

本文凸包围体的目标

紧致： 能够自适应模型，根据模型形状特点有不同的方向；

快速： 生成凸包围体的速度要快，利用 GPU 加速；

灵活： 通过参数 k 调节凸包围体的简单性和紧致程度。

[1] James T Klosowski et al. "Efficient collision detection using bounding volume hierarchies of k-DOPs". In: *IEEE Transactions on Visualization and Computer Graphics* 4.1 (1998), pp. 21–36.

本文目标

***k*-DOP**[1]的局限性：方向**固定**且为**有限的偶数**，不同模型其截面方向一致，**不够紧致**；
而凸包很 (最) 紧致，但面片数量太多，构造复杂度 $O(n \log n)$。

本文凸包围体的目标

- 紧致： 能够自适应模型，根据模型形状特点有不同的方向；
- 快速： 生成凸包围体的速度要快，利用 GPU 加速；
- 灵活： 通过参数 k 调节凸包围体的简单性和紧致程度。

[1] James T Klosowski et al. "Efficient collision detection using bounding volume hierarchies of k-DOPs". In: *IEEE Transactions on Visualization and Computer Graphics* 4.1 (1998), pp. 21–36.

描述 PPT。本文提出的 k-CBP 与 k-DOP 不同之处在于：(1) k-DOP 中 k 值通常仅局限于少数几个，例如 k = 6,8,14,18,20,26 等，后面那篇文献最大支持 k = 46），且 k-DOP 中方向是成对平行的，k 值是偶数，而 k-CBP 中的 k 理论上可以是任意的，奇偶都可以；(2) k-DOP 中的凸多面体方向是固定的即 k 值确定之后，不管输入模型的点集分布如何，k-DOP 的方向始终保持一致，本文则提出了一种能够自适应模型的算法使得构造的凸包围多面体足够紧致；(3) k-CBP 中 k 取值灵活，必须找出一种算法生成 k 个法向，比只有少数几个可直接通过枚举方向的 k-DOP 取值更复杂。通过更加紧致的 k-CBP 去近似原始模型，能够达到更好的精度，且输入 k 值可以自由控制，为其能够用于碰撞检测等应用提供足够大的灵活性。

目录

碰撞检测算法

碰撞检测算法

许多应用的基础，例如在 3D 游戏，物理仿真，机器人，虚拟现实等领域中[2]。

分类

加速结构： SPT（如四叉树、KD 树等） v.s **BVH**（OBB 树、k-DOP 树等）

表现形式： **刚体** v.s 可变形，凸体 v.s 凹体，CSG v.s 参数曲面 v.s **多边形网格**

碰撞环境： **成对** v.s **多体**，**静止** v.s **运动**，**离散** v.s 连续

[2] Christer Ericson. *Real-time collision detection*. San Francisco, CA: Morgan Kaufmann Publishers, 2005.

(介绍 PPT 后)，本文后面的实验就是基于 BVH 的，不可变形的三角网格。//现在研究较多的是连续的可变形的碰撞检测布料模拟头发模拟等。

基于 BVH 的碰撞检测算法

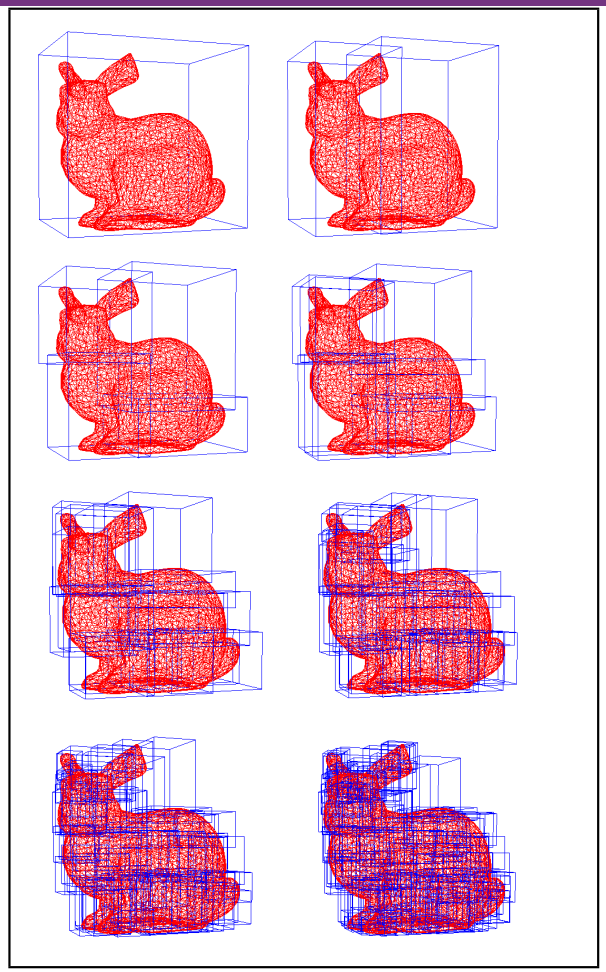


Figure 2: 八层 BVH 示例

算法 1 自顶向下层次遍历 BVH

输入: 两个 BVH 树的根节点 $node_1$，$node_2$
输出: 模型是否相交

```
1:  function TraverseBVHTree(node1, node2)
2:      if node1.bv ∩ node2.bv = ∅ then
3:          return False  // 包围体重合测试, 包围体不相交直接返回
4:      else
5:          if node1.children = ∅ then
6:              if node2.children = ∅ then
7:                  // 最底层叶子节点原生几何相交测试
8:                  return CheckIntersection(node1.primitives, node2.primitives)
9:              else
10:                 for all child ∈ node2.children do
11:                     TraverseBVHTree(node1, child)  // 递归调用
12:                 end for
13:             end if
14:         else
15:             for all child ∈ node1.children do
16:                 TraverseBVHTree(child, node2)  // 递归调用
17:             end for
18:         end if
19:     end if
20: end function
```

代价函数： $T_{cost} = n_v * C_v + n_p * C_p + (n_u * C_u)$(运动)

基于 BVH 的碰撞检测算法

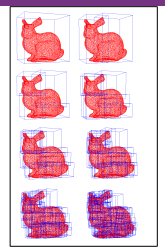


Figure 2: 八层 BVH 示例

算法 1 自顶向下层次遍历 BVH

输入: 两个 BVH 树的根节点 $node_1$，$node_2$
输出: 模型是否相交

```
1:  function TraverseBVHTree(node_1, node_2)
2:      if node_1.bv ∩ node_2.bv = ∅ then
3:          return False  // 包围体重合测试, 包围体不相交直接返回
4:      else
5:          if node_1.children = ∅ then
6:              if node_2.children = ∅ then
7:                  // 最底层叶子节点做生几何相交测试
8:                  return CheckIntersection(node_1.primitives, node_2.primitives)
9:              else
10:                 for all child ∈ node_2.children do
11:                     TraverseBVHTree(node_1, child)  // 递归调用
12:                 end for
13:             end if
14:         else
15:             for all child ∈ node_1.children do
16:                 TraverseBVHTree(child, node_2)  // 递归调用
17:             end for
18:         end if
19:     end if
20: end function
```

代价函数：$T_{cost} = n_v * C_v + n_p * C_p + (n_u * C_u)$(运动)

基于包围体树的碰撞检测算法，一般首先都会初始化环境然后构建层次结构的包围体树，碰撞检测时从顶层开始逐渐往下层遍历，到最底层叶子节点后开始三角网格模型相交测试，当发现三角网格相交后立即终止遍历，确定模型发生碰撞。评价碰撞检测算法的指标一般用上面这个公式来衡量，其中 nv 和 np 分别表示参与包围体节点相交测试的数量和参与原始几何相交测试的数量，Cv 和 Cp 则表示相应的平均测试耗费的代价。当在运动场景时还需要加上 nu 和 Cn 就是模型旋转或者运动后包围体更新的数量和更新的代价。本文算法就是尽早发现包围体不相交的情况，减少 np 和 cp 的数量。

目录

问题的定义

凸包围 k 面体

k-Convex Bounding Polyhedron，简称 k-CBP，可通过 k 个半空间定义：

$$\begin{cases} k\text{-CBP} = \bigcap_{i=1}^{k} \boldsymbol{H_i} \\ \boldsymbol{H_i} = \left\{ \boldsymbol{p} \in \mathbb{R}^3 \middle| \boldsymbol{n_i} \cdot \boldsymbol{p} \leq w_i, w_i \in \mathbb{R} \right\}, \end{cases} \tag{1}$$

其中，$\boldsymbol{n_i}$ 是半空间 $\boldsymbol{H_i}$ 的法向，方向指向包围体外部，w_i 是输入点集中沿 $\boldsymbol{n_i}$ 方向投影的最大值。

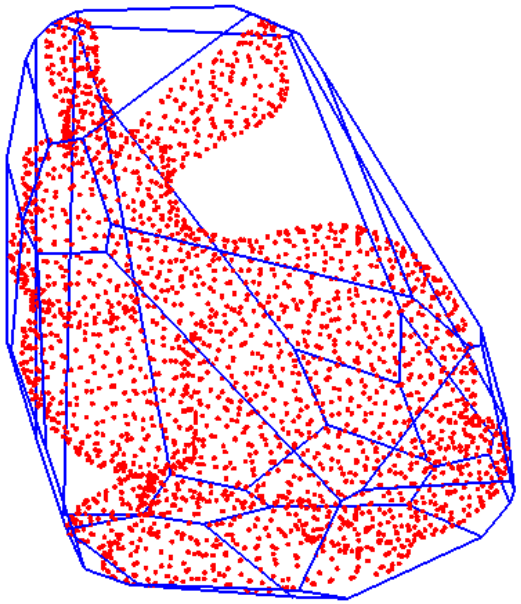


Figure 3: 34-CBP

问题的定义

凸包围 k 面体

k-Convex Bounding Polyhedron，简称 k-CBP，可通过 k 个半空间定义：

$$\begin{cases} k\text{-CBP} = \bigcap_{i=1}^{k} \boldsymbol{H_i} \\ \boldsymbol{H_i} = \left\{ \boldsymbol{p} \in \mathbb{R}^3 \middle| \boldsymbol{n_i} \cdot \boldsymbol{p} \leq w_i, w_i \in \mathbb{R} \right\}, \end{cases} \tag{1}$$

其中，$\boldsymbol{n_i}$ 是半空间 $\boldsymbol{H_i}$ 的法向，方向指向包围体外部，w_i 是输入点集中沿 $\boldsymbol{n_i}$ 方向投影的最大值。

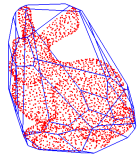
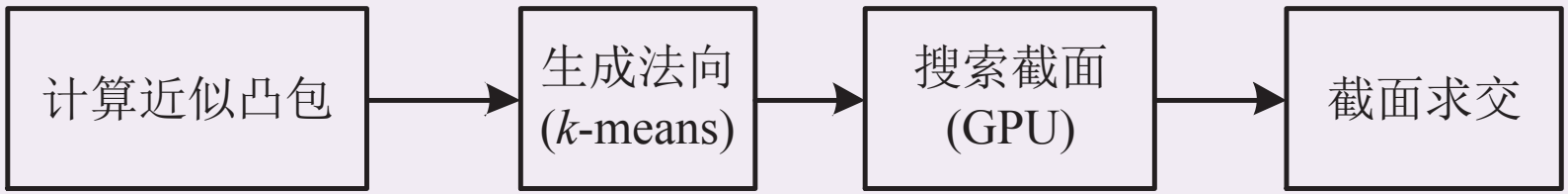


Figure 3: 34-CBP

这是凸包围多面体的定义，通俗些讲，其实就是空间中将模型包裹起来的 k 个平面，右图就是一个 k=34 的例子。其实这个问题的关键就是如何确定这些平面的方向。

算法流程

构造 k-CBP 算法流程图



关键步骤

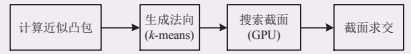
定法向： 结合近似内凸包和 k-means；

搜截面： GPU 中沿各法向搜索切点构造截面；

求交点： 截面对偶映射求得交点。

算法流程

构造 k-CBP 算法流程图



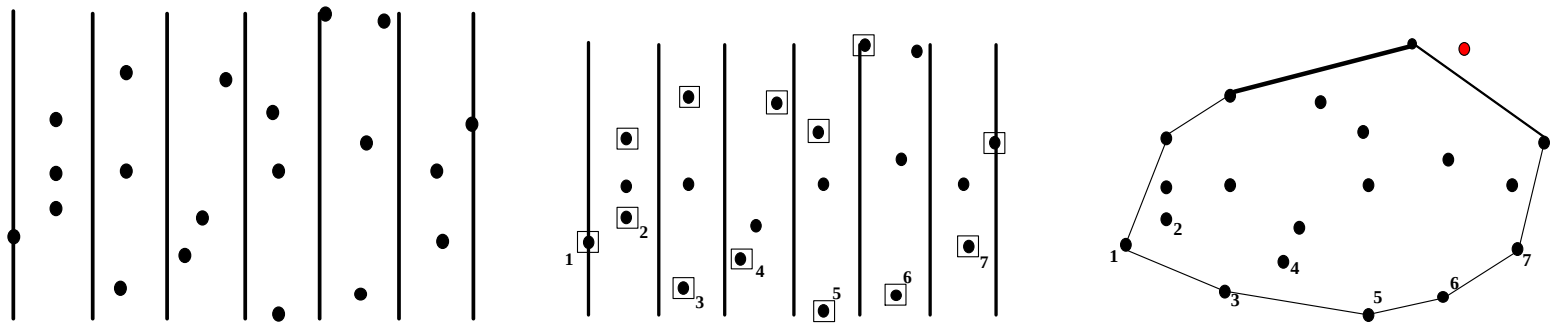
关键步骤

定法向： 结合近似内凸包和 k-means；
搜截面： GPU 中沿各法向搜索切点构造截面；
求交点： 截面对偶映射求得交点。

这是本文构造凸包围多面体的算法流程图，关键就是这 3 步。
定法向：方向的好坏直接决定最后多面体的紧致程度，本文结合了近似内凸包和 kmeans 聚类算法来确定法向。
搜截面：因为法向确定后，搜索截面的过程各个法向之间相互独立互不影响，因此可以较方便用 GPU 加速。
求交点：确定好平面后直接求交即可得到 k-cbp 的顶点。

目录

近似凸包的构造

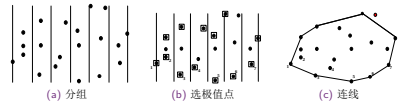


(a) 分组 (b) 选极值点 (c) 连线

Figure 4: 二维近似内凸包的构造

构造近似内凸包[3]，算法复杂度为 $O(n+\lambda)$，扩展到三维为 $O(n+\lambda^2 \log \lambda)$，然后利用 k-means 聚类。

[3] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. "Approximation algorithms for convex hulls". In: *Communications of the ACM* 25.1 (1982), pp. 64–68.



(a) 分组 (b) 选极值点 (c) 连线

Figure 4: 二维近似内凸包的构造

构造近似内凸包[3]，算法复杂度为 $O(n+\lambda)$，扩展到三维为 $O(n+\lambda^2\log\lambda)$，然后利用 k-means 聚类。

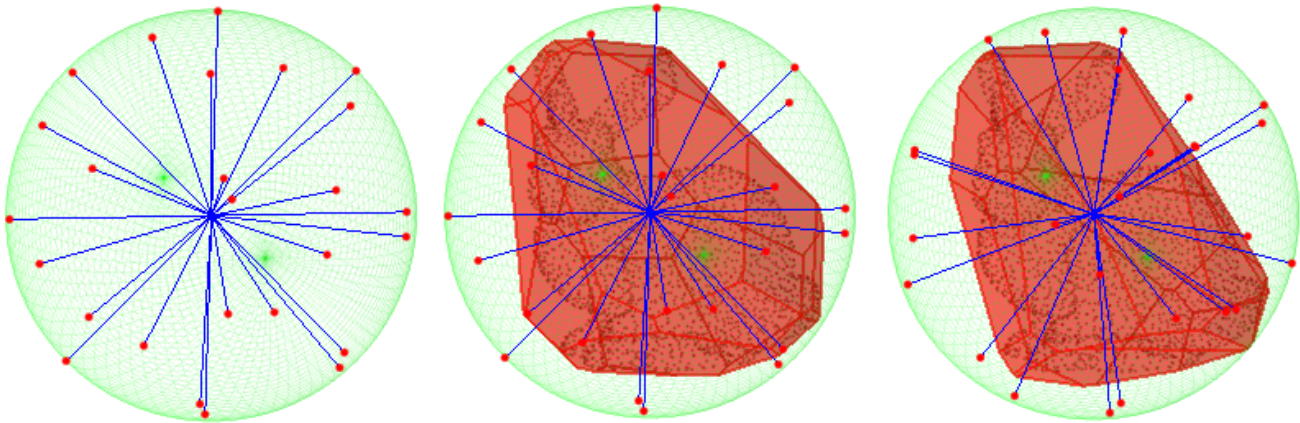
[3] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. "Approximation algorithms for convex hulls". In: *Communications of the ACM* 25.1 (1982), pp. 64–68.

以二维为例，近似内凸包构造分为 3 个步骤，先分组，然后选出每组的极值点，最后将极值点按条件连接起来即可。此算法时间复杂度为 $O(n+\lambda)$，λ 为分的组数。

三维情况类似。

得到近似凸包后，根据凸包面片的方向作为参与 kmeans 聚类算法的点集。

k-means 聚类



(a) (b) (c)

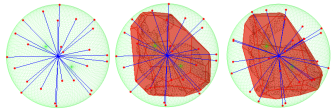
Figure 5: 通过聚类确定法向

初始方向： 均匀分布；

距离度量： 余弦；

中心更新： 中心点：$\boldsymbol{c_i} = \frac{\sum_{i=1}^{i=n} \omega_i \cdot \boldsymbol{n_i}}{\sum_{i=1}^{i=n} \omega_i}$，权重 ω_i 为面片面积。

k-means 聚类



(a) (b) (c)

Figure 5: 通过聚类确定法向

初始方向： 均匀分布；
距离度量： 余弦；
中心更新： 中心点：$\boldsymbol{c}_i = \frac{\sum_{j=1}^{j=n} \omega_j \cdot \boldsymbol{n}_j}{\sum_{j=1}^{j=n} \omega_j}$，权重 ω_i 为面片面积。

聚类算法需要关注下面 3 个问题，一是初始的方向，本文的算法采用均匀分布的方式确定，如图 a，在球面上均匀分布 k 个点，球心和点的连线作为初始方向，图中 k=26。

然后是类与类之间的距离如何度量，用余弦的方式可以使方向相近的点聚到一类。

中心点的更新，以面片所在方向的面积作为权重，使得最后结果靠近面片较大的方向。图 c 为聚类后的结果，比图 b 紧致 15% 左右。

目录

搜索截面

截面＝法向 + 点

法向已得，求投影点：对每个法向 $\boldsymbol{n_i}$，从输入模型的所有点中寻找最大投影值的点作为切点。时间复杂度为 $O(k \cdot n)$，其中 k 为法向数量，n 为模型所含点数。

并行可行性

各法向的计算相互独立，借助 GPU 并行加速。典型 GPU 并行平台：着色器（GLSL 为例）和基于 GPU 的通用计算框架（CUDA 为例）。

截面=法向 + 点

法向已得，求投影点：对每个法向 $\boldsymbol{n}_i$，从输入模型的所有点中寻找最大投影值的点作为切点。时间复杂度为 $O(k \cdot n)$，其中 k 为法向数量，n 为模型所含点数。

并行可行性

各法向的计算相互独立，借助 GPU 并行加速。典型 GPU 并行平台：着色器（GLSL 为例）和基于 GPU 的通用计算框架（CUDA 为例）。

方向确定后，直接沿着方向搜索切点即可。因为各方向搜索过程相互独立，所以可借助 GPU 加速，本文采用两种典型的 GPU 加速平台，一种是 OpenGL 着色语言为代表的着色器平台，另一种是 GPGPU 通用计算框架 CUDA。

GLSL 实现

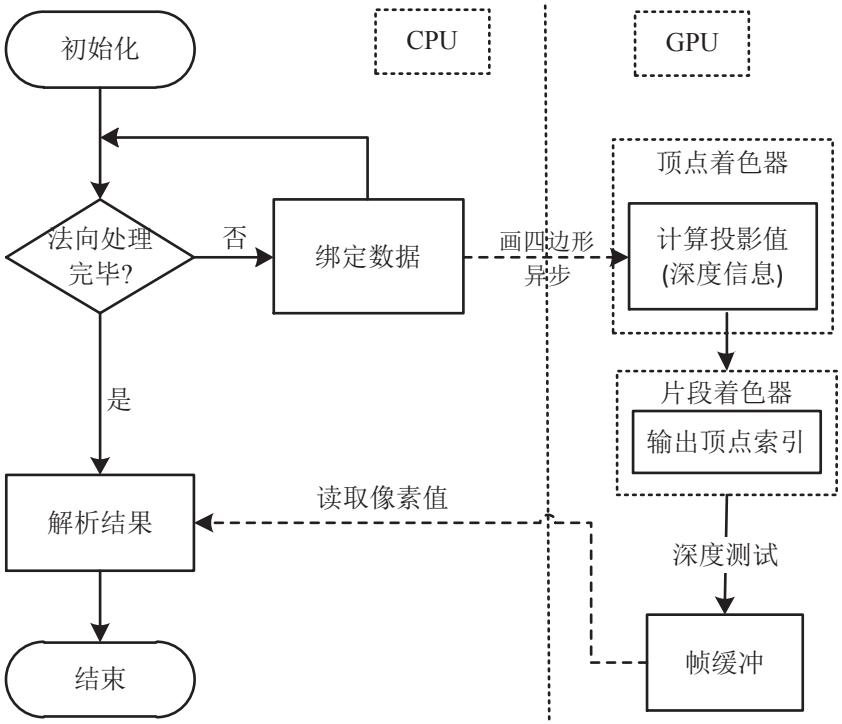


Figure 6: 基于 Z Buffer 算法流程图

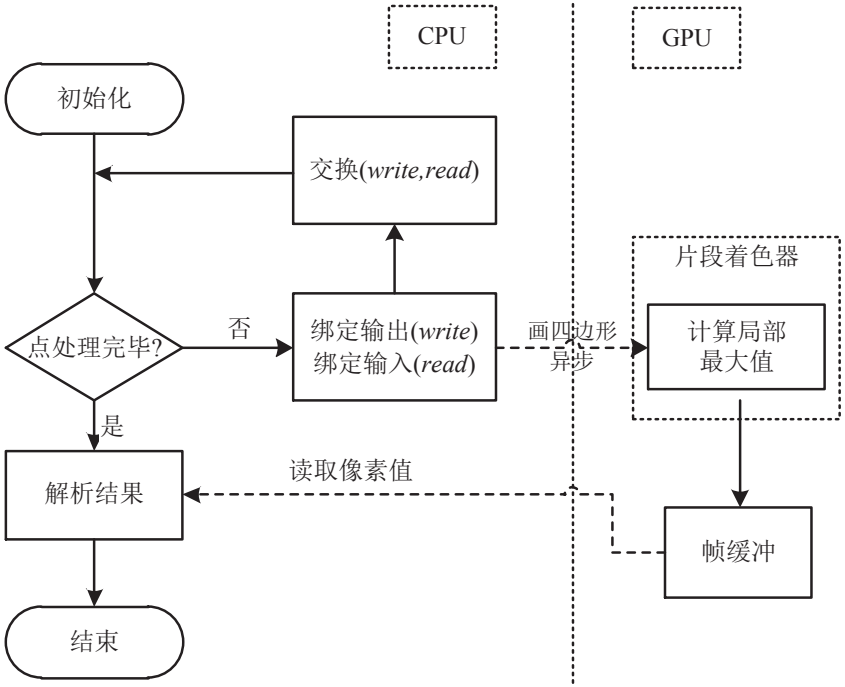


Figure 7: 基于乒乓技术算法流程图

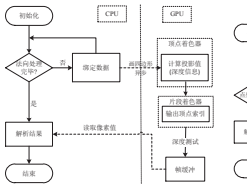


Figure 6: 基于 Z Buffer 算法流程图

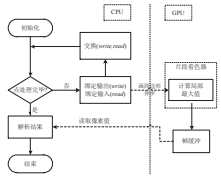


Figure 7: 基于乒乓技术算法流程图

GLSL 中，本文提出了两种算法，一种是利用 OpenGL 的深度测试，深度测试实际上是比较三维点中 z 值的大小，在顶点着色器中，将所有点的 x,y 坐标值设为一样，并将该点在法向上的投影作为 z 值即深度值，所有点经过深度测试后将只有 z 值最大的点被保留下来，该点就是沿着这个法向的切点。这样经过 k 次渲染即可找到所有的 k 个切点。

另外一种是乒乓技术，乒乓技术是需要把前一次运算结果传递给下一次运算用来作为后继运算的输。算法分批处理输入点集，每次计算能得到 k 个局部投影最大值，当所有点都被处理完毕后得到沿着所有法向投影最大值的点即切点。所以这个算法对 k 不是很敏感，因为 k 相对较小时，GPU 一次总能处理 k 个方向，这点从后面的实验结果也能看出。

CUDA 实现

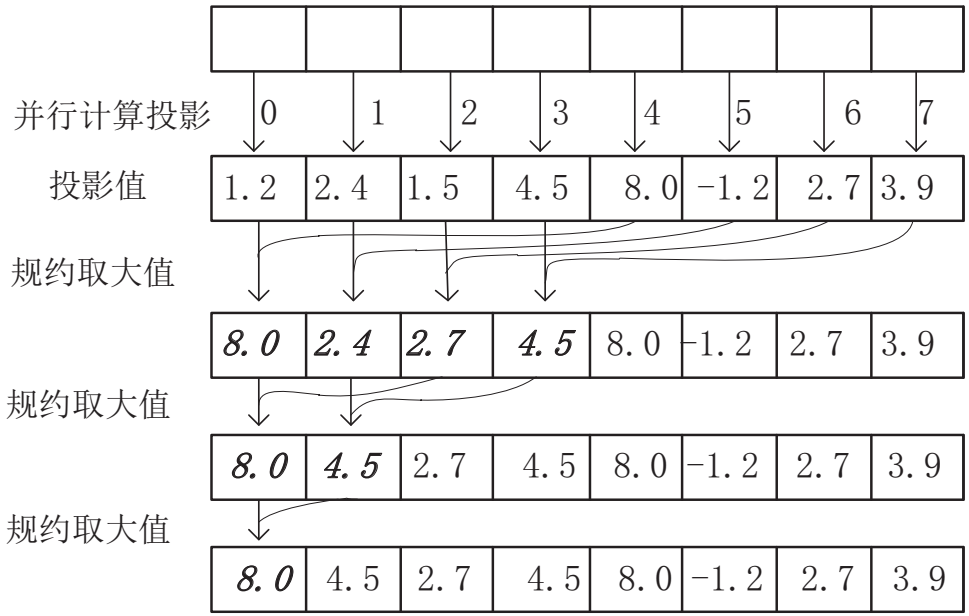


Figure 8: 并行规约求最大投影值

将输入点交给数量为 t 的线程计算点积得到投影值，线程 i 和 $i+t/2$ 比较选取较大者，经 $\log_2 t$ 次比较可得最大值。OpenCL 等并行计算框架类似。

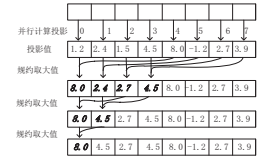


Figure 8: 并行规约求最大投影值

将输入点交给数量为 t 的线程计算点积得到投影值，线程 i 和 $i+t/2$ 比较选取较大者，经 $\log_2 t$ 次比较可得最大值。OpenCL 等并行计算框架类似。

CUDA 实现中原理如上图所示，将输入点交给数量为 t 的线程计算点积得到投影值，线程 i 和 $i+t/2$ 比较选取较大者，经 $\log_2 t$ 次比较可得最大值。这种规约方式在其他 GPGPU 框架中也类似，如 OpenCL 等。

求交算法

直接枚举

通过枚举所有每 3 个平面交于 1 点的情况，然后排除在平面外部的交点，剩下的构成 k-CBP 的顶点，时间复杂度为 $O(k^3)$。

对偶映射

法向 $\boldsymbol{n}(a,b,c)$ + 平面上点 $\boldsymbol{p}(x_0,y_0,z_0) \Rightarrow$ 平面方程 $ax+by+cz=ax_0+by_0+cz_0=d,\ d\neq 0$
对偶点为 $\boldsymbol{p'}(a/d,b/d,c/d)$，对 k 个映射点求凸包，凸包平面映射回原来的交点，时间复杂度为 $O(k\log k)$。[4]

[4] 邓俊辉. 计算几何-算法与应用. 北京: 清华大学出版社, 2005.

直接枚举

通过枚举所有每 3 个平面交于 1 点的情况，然后排除在平面外部的交点，剩下的构成 k-CBP 的顶点，时间复杂度为 $O(k^3)$。

对偶映射

法向 $\boldsymbol{n}(a,b,c)$ + 平面上点 $\boldsymbol{p}(x_0,y_0,z_0) \Rightarrow$ 平面方程 $ax+by+cz=ax_0+by_0+cz_0=d,\ d\neq 0$
对偶点为 $\boldsymbol{p}'(a/d,b/d,c/d)$，对 k 个映射点求凸包，凸包平面映射回原来的交点，时间复杂度为 $O(k\log k)$。[4]

[4] 邓俊辉. 计算几何-算法与应用. 北京: 清华大学出版社, 2005.

确定平面后，接下来就是求每个平面上的交点。一种比较直接的算法是通过枚举所有每 3 个平面交于 1 点的情况，然后排除在平面外部的交点，剩下的构成 k-CBP 的顶点，时间复杂度为 $O(k^3)$。通过计算几何中的对偶映射技术可以将复杂度降低到 $O(k\log k)$。

目录

生成 k-CBP 的效率：GLSL 实验结果

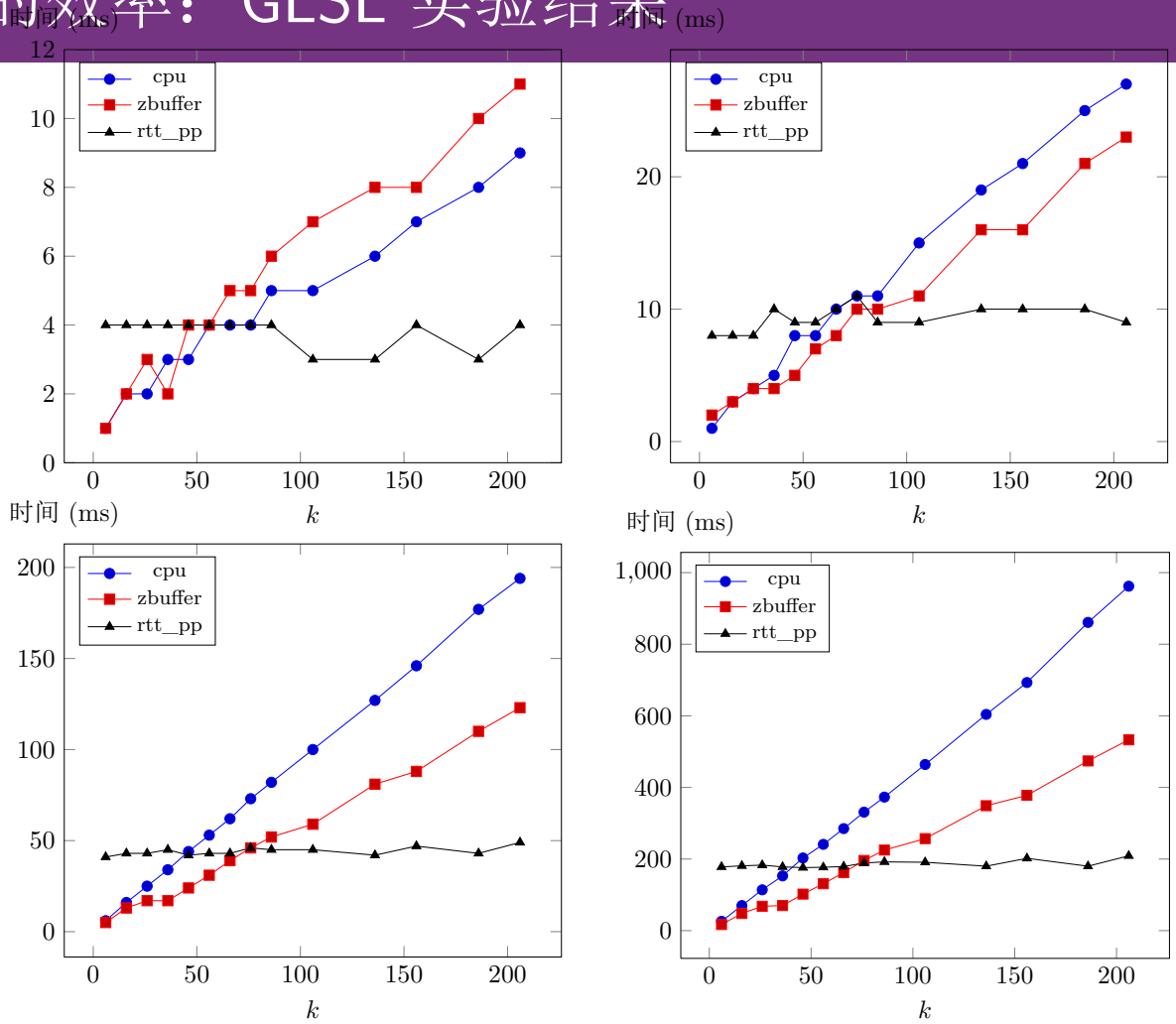


Figure 9: Apple(8k), Buddha(31k), Alice(224k), Bugatti(1011k)

生成 k-CBP 的效率：GLSL 实验结果

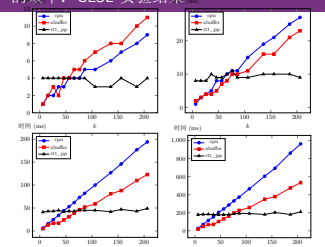


Figure 9: Apple(8k), Buddha(31k), Alice(224k), Bugatti(1011k)

这是基于 OpenGL 着色语言实现的实验结果。从实验结果可以看出，当模型规模不大时，如左上图所示的含有 8 千多个点的 Apple 模型，Z Buffer 算法和传统的 CPU 算法差别不是很大，因此在实际应用中当模型规模较小时可直接用 CPU 计算即可。随着输入模型所含有点的数量规模的增加，CPU 和 GPU 运行时间之间的差距也越来越大。当多面体面数增加即 k 的增大时，在 Z Buffer 算法中，需要更多的绘制次数，因此其运行时间也有所增加，而在基于乒乓技术的算法中，当点规模一定时，k 的变化对最后运行时间影响不明显，因此当较大的 k 时，这种算法更快。

生成 k-CBP 的效率：CUDA 实验结果

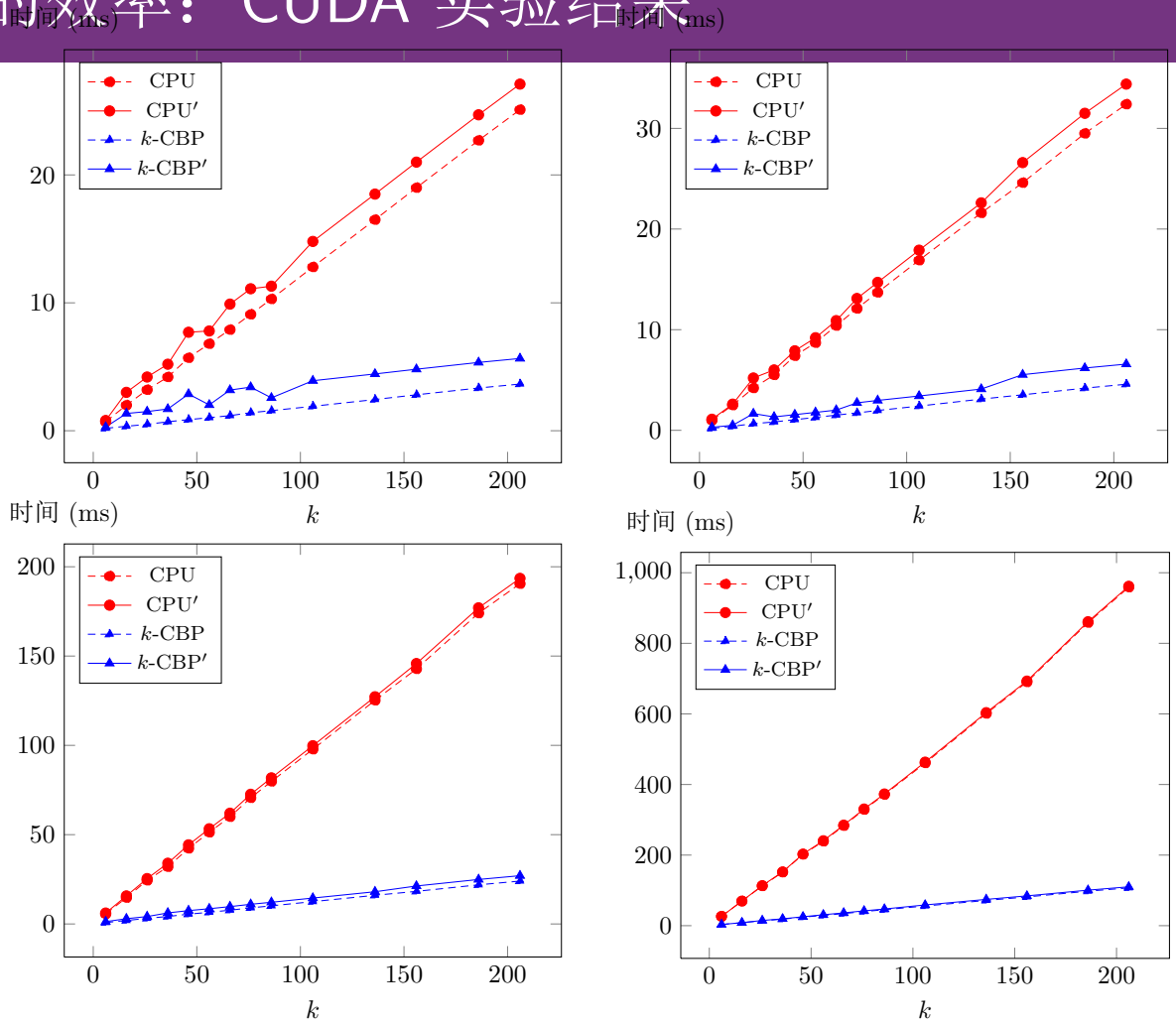


Figure 10: Budda(31k),Dinosaur(40k),Alice(224k), Bugatti(1011k)

生成 k-CBP 的效率：CUDA 实验结果

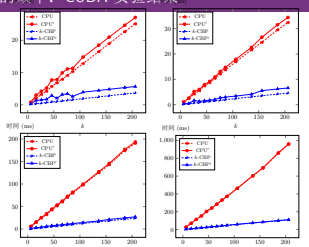


Figure 10: Budda(31k),Dinosaur(40k),Alice(224k), Bugatti(1011k)

这是 CUDA 的实验结果，图中横纵坐标分别代表多面体面数和运行时间, 其中虚线代表搜索截面的过程, 实线为构造凸包围多面体总体耗时. 当模型点数量较大时, 搜索截面的过程占据了算法绝大多数时间, 且随着凸包围多面体的面数 k 值增加而线性增长, 这与搜索截面时间复杂度 $(O(k \cdot n))$ 一致, 截面求交过程的时间复杂度为 $O(k \log k)$, 当点数量极大时, 实线虚线几乎重合即求交等步骤耗时相比整体算法而言几乎可忽略.

生成 k-CBP 的效率：与文献 [5] 算法对比

Table 1: 本文算法与文献[5]算法对比

k	Apple(8118 points)			Bugatti(1010815 points)		
	文献 [5](ms)	k-CBP(ms)	Speedup	文献 [5](ms)	k-CBP(ms)	Speedup
6	0.4	0.12	3.20	24.2	3.20	7.56
16	0.9	0.26	3.43	44.5	8.44	5.27
26	1.4	0.41	3.38	66.5	13.65	4.87
36	1.9	0.52	3.65	91.1	18.34	4.97
46	2.5	0.67	3.74	119.5	24.13	4.95
56	2.9	0.79	3.66	138.4	28.86	4.80
66	3.5	0.95	3.69	170.6	34.10	5.00
76	4.0	1.08	3.70	197.1	39.85	4.95
86	4.5	1.22	3.69	219.8	45.08	4.88
106	5.4	1.49	3.62	267.8	55.52	4.82
136	6.8	1.92	3.54	342.9	71.24	4.81
156	7.7	2.17	3.55	411.3	81.18	5.07
186	9.3	2.60	3.58	479.4	97.39	4.92
206	10.5	2.85	3.68	523.0	106.87	4.89

当点数量较小时，能够提高 3-4 倍速度，模型变大，加速比更大，Bugatti 模型的提速达到 4∼8 倍。

[5] Mattias Karlsson, Olov Winberg, and Thomas Larsson. "Parallel Construction of Bounding Volumes". In: *The Annual Swedish Computer Graphics Association Conference(SIGRAD)*. 2010, pp. 65–69.

生成 k-CBP 的效率：与文献 [5] 算法对比

Table 1: 本文算法与文献[5]算法对比

k	Apple(8118 points)			Bugatti(1010815 points)		
	文献 [5](ms)	k-CBP(ms)	Speedup	文献 [5](ms)	k-CBP(ms)	Speedup
6	0.4	0.12	3.20	24.2	3.20	7.56
16	0.9	0.26	3.43	44.5	8.44	5.27
26	1.4	0.41	3.38	66.5	13.65	4.87
36	1.9	0.52	3.65	91.1	18.34	4.97
46	2.5	0.67	3.74	119.5	24.13	4.95
56	2.9	0.79	3.66	138.4	28.86	4.80
66	3.5	0.95	3.69	170.6	34.10	5.00
76	4.0	1.08	3.70	197.1	39.85	4.95
86	4.5	1.22	3.69	219.8	45.08	4.88
106	5.4	1.49	3.62	267.8	55.52	4.82
136	6.8	1.92	3.54	342.9	71.24	4.81
156	7.7	2.17	3.55	411.3	81.18	5.07
186	9.3	2.60	3.58	479.4	97.39	4.92
206	10.5	2.85	3.68	523.0	106.87	4.89

当点数量较小时，能够提高 3-4 倍速度，模型变大，加速比更大，Bugatti 模型的提速达到 4~8 倍。

[5] Mattias Karlsson, Olov Winberg, and Thomas Larsson. "Parallel Construction of Bounding Volumes". In: *The Annual Swedish Computer Graphics Association Conference(SIGRAD)*. 2010, pp. 65–69.

这是和一篇专门讲并行求包围体的算法的论文的对比结果（Swedish 瑞典，sigrad 会是 Eurographics 的子会议议题官网说的是 the Swedish Chapter of Eurographics），可以看出当点数量较小时，能够提高 3-4 倍速度，模型变大，加速比更大，Bugatti 模型的提速达到 4∼8 倍。

生成 k-CBP 的紧致程度：k-DOP v.s k-CBP

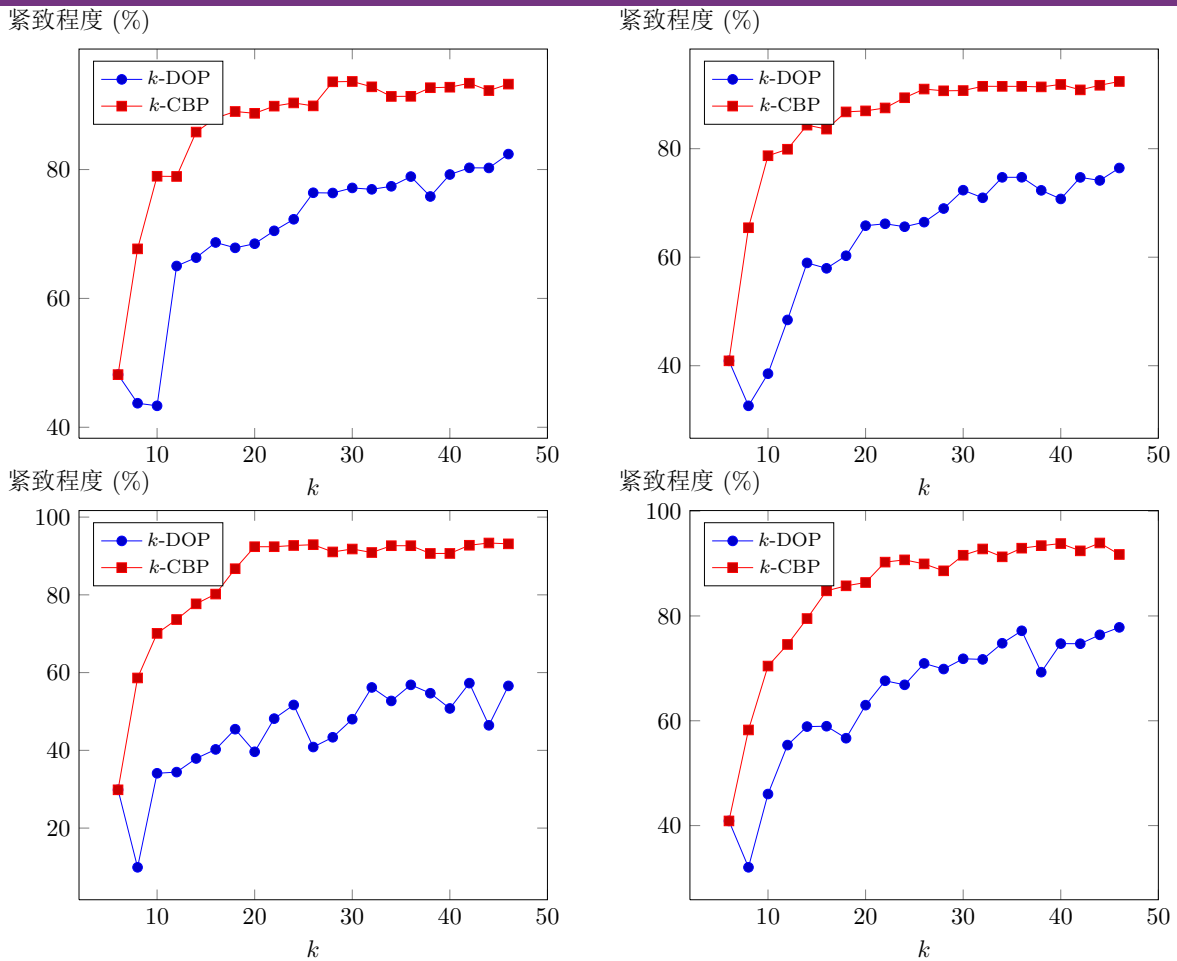


Figure 11: 紧致程度对比：Apple(8k), Budda(31k), Dinosaur(40k), Alice(224k)

生成 k-CBP 的紧致程度：k-DOP v.s k-CBP

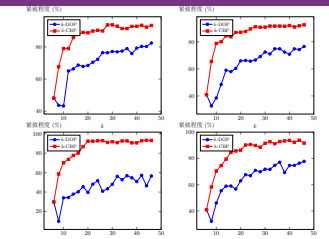


Figure 11: 紧致程度对比：Apple(8k), Budda(31k), Dinosaur(40k), Alice(224k)

这是本文算法和 k-DOP 的对比，较 k-DOP 提升 10% ∼ 40%，下图可视化结果。

生成 k-CBP 的紧致程度：k-DOP v.s k-CBP

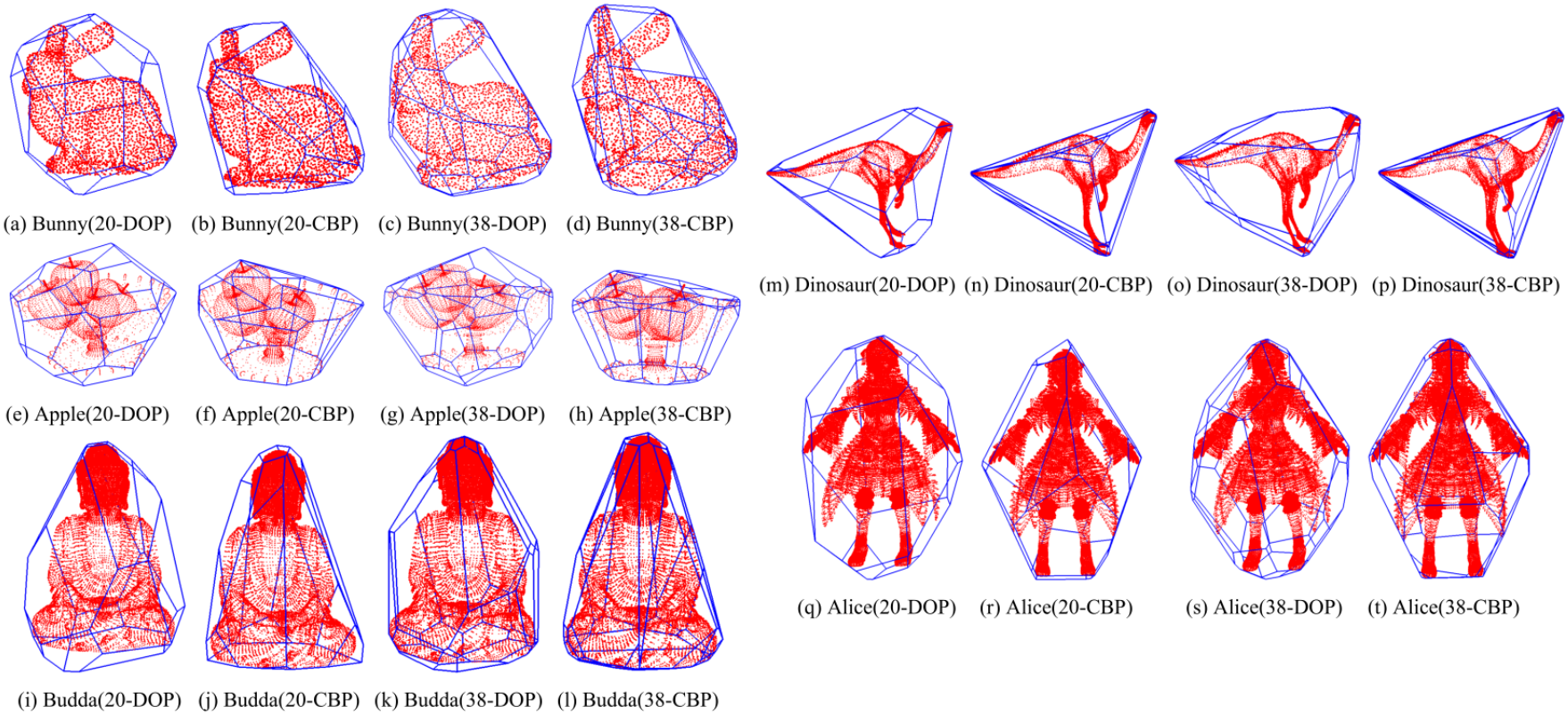


Figure 12: k-CBP 与 k-DOP 对比

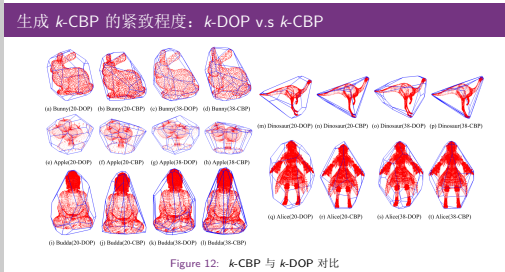


Figure 12: k-CBP 与 k-DOP 对比

该图中是不同 k 值的对比，左边为 k-DOP 结果，右边为本文结果。从外观上看，确实也紧致了不少。

生成 k-CBP 的紧致程度：k-CBP v.s 凸包

Table 2: k-CBP 与 QuickHull 凸包算法比较

Model	f(CHull)	f(k-CBP)	τ (k-CBP)	t(CHull(ms))	t(k-CBP(ms))
Apple	499	30	93.67%	5.5	1.30
Budda	1608	46	92.39%	21.3	2.86
Dinosaur	1240	44	93.34%	22.6	1.99
Alice	1332	44	93.92%	85.8	8.47
Bugatti	24654	44	95.06%	688.7	25.41

结论

与凸包相比，本文算法在大大简化包围体平面数量的同时能保持较好的紧致程度（Bugatti 凸包面的 0.17% 的达到 95.06% 紧致程度，构造速度快 27 倍），下图为可视化结果。

生成 k-CBP 的紧致程度：k-CBP v.s 凸包

Table 2: k-CBP 与 QuickHull 凸包算法比较

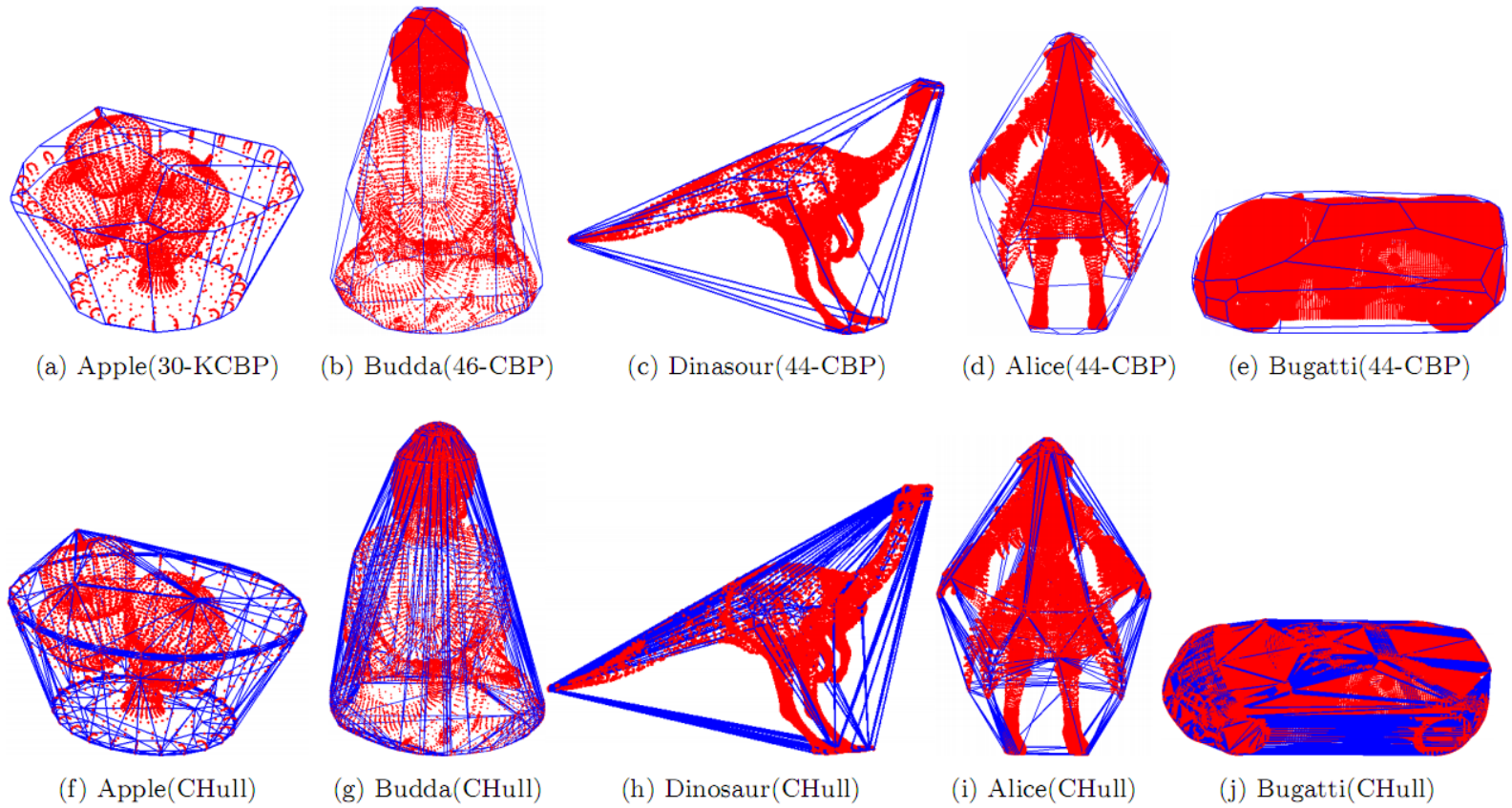
Model	f(CHull)	f(k-CBP)	τ (k-CBP)	t(CHull(ms))	t(k-CBP(ms))
Apple	499	30	93.67%	5.5	1.30
Budda	1608	46	92.39%	21.3	2.86
Dinosaur	1240	44	93.34%	22.6	1.99
Alice	1332	44	93.92%	85.8	8.47
Bugatti	24654	44	95.06%	688.7	25.41

结论

与凸包相比，本文算法在大大简化包围体平面数量的同时能保持较好的紧致程度（Bugatti 凸包面的 0.17% 的达到 95.06% 紧致程度，构造速度快 27 倍），下图为可视化结果。

这是本文算法和 CGAL 中的凸包的对比，其中 f 表示面数量，τ 的值为紧致程度，紧致程度是用凸包的体积除以凸包围多面体的体积来衡量的。与凸包相比，本文算法在大大简化包围体平面数量的同时能保持较好的紧致程度（Bugatti 凸包面的 0.17% 的达到 95.06% 紧致程度，构造速度快 27 倍），下图为可视化结果。

生成 k-CBP 的紧致程度：k-CBP v.s 凸包



(a) Apple(30-KCBP) (b) Budda(46-CBP) (c) Dinasour(44-CBP) (d) Alice(44-CBP) (e) Bugatti(44-CBP)

(f) Apple(CHull) (g) Budda(CHull) (h) Dinosaur(CHull) (i) Alice(CHull) (j) Bugatti(CHull)

Figure 13: k-CBP 与凸包对比

目录

AABB 树法

将生成的 k-CBP 视为普通的三角形，实现简单，适用于模型较小的静止碰撞检测场景。

GJK 法

计算凸多面体之间的最近距离的 GJK 算法[6]。
Minkowski 差，即 $\mathbb{A} - \mathbb{B} = \{\boldsymbol{a} - \boldsymbol{b} | \boldsymbol{a} \in \mathbb{A}, \boldsymbol{b} \in \mathbb{B}\}$。GJK 算法的核心思想在于若两个凸多边形相交，则凸多边形顶点
的 Minkowski 差所围成的多边形必包含原点，因为若 $\mathbb{A}$ 和 $\mathbb{B}$ 相交即 $\mathbb{A}$ 和 $\mathbb{B}$ 必含有公共交集，即至少含有一点同时属
于 $\mathbb{A}$ 和 $\mathbb{B}$，该点的 Minkowski 差即为原点 $\boldsymbol{O}(0, 0)$。

[6] Gino van den Bergen. "A fast and robust GJK implementation for collision detection of convex objects". In: *Journal of Graphics Tools* 4.2 (1999), pp. 7–25.

AABB 树法

将生成的 *k*-CBP 视为普通的三角形，实现简单，适用于模型较小的静止碰撞检测场景。

GJK 法

计算凸多面体之间的最近距离的 GJK 算法[6]。
Minkowski 差，即 $\mathbb{A}-\mathbb{B}=\{\boldsymbol{a}-\boldsymbol{b}|\boldsymbol{a}\in\mathbb{A},\boldsymbol{b}\in\mathbb{B}\}$。GJK 算法的核心思想在于若两个凸多边形相交，则凸多边形顶点的 Minkowski 差所围成的多边形必包含原点，因为若 $\mathbb{A}$ 和 $\mathbb{B}$ 相交即 $\mathbb{A}$ 和 $\mathbb{B}$ 必含有公共交集，即至少含有一点同时属于 $\mathbb{A}$ 和 $\mathbb{B}$，该点的 Minkowski 差即为原点 $\boldsymbol{O}(0,0)$。

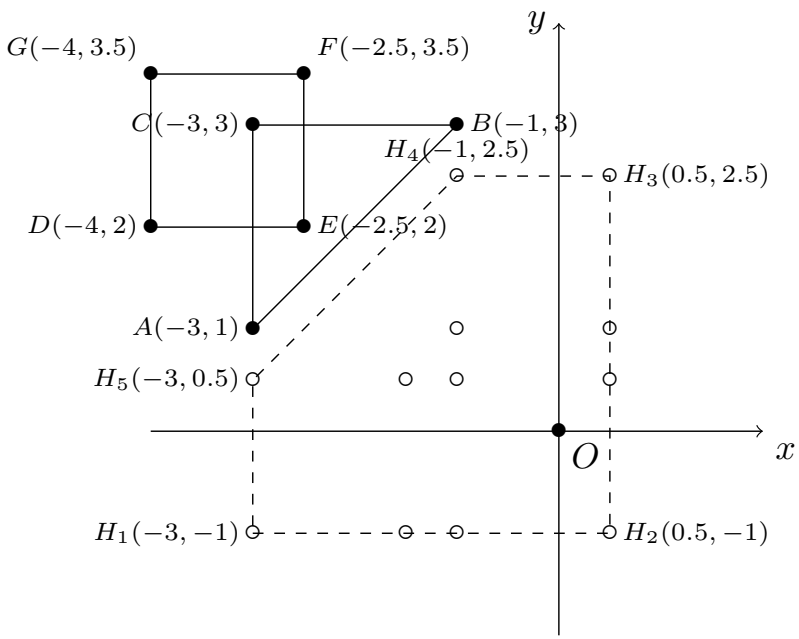
[6] Gino van den Bergen. "A fast and robust GJK implementation for collision detection of convex objects". In: *Journal of Graphics Tools* 4.2 (1999), pp. 7–25.

基于 k-CBP 的碰撞检测问题，首先要解决 k-CBP 间的相交测试，有下面两种方法：

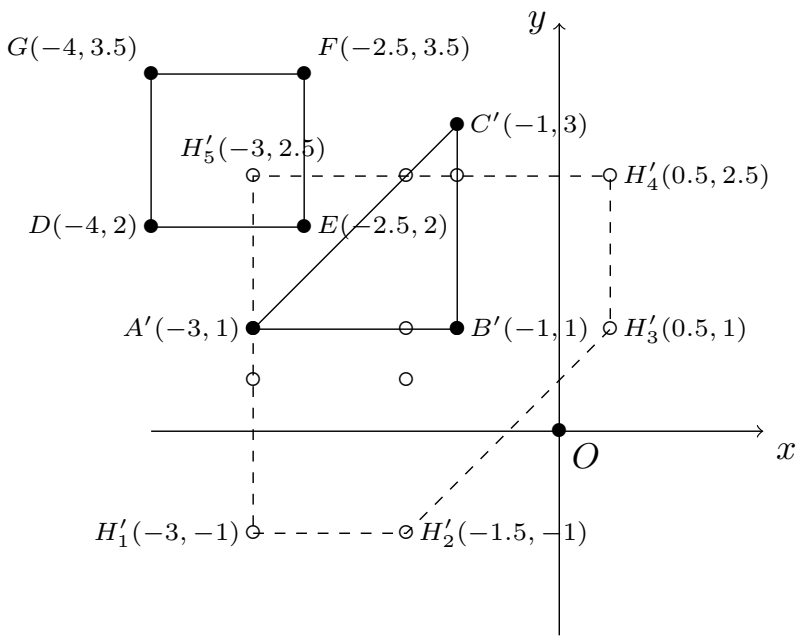
一是将 k-CBP 视为普通的三角网格模型，用传统 AABB 树方法进行相交检测。

二是利用计算凸体之间最近距离的 GJK 算法。该算法利用了 Minkowski（闵可夫斯基）差的概念。核心思想在于若两个凸多边形相交，则凸多边形顶点的 Minkowski 差所围成的多边形必包含原点。

二维 GJK 算法示例

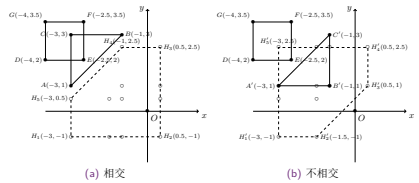


(a) 相交



(b) 不相交

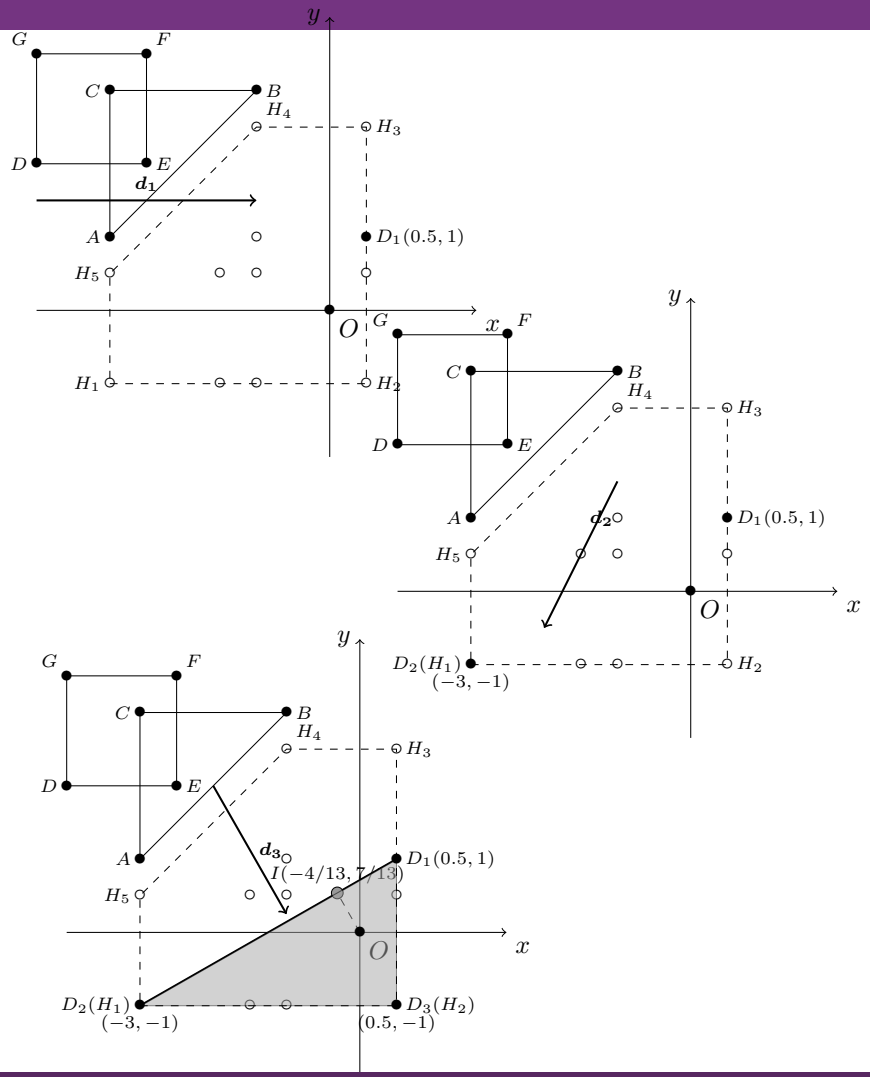
二维 GJK 算法示例



(a) 相交　(b) 不相交

以两个例子来说明，左图四边形和三角形相交，其 Minkowski 差围成的多边形 H1H2...H5 就包含原点。而右边的不相交，其 Minkowski 差围成的多边形不包含原点。

GJK 算法

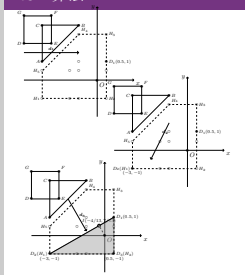


算法 2 基于 GJK 的 k-CBP 相交检测算法

输入: 两个 k-CBP $k\text{-}CBP_1, k\text{-}CBP_2$
输出: k-CBP 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
```

GJK 算法



算法 2 基于 GJK 的 k-CBP 相交检测算法

输入: 两个 k-CBP $k\text{-}CBP_1$, $k\text{-}CBP_2$
输出: k-CBP 是否相交

```
function KCBPDetectionBasedOnGJK(k-CBP1, k-CBP2)
    d ← initNormal()
    D ← Support(k-CBP1, k-CBP2, d)
    S ← {p}
    iter ← 1, d ← −d
    while iter ← iter + 1 < Maxiter do
        D ← Support(k-CBP1, k-CBP2, d)
        if D · d < 0 then
            return False
        end if
        S ← S ∪ D
        contains ← CheckContainAndUpdate(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
        if contains then
            return True  // 包含原点，直接返回相交，否则继续迭代
        end if
    end while
    return False  // 达到最大迭代次数，根据需求返回相交或者不相交
end function
```

接下来就是怎样判断 Minkowski 差包含原点。GJK 算法是一个迭代算法，刚开始取一个方向计算该方向上的支持点（Minkowski 差围成多边形沿该方向的投影点），

通过上一步的结果计算下一步迭代方向，如左图（鼠标操作）第一个支持点 D1，下一次迭代找到 D2，第三次迭代 D3，此时 D1D2D3 就包含了原点。

目录

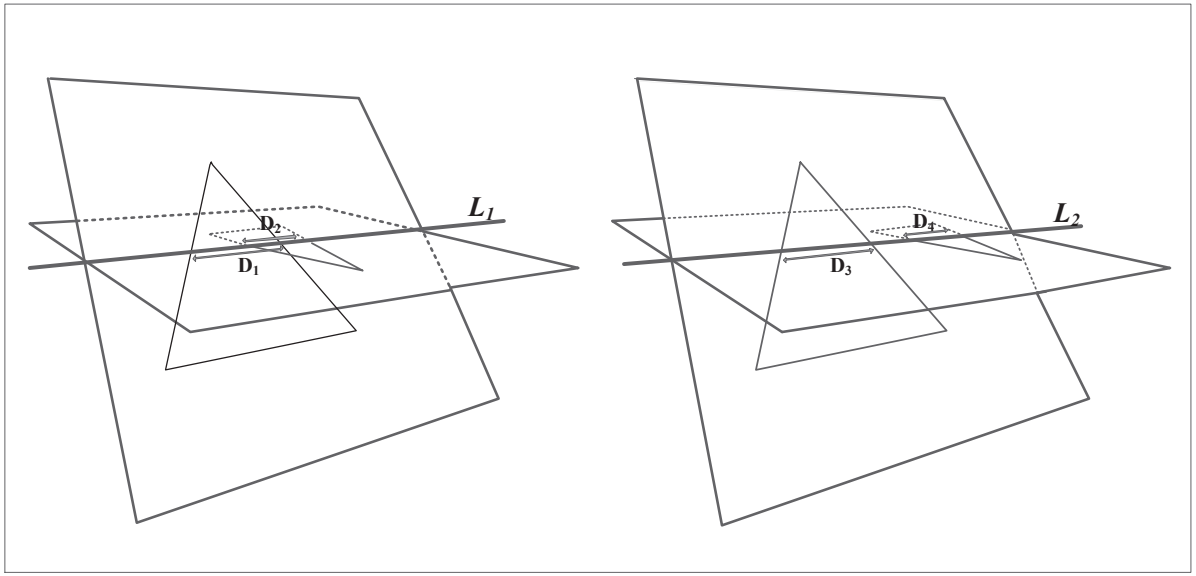


Figure 14: 两个非共面三角形的位置关系

三角形 T_1, T_2 坐标 $\Rightarrow$ 平面方程 $\Pi_1, \Pi_2 \Rightarrow$ T_2 到 Π_1 的有向距离 $l_{1i}, i \in \{1,2,3\}$[7]:

(1) 若 $\forall i \in \{1,2,3\}, l_{1i} = 0$，即三角形 T_2 的三个顶点到三角形 T_1 所在 Π_1 的距离都为 0，则两个三角形共面；$\Rightarrow$ 共面三角形求交。

(2) 若 $\forall i \in \{1,2,3\}, l_{1i} > 0$ 或 $\forall i \in \{1,2,3\}, l_{1i} < 0$，即三角形 T_2 的三个顶点到三角形 T_1 所在 Π_1 的有向距离同号，则 T_2 在 Π_1 的同一侧，可立即排除相交；

(3) 其他，三角形 T_2 必交 Π_1 于一条线段。$\Rightarrow$ 判断两个区间线段 D_1, D_2 是否相交。

[7] Tomas Moller. "A Fast Triangle-Triangle Intersection Test". In: *Journal of Graphics Tools* 2.2 (1997), pp. 25–30.

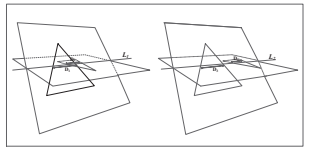


Figure 14: 两个非共面三角形的位置关系

三角形 T_1, T_2 坐标 $\Rightarrow$ 平面方程 $\Pi_1, \Pi_2 \Rightarrow$ T_2 到 Π_1 的有向距离 $I_{1i}, i \in \{1,2,3\}$[7]:

(1) 若 $\forall i \in \{1,2,3\}, I_{1i} = 0$，即三角形 T_2 的三个顶点到三角形 T_1 所在 Π_1 的距离都为 0，则两个三角形共面；$\Rightarrow$ 共面三角形求交。

(2) 若 $\forall i \in \{1,2,3\}, I_{1i} > 0$ 或 $\forall i \in \{1,2,3\}, I_{1i} < 0$，即三角形 T_2 的三个顶点到三角形 T_1 所在 Π_1 的有向距离同号，则 T_2 在 Π_1 的同一侧，可立即排除相交；

(3) 其他，三角形 T_2 必交 Π_1 于一条线段。$\Rightarrow$ 判断两个区间线段 D_1, D_2 是否相交。

[7] Tomas Moller. "A Fast Triangle-Triangle Intersection Test". In: *Journal of Graphics Tools* 2.2 (1997), pp. 25–30.

三角形的相交测试也较简单。因为三角形三个点坐标已知，可推出平面方程，进而得到点到另一个平面的有向距离，根据有向距离分 3 种情况。（1）...（2）...（3）转化为投影到公共交线的区间线段是否相交。

目录

k-CBP 的有效性

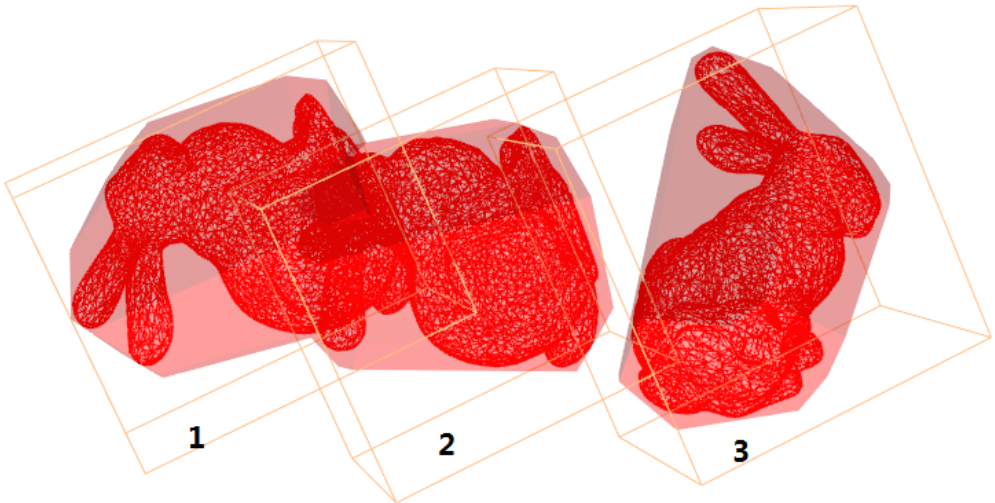


Figure 15: k-CBP 应用于碰撞检测示例

图中模型 1 与 2、2 与 3 的包围盒分别相交，而其 16-CBP 仅 1 与 2 相交，实际模型仅 1 与 2 相交。

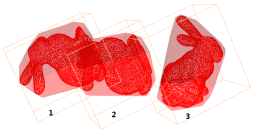


Figure 15: k-CBP 应用于碰撞检测示例

图中模型 1 与 2、2 与 3 的包围盒分别相交，而其 16-CBP 仅 1 与 2 相交，实际模型仅 1 与 2 相交。

这个例子展示了将 k-CBP 应用于碰撞检测的有效性。图中模型 1 与 2、2 与 3 的包围盒分别相交，而其 16-CBP 仅 1 与 2 相交，实际模型仅 1 与 2 相交。

静止场景碰撞检测算法

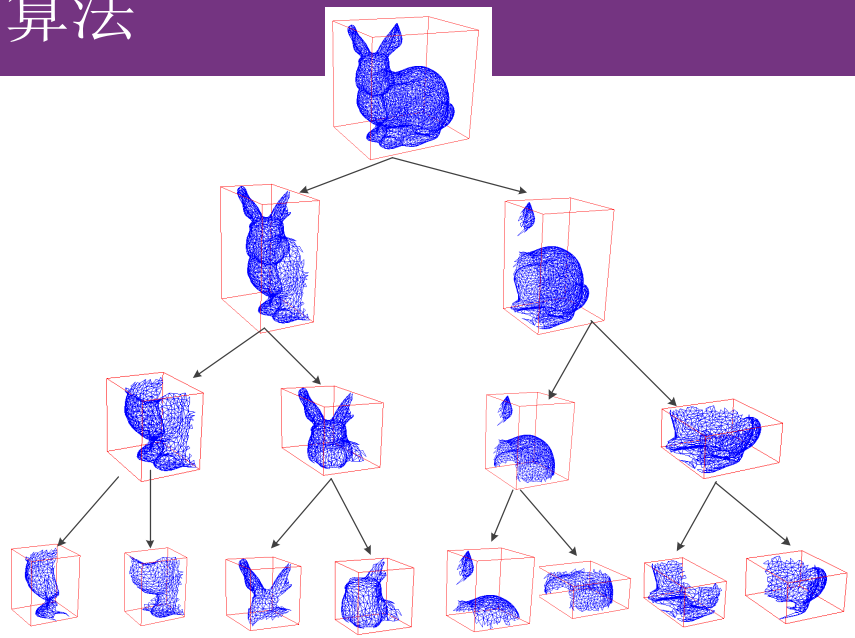


Figure 16: Bunny 模型的 AABB 树形结构 (部分) ▸ 动态图

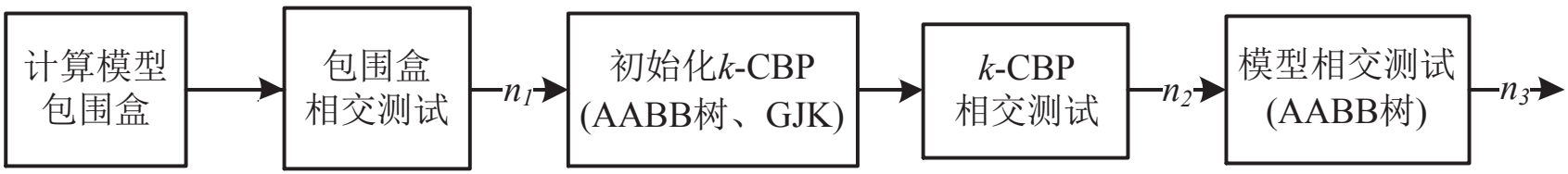


Figure 17: 基于 k-CBP 的碰撞检测算法流程图

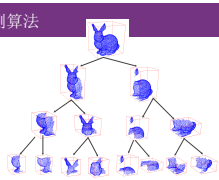


Figure 16: Bunny 模型的 AABB 树形结构 (部分)

Figure 17: 基于 k-CBP 的碰撞检测算法流程图

这是 Bunny 模型的 AABB 树结构，【看动态图】，静止场景下基于 k-CBP 的碰撞检测算法流程图如图 17 所示。

运动场景碰撞检测算法

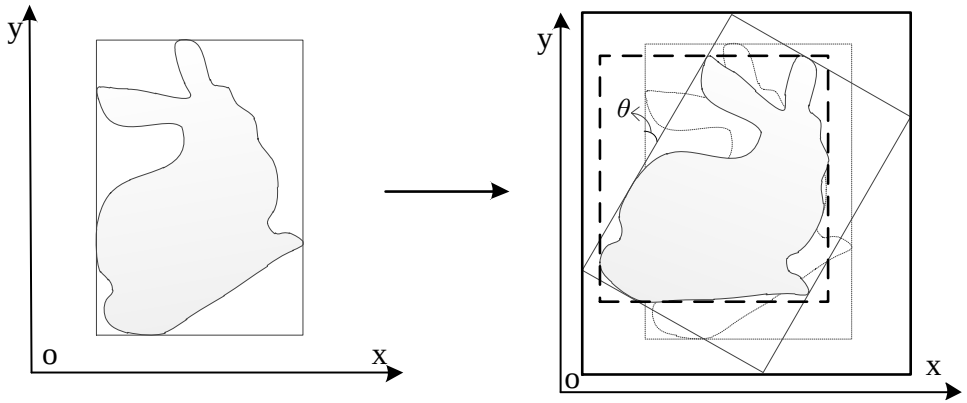


Figure 18: AABB 更新策略图

将变换矩阵 $\boldsymbol{M} = \boldsymbol{R}(\boldsymbol{n}, \theta) \cdot \boldsymbol{T}(\boldsymbol{t})$ 应用于 GJK 顶点、AABB 顶点。

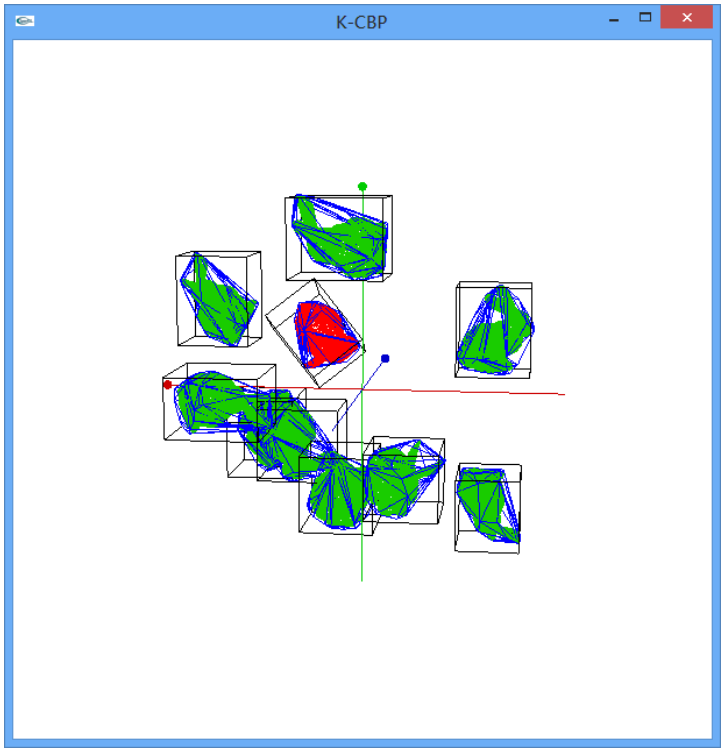


Figure 19: 运动场景碰撞检测示例 ▸ 动态图

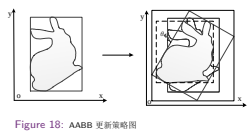


Figure 18: AABB 更新策略图

将变换矩阵 $\boldsymbol{M} = \boldsymbol{R}(\boldsymbol{n}, \theta) \cdot \boldsymbol{T}(\boldsymbol{t})$ 应用于 GJK 顶点、AABB 顶点。

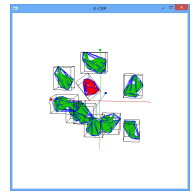


Figure 19: 运动场景碰撞检测示例 动态图

运动场景下需要将变换矩阵应用到 GJK 顶点上，AABB 更新时采取了一种近似策略，原先包围盒的 8 个顶点算出新的包围盒。图 19 是一个运动场景的示例，其中一个模型运动，运动过程中与其他所有模型进行相交检测。【看动态图】

目录

实验结果：k-CBP 用于碰撞检测的有效性

Table 3: 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) (对)
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

其中模型和凸包围多面体是否相交都采用了 AABB 树的方式进行判断。

实验结果：k-CBP 用于碰撞检测的有效性

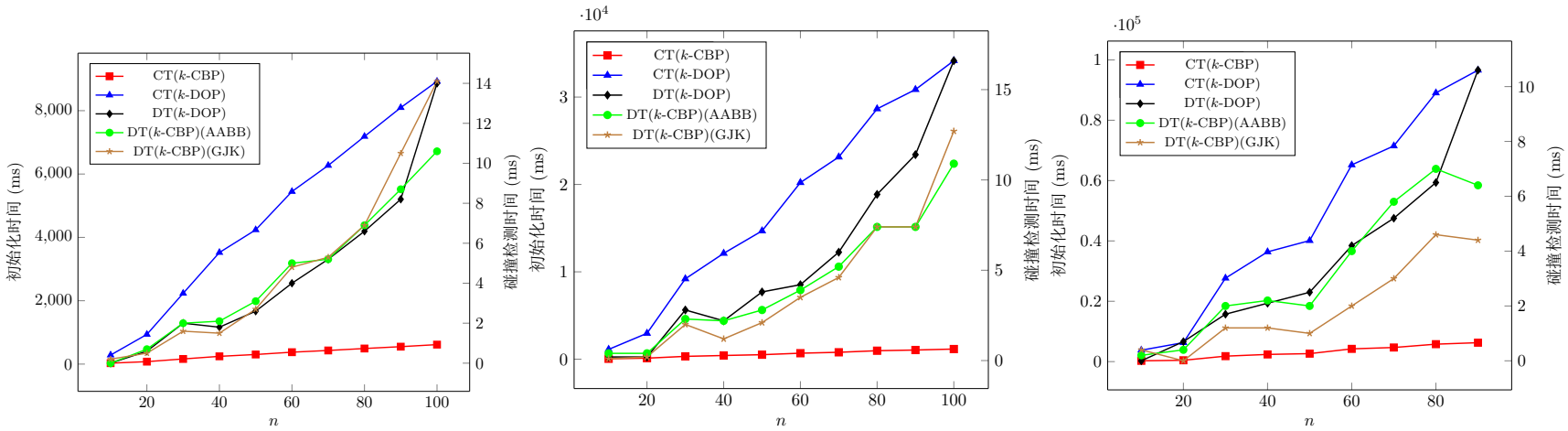
Table 3: k-CBP 和包围盒应用于碰撞检测结果对比

n	CT(Box) (ms)	CT(16-CBP) (ms)	DT(Box) (ms)	DT(16-CBP) (ms)	r(Box) (%)	r(k-CBP) (%)	DP(Model) (对)
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

其中模型和凸包围多面体是否相交都采用了 AABB 树的方式进行判断。

先用一个实验说明 k-CBP 用于碰撞检测的有效性。不能说因为用了 kCBP，反而比直接用包围盒更慢了。表中 n 为场景中模型数量，CT 为构造时间，DT 为场景中碰撞检测时间，r 为命中率即模型实际相交数量/包围体相交数量。

实验结果：静止场景与 k-DOP 树对比



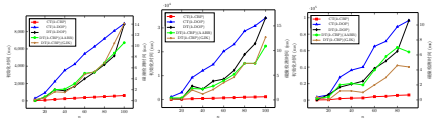
(a) Bunny(4968 个三角形) (b) Apple(8040 个三角形) (c) HappyBuddha(50000 个三角形)

Figure 20: 静止场景下本文算法与基于 k-DOP 树算法实验结果对比（$k = 24$）

基于 k-DOP 树[8]算法的碰撞检测库 ▸ CollDet 是 Gabriel Zachmann 等人实现的。

[8] Sven Trenkel, René Weller, and Gabriel Zachmann. "A benchmarking suite for static collision detection algorithms". In: *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Plzen, Czech Republic: Union Agency, 2007.

实验结果：静止场景与 k-DOP 树对比



(a) Bunny(4968 个三角形) (b) Apple(8040 个三角形) (c) HappyBuddha(50000 个三角形)

Figure 20: 静止场景下本文算法与基于 k-DOP 树算法实验结果对比（$k = 24$）

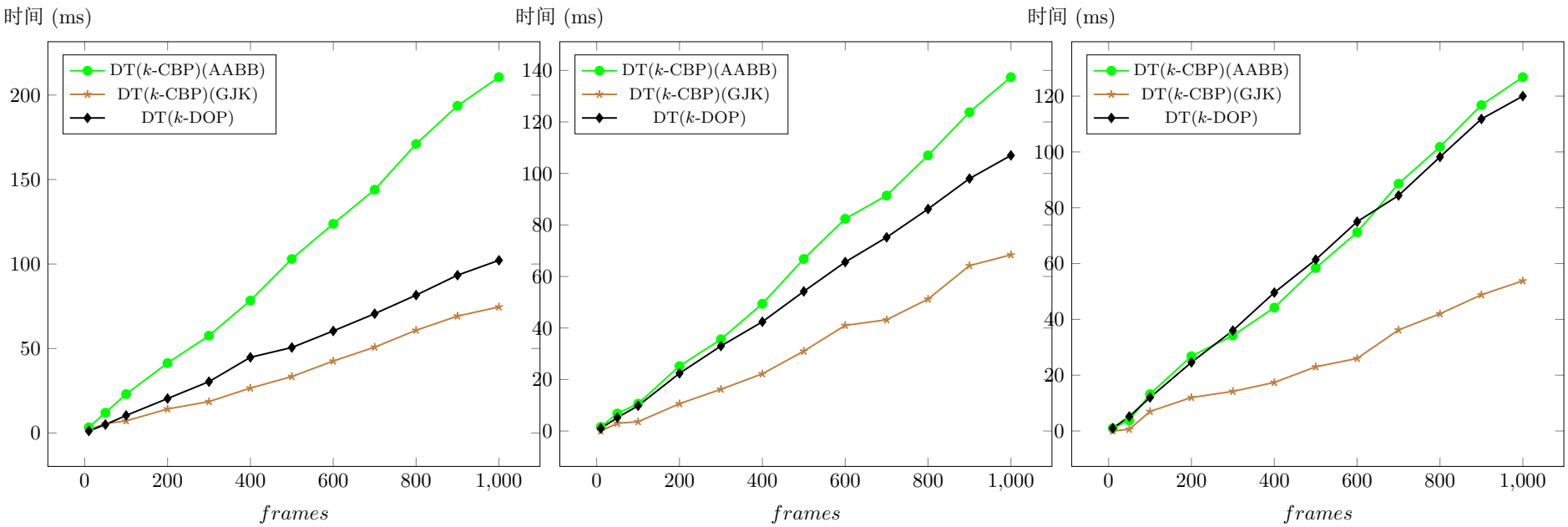
基于 k-DOP 树[8]算法的碰撞检测库 CollDet 是 Gabriel Zachmann 等人实现的。

[8] Sven Trenkel, René Weller, and Gabriel Zachmann. "A benchmarking suite for static collision detection algorithms". In: *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Plzen, Czech Republic: Union Agency, 2007.

这是与基于 k-DOP 树的算法在静止场景下的对比。坐标轴 Y 轴左边标的是初始时间，右边是碰撞检测时间，横坐标为模型数量。红线和蓝线为初始化时间。

从中可以看出，模型较小时，三种算法碰撞检测时间差不多，模型变大，本文基于 GJK 的算法优势就体现出来了。

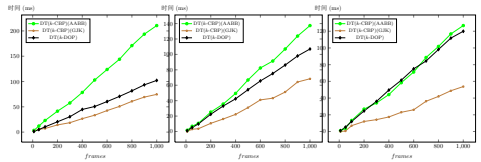
实验结果：运动场景与 k-DOP 树对比



(a) Bunny(4968 个三角形) (b) HappyBuddha(50000 个三角形) (c) Hand(128314 个三角形)

Figure 21: 运动场景下本文算法与基于 k-DOP 树算法实验结果对比（$k = 24, n = 10$）

实验结果：运动场景与 k-DOP 树对比



(a) Bunny(4968 个三角形) (b) HappyBuddha(50000 个三角形) (c) Hand(128314 个三角形)

Figure 21: 运动场景下本文算法与基于 k-DOP 树算法实验结果对比（$k = 24, n = 10$）

这是运动场景下的对比结果，横坐标为运动的次数，纵坐标为运动相应次数碰撞检测所耗费的时间。

可以看出，随着模型变大，基于 AABB 的算法跟 k-DOP 树算法差距越来越小，而基于 GJK 的算法优势明显。

目录

总结与展望

总结

(1) 提出了一种构造紧致凸包围多面体–k-CBP 的算法；

(2) 构造 k-CBP 速度上比现有算法快 3∼8 倍；

(3) 构造的 k-CBP 紧致程度比现有的 k-DOP 紧致 10% ∼ 40%；

(4) 提出了一种基于 k-CBP 的碰撞检测算法，该算法较 k-DOP 树算法初始化时间快 8 倍以上，静止场景快 0.8 ∼ 3.2 倍，运动场景快 0.8 ∼ 5.6 倍。

展望

(1) 碰撞检测算法如何摆脱对 AABB 树的依赖；应用于近似碰撞检测算法；应用于可变形的模型连续碰撞检测，如何快速更新 k-CBP ；

(2) 如何将 k-CBP 应用于如机器人抓取、路径规划等其他应用领域中。

凸包围多面体生成算法及应用
└总结与展望

└总结与展望

总结一下 ...PPT

总结与展望

总结

(1) 提出了一种构造紧致凸包围多面体–k-CBP 的算法；

(2) 构造 k-CBP 速度上比现有算法快 3∼8 倍；

(3) 构造的 k-CBP 紧致程度比现有的 k-DOP 紧致 10% ∼ 40%；

(4) 提出了一种基于 k-CBP 的碰撞检测算法，该算法较 k-DOP 树算法初始化时间快 8 倍以上，静止场景快 0.8 ∼ 3.2 倍，运动场景快 0.8 ∼ 5.6 倍。

展望

(1) 碰撞检测算法如何摆脱对 AABB 树的依赖；应用于近似碰撞检测算法；应用于可变形的模型连续碰撞检测，如何快速更新 k-CBP ；

(2) 如何将 k-CBP 应用于如机器人抓取、路径规划等其他应用领域中。

主要参考文献 I

[1] James T Klosowski et al. "Efficient collision detection using bounding volume hierarchies of k-DOPs". In: *IEEE Transactions on Visualization and Computer Graphics* 4.1 (1998), pp. 21–36.

[2] Christer Ericson. *Real-time collision detection*. San Francisco, CA: Morgan Kaufmann Publishers, 2005.

[3] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. "Approximation algorithms for convex hulls". In: *Communications of the ACM* 25.1 (1982), pp. 64–68.

[4] 邓俊辉. 计算几何-算法与应用. 北京: 清华大学出版社, 2005.

[5] Mattias Karlsson, Olov Winberg, and Thomas Larsson. "Parallel Construction of Bounding Volumes". In: *The Annual Swedish Computer Graphics Association Conference(SIGRAD)*. 2010, pp. 65–69.

主要参考文献 II

[6] Gino van den Bergen. "A fast and robust GJK implementation for collision detection of convex objects". In: *Journal of Graphics Tools* 4.2 (1999), pp. 7–25.

[7] Tomas Moller. "A Fast Triangle-Triangle Intersection Test". In: *Journal of Graphics Tools* 2.2 (1997), pp. 25–30.

[8] Sven Trenkel, René Weller, and Gabriel Zachmann. "A benchmarking suite for static collision detection algorithms". In: *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Plzen, Czech Republic: Union Agency, 2007.

感谢

致谢

(1) 导师雍俊海老师的精心指导；
(2) 施侃乐老师帮助；
(3) 研究所各个项目的历练；
(4) 王斌老师、陈莉老师的评审及意见，答辩委员会老师聆听和指导。

Q & A

Questions?

Thank you!