第14章　碰撞检测

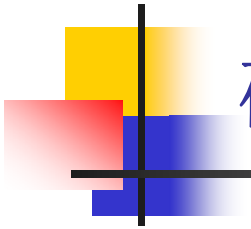
金小刚　**Email: jin@cad.zju.edu.cn**

*浙江大学**CAD&CG**国家重点实验室*

*紫金港校区 信息与控制大楼**506***

碰撞检测(Collision Detection, CD)

- 碰撞检测是计算机图形学和虚拟现实中最基本且非常重要的组成部分。
- 应用广泛，可用于
  - 虚拟制造
  - **CAD/CAM**
  - 计算机动画
  - 物理建模
  - 三维游戏
  - 飞机和汽车驾驶模拟
  - 机器人
  - 路径和运动规划
  - 装配



碰撞处理

- 碰撞检测 (Collision Detection)
  - 返回两个或多个物体是否发生碰撞的布尔判断
- 碰撞确定 (Collision Determination)
  - 找到物体之间的实际相交位置
- 碰撞响应 (Collision Response)
  - 针对两个物体之间的碰撞决定采取何种操作

本章内容

- 使用射线进行碰撞检测
  - 用线条来表示复杂物体，然后对线条和环境中的图元进行相交测试。主要用于游戏。
- 使用**BSP**树进行动态碰撞检测
- 一般层次碰撞检测
- **OBBTree** （层次碰撞检测的具体实现之一）
- ***k*-DOPTree** （层次碰撞检测的具体实现之二）
  - 无论模型的大小、距离，均能取得实时交互速度
  - 可以处理多边形黏体**(Polygon Soup),** 如凹体、非流形
  - 模型可以进行刚体运动（旋转、平移）
  - 可以提供高效的紧凑包围体

本章内容

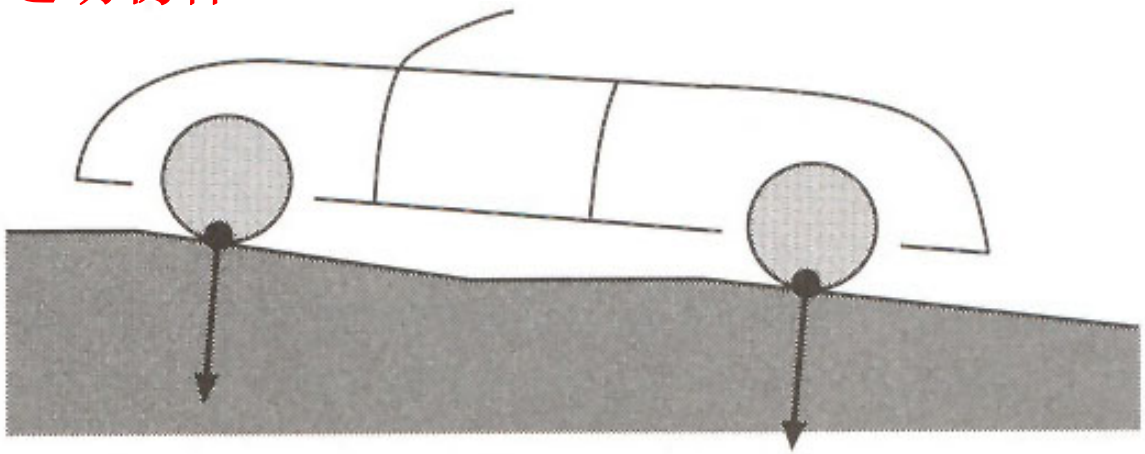
- 多物体碰撞检测系统
  - 一个好的碰撞检测系统应能处理包含成百上千的运动物体的场景。假设场景中有**n**个运动物体和**m**个静止物体，则需要测试的次数为：

$$nm + \binom{n}{2}$$

    其中第一项为运动物体和静止物体之间的测试次数，第二项为运动物体自身之间的测试次数。
  - 随着**n**、**m**的增大，计算开销越来越大，因此需要一种巧妙的方法来处理这些情形！（这正是本节需解决的问题）
- 其它主题
  - **Front tracking, Time-critical collision detection,** 最短距离计算

使用射线进行碰撞检测

- 是一种特定环境下的快速碰撞检测技术，实际应用效果非常好。
- 例子：一辆小汽车沿着斜坡向上行驶，可利用路上的信息（路面形成的图元）来引导汽车向上行驶。但对于游戏等应用，并不需要这种复杂的碰撞检测。可使用一组射线来近似运动物体。



不是计算整辆测车子与环境（路面）之间的碰撞，而是在每个轮子上引出一条射线，然后对这些射线与环境进行相交测试。

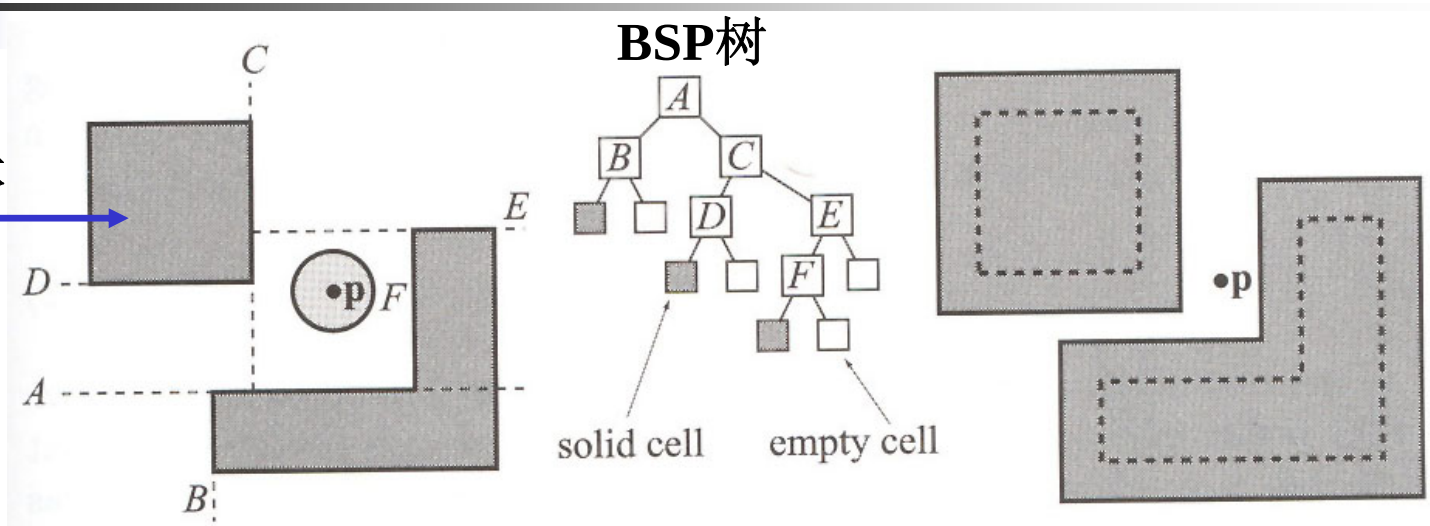
使用射线进行碰撞检测

- 假设小汽车一开始在一个平面上，4个轮子是小汽车唯一与环境接触的地方。可在4个轮子上各生成一条射线，射线的起始点位于轮子和环境之间的接触点上，然后对轮子上的射线与环境进行相交测试。
- 如果从射线原点到环境的距离为0，则轮子在地面上；如果这个距离大于0，则轮子和环境没有接触；如果距离为负，则轮子陷在环境中。
- 可用这些值来计算碰撞响应：负距离可使小汽车（所在轮子处）向上运动，正距离可使小汽车向下运动。
- 相交测试加速技术：层次表示、BSP树等。
- 为了避免射线在两个方向进行搜索，可将测试射线的原点向后移动直到它在环境包围盒的外面，然后再对这条射线和环境进行相交测试。

使用BSP树进行动态碰撞检测

- **Melax的碰撞检测算法：**
  - 对用**BSP**树描述的几何体进行碰撞检测，其中碰撞体可以是
    - 球体
    - 圆柱体（可以用来近似人物几何体）
    - 物体凸包
  - 算法可以进行动态碰撞检测
  - 已经用于商业游戏

球体碰撞体

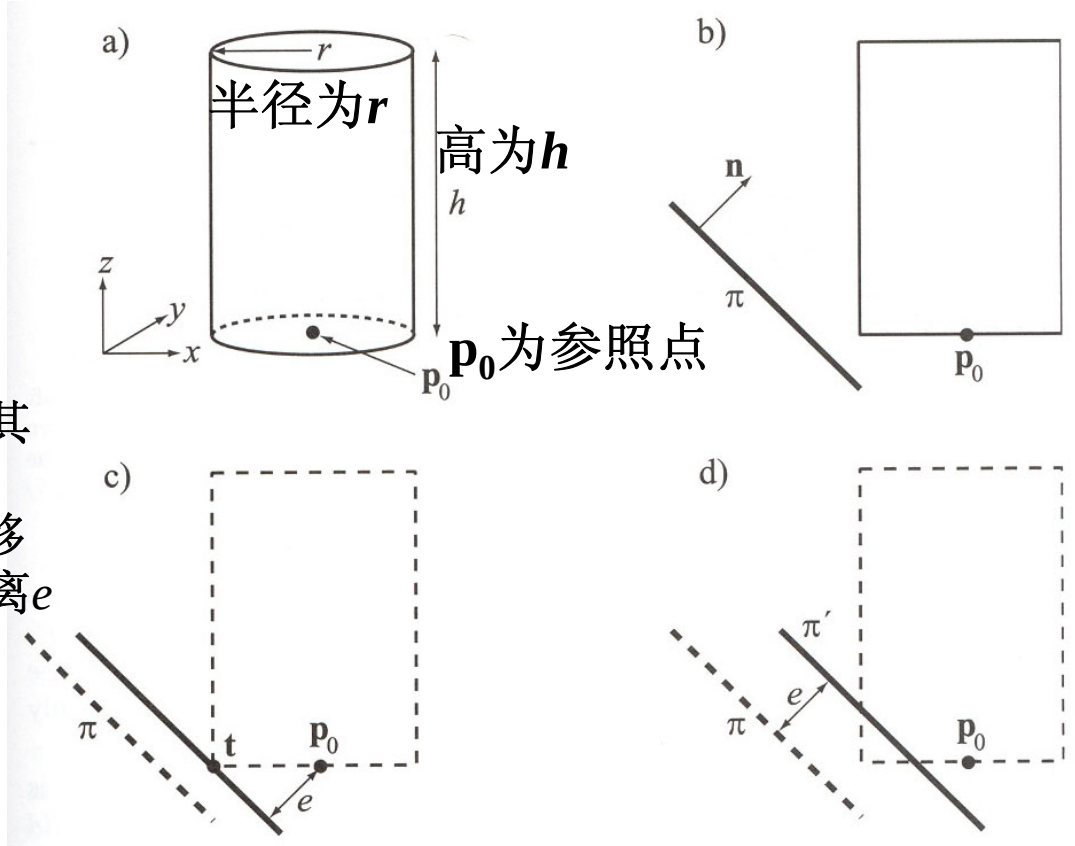


- 一个点从p_0移动到p_1，可以用线段p_0p_1表示。线段和标准的**BSP**树之间的相交测试可以高效地进行。这里可能会有多个交点，但只有第一个相交点表示这个点与几何体之间的碰撞。
- 扩展到球体：将**BSP**树向外扩展园的半径长度
- 对**p**和扩展后的**BSP**树进行相交测试
- 拐角应该是圆形，因此是这种算法引入的一种近似形式
- 测试平面从 π：**n.x**+d≥**0**调整为 π’：**n.x**+$d\pm r$≥**0**（±符号取决于碰撞搜索在平面的哪一侧）

平面/圆柱体的相交测试

- 在游戏中，球体往往不能很好地对人物进行近似
- 人物的凸包或包围人物的圆柱体相对来说更好
- 目标：对**BSP**树和由一组顶点***S***形成的运动凸包进行相交测试
- 圆柱体在游戏中更接近人体，对圆柱体进行的相交测试也较快

平面/圆柱体的相交测试



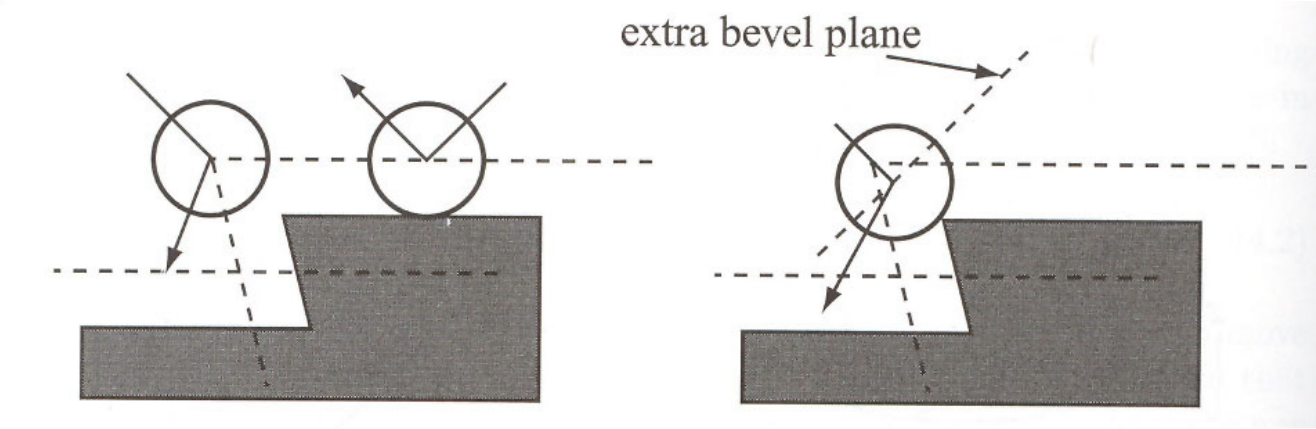
希望解决的问题：圆柱体和平面π进行相交测试

对平面π移动使其刚好接触到圆柱体，计算从$\mathbf{p}_0$到移动之后平面的距离e

用距离e将平面π移动到新位置π'

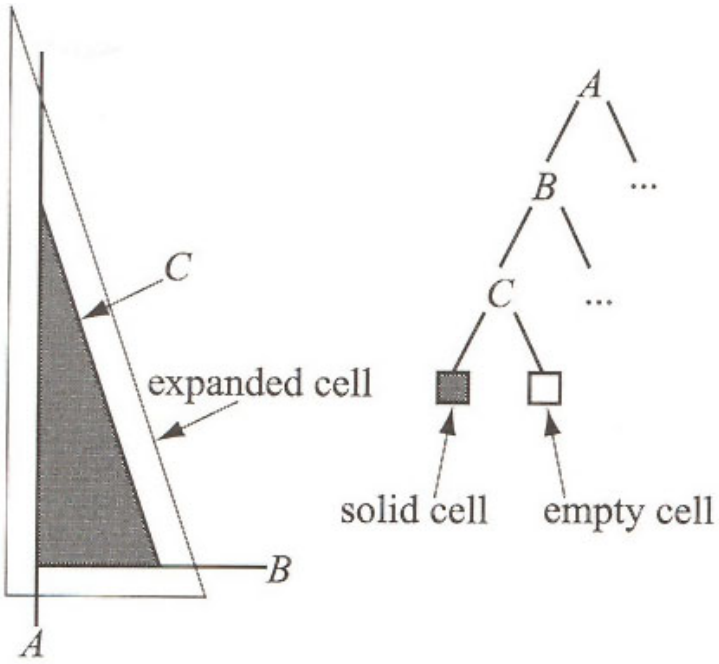
最终结果：将平面π与圆柱体之间的相交测试转换为点$\mathbf{p}_0$与新平面π'之间的测试

例外情形

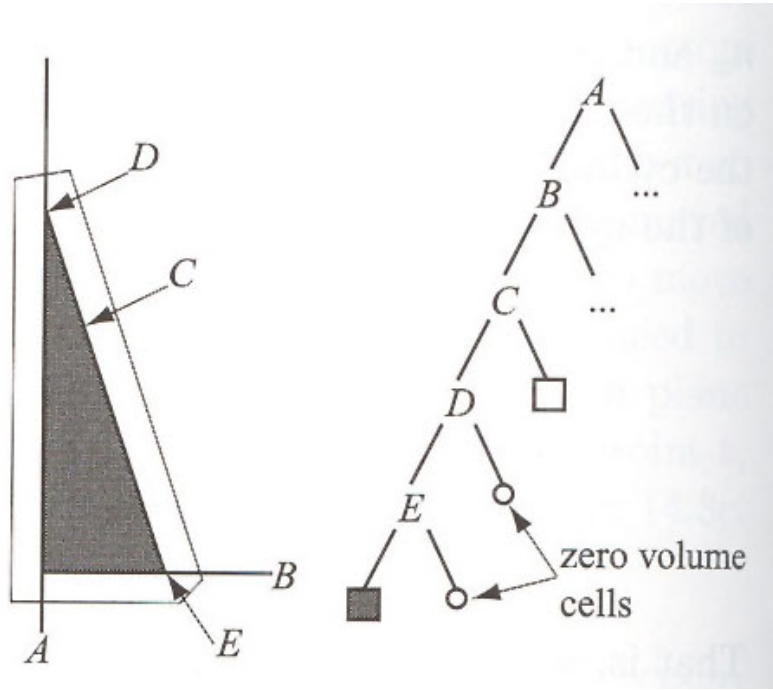


- 方法会产生不精确性
- 引入额外的倒角面来解决
- 如果相邻平面间的外角大于**90**度，就插入一个额外平面
- 不能去掉所有的错误
- 缺点：所花费的代价要比不使用调整平面高出**2.5**至**3.5**倍。但由于碰撞检测所耗费的时间**(66us)**相对于绘制画面的时间**(33000us)**并不多，因此由此引入的代价可以忽略。

额外的倒角面引起的BSP树变化



一个正常的单元和对应的**BSP**树



在单元中增加了倒角面，**BSP**树相应发生了改变

一般层次碰撞检测

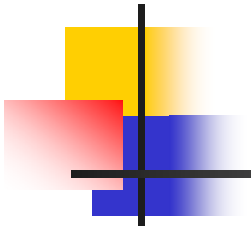
- 一般层次碰撞检测算法的三个特点
  - 使用包围体为每个模型创建一个层次表示形式
  - 不论采用何种包围物体，用于碰撞检测的高层代码总是类似的
  - 可用一个简单的代价函数(**Cost Function**)对性能进行修剪、计算、比较

层次创建

- 模型是由很多图元表示的。超过3个顶点的多边形可以转化为三角形。
- 在碰撞检测算法中，模型常用的层次结构为***k***叉树数据结构，每个节点最多有***k***个子节点。
- 很多算法采用最简单的二叉树数据结构。每个内部子节点有一个包围体，它包含所有的子节点；在每个叶节点，存在一个或多个图元（三角形）。
- 将任意节点的包围体A（内部节点或叶节点）表示为$\mathbf{A}_{bv}$，将属于A的子节点表示为A_c。

层次创建的三种形式

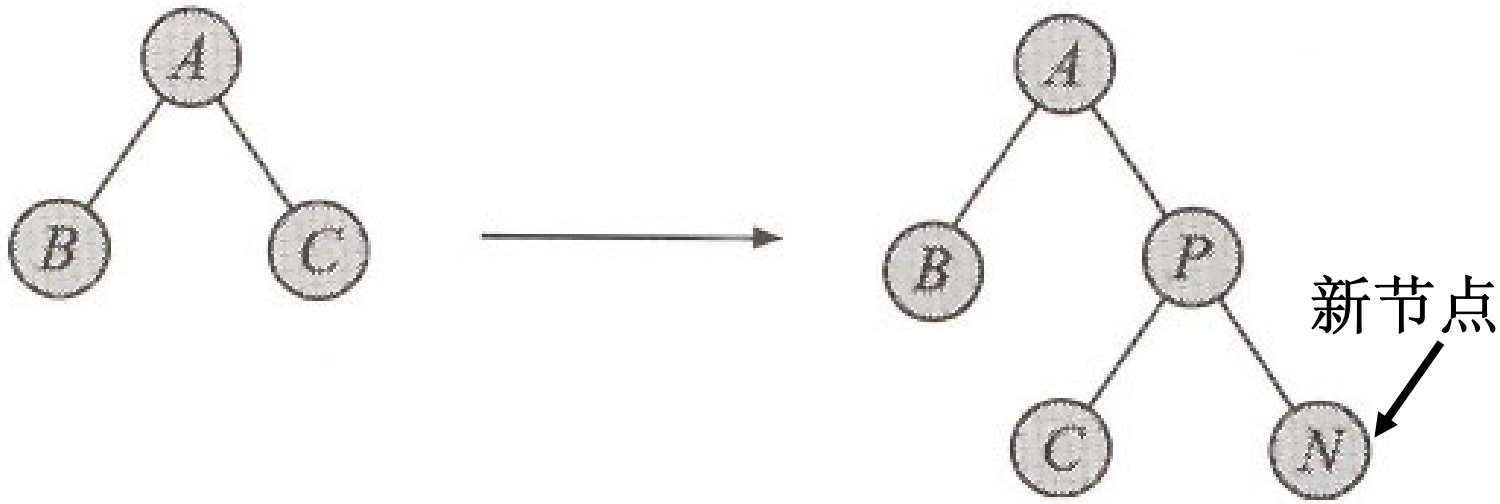
- 为了创建一个高效紧凑的结构，需尽可能将包围体的体积最小化。层次创建有的三种形式:
  - 自下而上
  - 渐进树插(Incremental Tree-insertion)
  - 自上而下



自下而上方式

- 首先将大量图元组合起来并找到它们的包围体，图元应紧密地靠在一起，可利用图元之间的距离来确定。
- 然后，用同样的方法创建一个新的包围体，或者组合现有的包围体。
- 直到只剩下唯一的包围盒，把它作为根节点。

*Incremental Tree-insertion*方式



- 从一棵空树开始，然后将所有图元和相应的包围体逐个加入到这棵树上。
- 为了使得树的效率比较高，必须在树中找到使得整棵树体积增加量最小的插入点。

自上而下方式

- 这是大多数层次构造算法所采用的方法。主要步骤

1. 首先找到包含模型所有图元的包围体，并把其作为根节点。
2. 然后，采用分而治之的策略，将**BV**分割成**k**个部分。
3. 采用递归方式，对于每部分进行类似的分割

潜在优点：可以根据需要创建层次。

层次结构的好坏衡量准则

- 对于平衡树，遍历每个叶子节点所耗费的时间几乎相等，所以平衡树的性能最好！

- 但并非对于所有输入，平衡树都是最好的。例如，如果模型的某个部分很少或从来不用于碰撞查询，则这部分可以位于一棵不平衡树的深层部分，而保证经常查询的部分尽可能靠近根节点。

层次间的碰撞检测

- 用户通常需要的碰撞检测情形
  1. 用户只对两个模型是否碰撞感兴趣，只要确定两个三角形发生重叠，测试过程即可终止。
     ——————————*碰撞检测(**collision detection**)*
  2. 用户希望得到所有的重叠三角形。
     ——————————*碰撞确定(**collision determination**)*

- 下面给出第一种情形的伪代码。稍作修改便可处理第二种情形

碰撞检测伪代码

FindFirstHitCD(A, B)
Return({TRUE, FALSE})
1: if(isLeaf(A) and isLeaf(B)) // A 、B都是叶节点
2: for each triangle pair $T_A \in A_C$ and $T_B \in B_C$
3: if(overlap(T_A, T_B)) TRUE;
4: else if(isNotLeaf(A) and isNotLeaf(B)) // A 、B都不是叶节点
5: if(Volume(A) > Volume(B)) // 为了获得更好的性能
6: for each child $C_A \in A_C$
7: **FindFirstHitCD**(C_A, B))
8: else
9: for each child $C_B \in B_C$
10: **FindFirstHitCD**(A, C_B))
11: else if(isLeaf(A) and isNotLeaf(B)) // A是叶节点， B不是
12: for each child $C_B \in B_C$
13: **FindFirstHitCD**(C_B, A))
14:else
15: for each child $C_B \in A_C$
16: **FindFirstHitCD**(C_A, B)) // B是叶节点，A不是
17: return FALSE;

代价函数

- 代价函数：$t = n_v c_v + n_p c_p + n_u c_u$
  - n_v: BV/BV重叠测试的数目
  - c_v: BV/BV重叠测试的开销
  - n_p: 有重叠的图元对数目
  - c_p: 测试两个图元是否有重叠的开销
  - n_u: 由于模型运动的原因需要更新的BV数目
  - c_u: 更新一个BV的开销
- 比较好的模型层次分解会产生较低的n_v, n_p, n_u。
- 比较好的重叠测试方法会降低c_v和c_p的值。但这又会与上面的目标冲突，因为快的重叠测试意味着不是很紧凑的包围体。

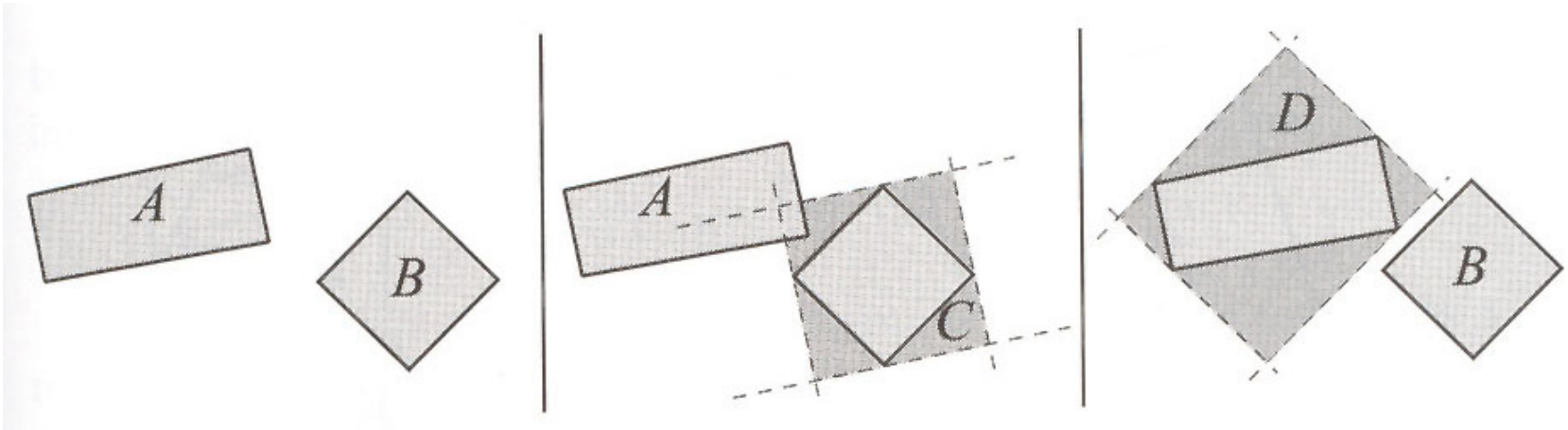
OBBTree

- 在**Siggraph 1996**年会上，**Gottschalk**等人的论文**“OBBTree: A Hierarchical Structure for Rapid Interference Detection”**对碰撞检测算法的深入研究具有很大的影响。
- 设计**OBBTree**的目的：针对碰撞检测中两个表面非常靠近而且几乎平行时的情形。
- 针对这这种情形， **OBBTree**要比***k*-DOPTree**要好
- 可用于公差分析和虚拟原型中

包围体的选择

- 选用**OBB**的原因
  - **AABB**和球体包围盒的逼近效果较差，**OBB**的紧贴程度更好
  - 作者提出了一种速度更快的**OBB**重叠测试方法，速度要比以前的方法快一个数量级。原因：对**OBB**进行变换，使得其中一个成为中心在原点的**AABB**。实际变换需**63**次操作。在**15**个轴测试中，只要在一个轴分离，测试即可结束。在变换之后，第一个轴测试后终止需**17**次操作，最后一个轴测试后终止需**180**次操作。

- **Van den Bergen提出的加速方法**



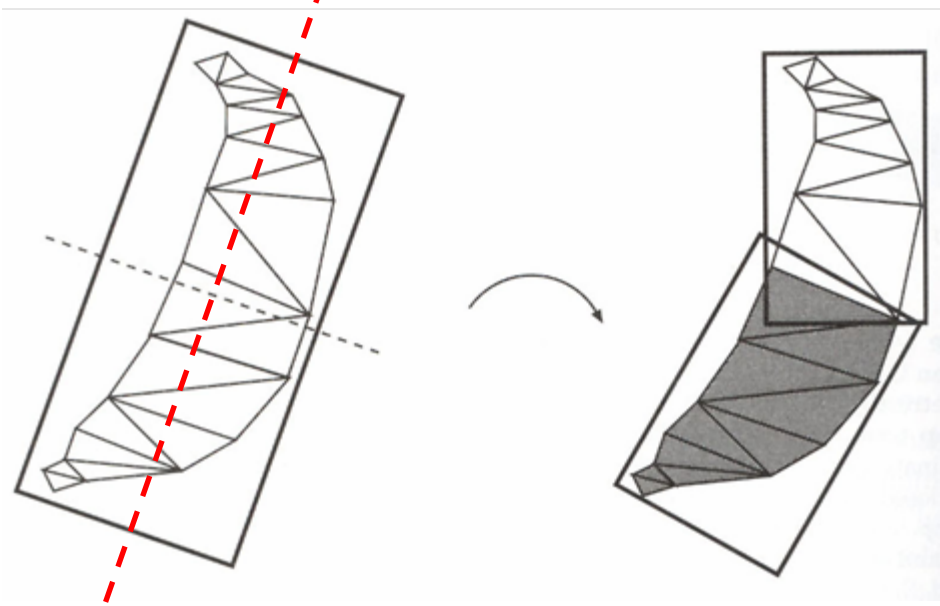
OBB/OBB重叠测试，省略9个轴

把B进行变换，进行AABB/AABB测试

把A进行变换，进行AABB/AABB测试

层次创建

- 基本的数据结构为一棵二叉树，每个内部节点为一个**OBB**，每个叶节点为一个三角形。
- 自上而下：为多边形**Soup**建立一个紧密贴合的**OBB**，然后沿着**OBB**的一个轴对这个**OBB**进行分割，把三角形也相应分成两组；对于每一组，计算一个新的**OBB**
- **Gottschalk**的分割策略：将包围盒的最长轴分割为长度相同的两部分。



处理刚体运动

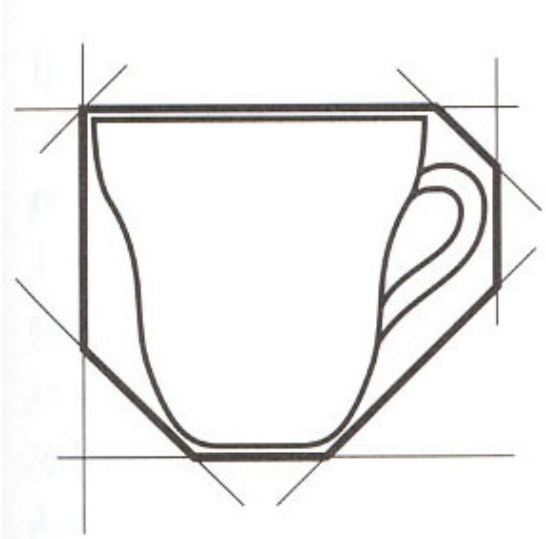
- 在**OBBTree**层次中，每个**OBB**可以和刚体变换矩阵$\mathbf{M}_A$存贮在一起，这个矩阵含有**OBB**对其父节点的相对位置和方向。

- 现在要对两个**OBB**进行测试（分别为*A*和*B*），则应在一个**OBB**所在的坐标系中进行*A*和*B*之间的重叠测试，不仿在*A*所在的坐标系中进行测试。这样，*A*就是中心位于原点的**AABB**，然后将*B*变换到*A*所在的坐标系中。变换矩阵为：$\mathbf{T}_{AB}=\mathbf{M}_A^{-1}\mathbf{M}_B$

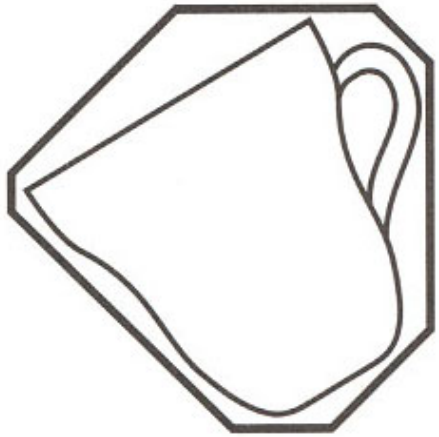
k-DOPTree

- 之所以选择*k*-DOP为碰撞检测层次中的包围体：
  - 重叠相交测试速度快（因为*k*-DOP的平面方向是固定的）
  - 包围盒逼近效果比较好
- 随着*k*的增大， *k*-DOP与物体的凸包越来越相似
- 测试表明，选$k=18$时具有最好的效率
- AABB的法向可以给出6个平面，两个法向之和给出了另外12个平面，这样就生成了一个18-DOP。
- 好的法向选择可使顶点在法向上投影的计算开销非常小，只需要进行加法和减法运算。（原因：非归一化法向的分量要么是0，要么是1）

物体旋转后k-DOP的近似计算

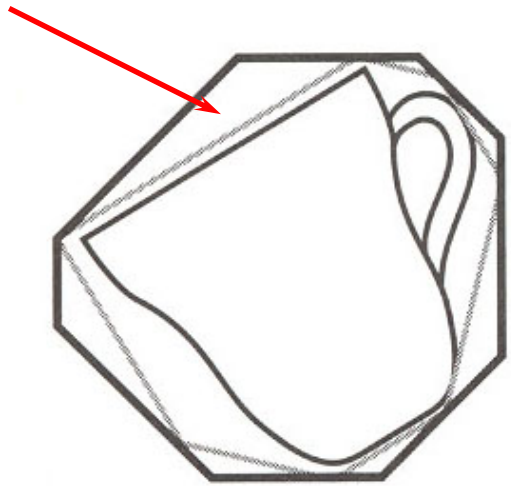


一个茶杯的8-DOP



茶杯旋转后，一个好的8-DOP拟合，但计算量较大

前一个8-DOP



近似方法：对前一个8-DOP进行变换

层次创建

- **k-DOPTree采用二叉树结构**
- **k-DOPTree与OBBTree的主要区别： k-DOPTree的叶子节点可以包含任意多的三角形。对于静态模型，每个叶子节点所允许的最大三角形数目为1；对于动态模型，每个叶子节点所允许的最大三角形数目为40。*原因*：对k-DOPTree进行变换的计算开销比较大。**
- 采用自上而下的方法进行层次创建。
- 当需要将**k-DOP**分割为两个子体时，对于如何在**x**、**y**、**z**中进行选择，主要采取四种方法：
  - 最小和方法：使子体之和最小
  - 最小最大法：使两个子体中较大一个最小化
  - 泼溅方法：计算三角形质心在每个轴上的投影变化，选择其中一个差异最大的轴。（最好的方法）
  - 最长边方法：在所选轴的方向上，包围体的长度最长

通过刚体运动对*k*-DOPTree进行更新

- 由于**k-DOP**的平面方向是固定的，因此物体的刚体运动会使得层次结构变得无效(可以处理平移，但不能处理旋转)，必须使用某种方法对层次结构进行更新。

- **Klosowski**提出的两种高效更新方法(可保持层次结构不变)：
  - 近似方法：避免为变换之后的几何体重新计算一个新的**k-DOP**，而是通过变换和使用原始的**k-DOP**拐角处的顶点位置。（前提： **k-DOP** 的顶点比几何体的顶点少。缺点：更新后的**k-DOP** 逼近几何体的效果不好）
  - 上山方法：利用当前节点包含的几何体凸包。测试沿**k-DOP**法线方向的最远顶点是否依然保持最远，如果不是，则用其邻近的最远点更新。（优点：逼近效果好一些。缺点：计算开销较大。）

多物体碰撞检测系统

- 前面介绍的方法主要用于测试两个实体之间的碰撞检测。如果要在不同的物体之间进行成百上千的碰撞检测，若采用逐个测试的策略，则执行效率上非常低，在实际应用中不可行。

- 将介绍两级碰撞检测系统：主要针对大规模场景中多个物体的运动。
  - 第一级主要找到环境中所有物体之间的可能碰撞，并将结果送入第二级。
  - 第二级对两个物体之间进行精确的碰撞检测。

第一级碰撞检测

- 为了尽量减少可能发生碰撞的物体数量，从而尽量减少调用第二级碰撞检测的次数，可以将每个物体封装在一个包围体中，然后找到所有重叠的包围体对。

- 简单处理方法：对每个物体使用**AABB**。为了避免对当前刚体运动（旋转会引起**AABB**的变化）的物体重新计算**AABB**，可以对**AABB**进行调整，使其成为一个固定的立方体，无论物体处于何种方向都可以包围该物体。使用固定立方体可以快速判断在包围体层次上彼此分离的物体对。

- 固定立方体以外的其它方法：球体、凸多面体

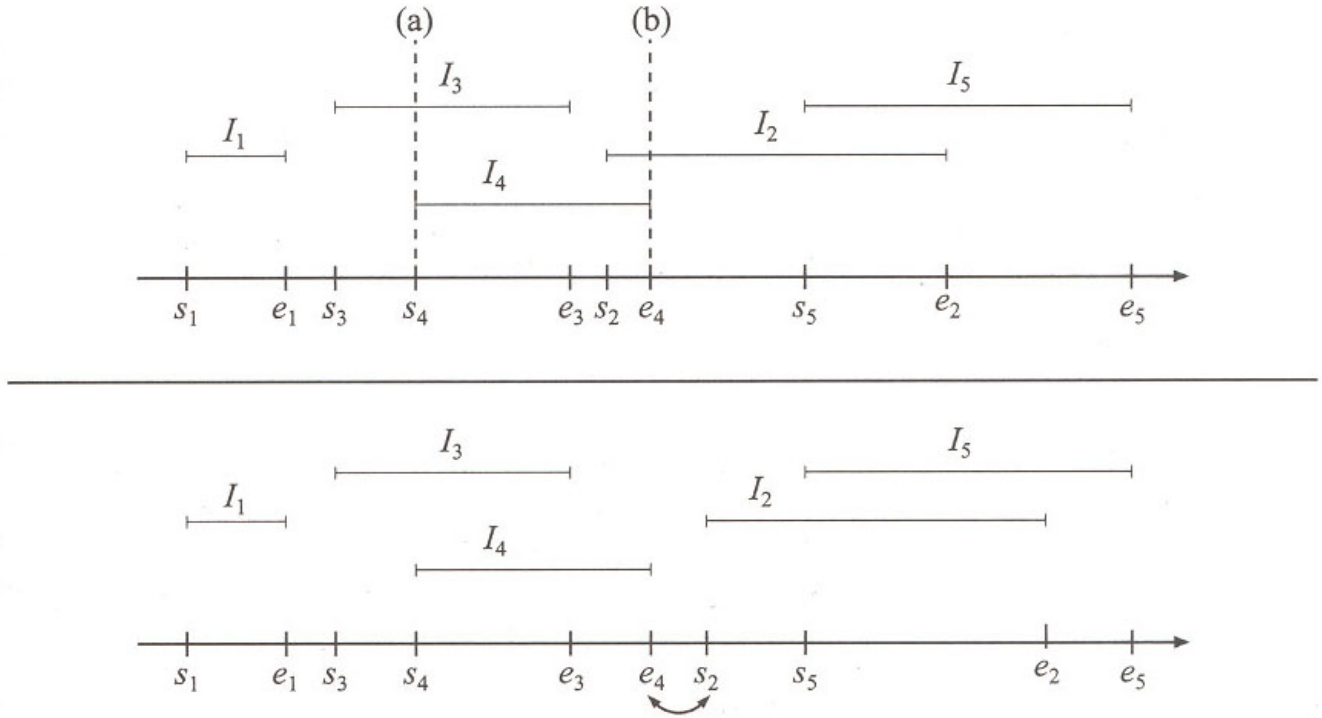
扫描-修剪方法

- 使用固定立方体。
- 利用虚拟环境中的时间相关性**(Frame-to-Frame Coherence)**。即两幅连续画面之间物体的位置和方向变化比较小。
- 三维包围盒重叠问题的时间复杂度为$O(n\log^2 n+k)$ (其中k为重叠对的数量)；若利用相关性，则时间复杂度降为$O(n+k)$ 。
- 如果两个**AABB**重叠，则在**3**个主轴方向上的所有一维时间区间（由**AABB**的起始点和终止点形成）也一定相互重叠。
- 当帧间相关性比较高时，可以对所有的一维区间的重叠情况进行高效检测。
- 用每个主轴的一维算法来解决**AABB**的三维问题！

扫描-修剪方法

- 假设用s_i和e_i表示某个特定轴上的n个区间，其中$s_i<e_i$且$0 \leq i < n$，这些值存贮在一个升序列表中。
- 从头到尾对这个列表进行扫描。
- 当遇到起始点s_i的时候，就将相应的区间放到一个活动表中；当遇到一个结束点时，就将相应的区间从活动表中清除。
- 当在活动列表中存在区间，而又遇到一个新区间的起始点时，则遇到的区间与列表中的所有区间重叠。

扫描-修剪方法



在上图，当活动列表中只存在一个区间I_3时，如果遇到I_4(在标记点(a)处)，则可以断定I_3和I_4重叠。如果遇到I_2，但因I_4仍然在活动列表中(还没遇到e_4),则I_2和I_4重叠。如果遇到e_4，则将I_4从活动列表中清除(在标记点(b)处) 。

在下图，将I_2移到右边，如果插入分类方法发现s_2和e_4需要改变位置，则可以肯定I_2和I_4不再彼此重叠。

扫描-修剪方法的算法复杂度分析

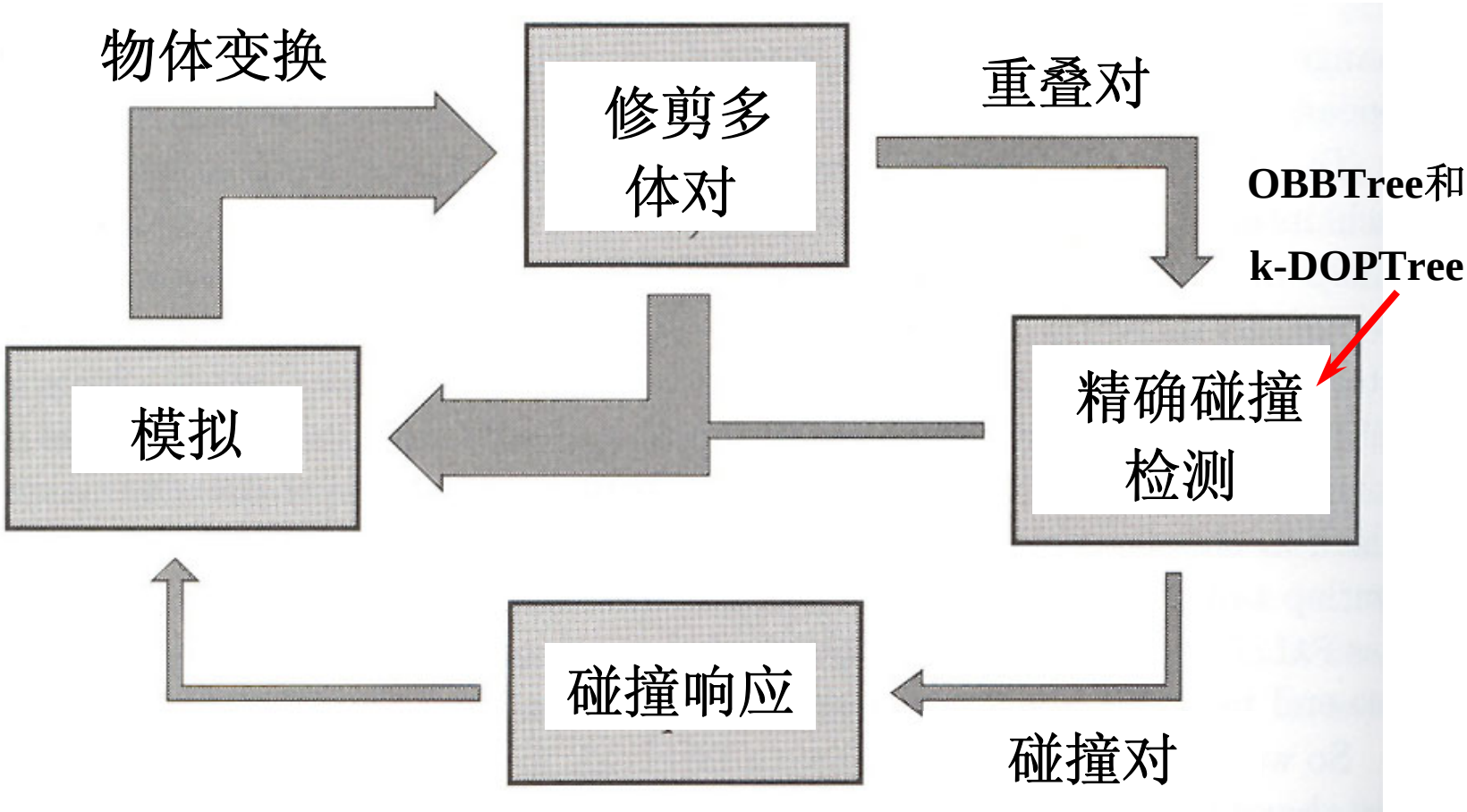
- 对所有区间进行排序的时间复杂度为$O(n\log n)$, 对列表进行扫描的时间复杂度为$O(n)$, 对k个重叠区间进行报告的时间复杂度为$O(k)$, 所以算法的时间复杂度为$O(n\log n+k)$

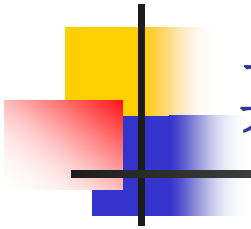
- 但由于时间相关性，画面之间的列表变化不是很大。在第一遍处理结束后，可以采用冒泡（Bubble）算法或插入分类算法，来提高排序的效率。对于几乎分类好的列表来说，算法的复杂度为$O(n)$。

插入分类算法

- 是一种逐步建立分类序列的方法。
- 首先从列表中的第一项开始，如果只考虑这个项，则表已经分好。
- 接下来，处理第二项，如果第二项比第一项小，则改变第一项和第二个项的位置；否则不改变。
- 继续增加新的项，直到处理完整个表。则结果表已经排好序。（算法的复杂度为$O(n)$）

多物体碰撞检测系统基本框架总结





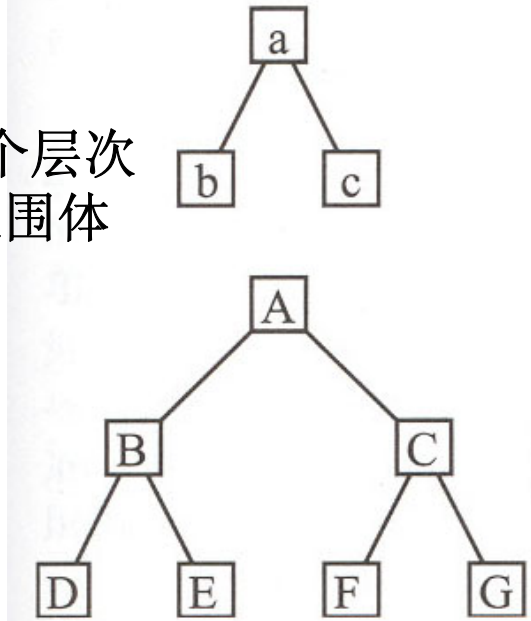
其它主题

- 向前跟踪(Front Tracking)的加速技术
- 限时碰撞检测(Time-critical collision detection)
- 距离查询(Distance queries）
- 任意变形物体的的碰撞检测
- 碰撞响应

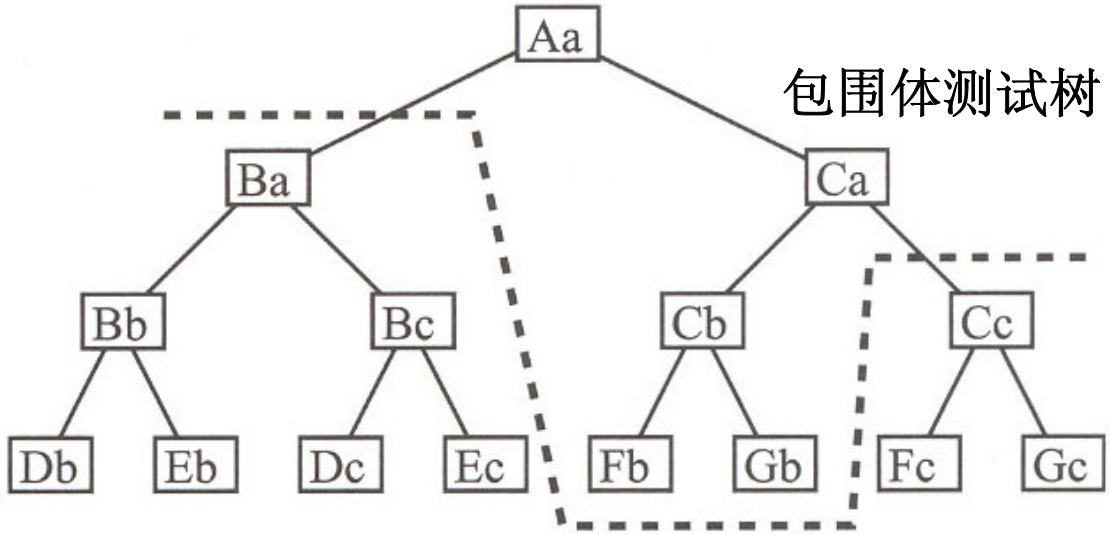
向前跟踪(Front Tracking)

- 利用时间相关性来加速碰撞查询，可以用于任何使用层次包围体的碰撞检测系统。

两个层次包围体



包围体测试树



节点表示左图中两个节点之间的测试。例如，最上面的节点**Aa**表示相互测试的两个根节点。

注：这里所示为整棵树，虽然测试过程在到达叶节点之前会终止。

向前跟踪(续)

- 当对包围体进行测试时，并不是所有节点都会重叠。
- 虚线表示重叠和非重叠之间的交界。在这条线之上的每对节点重叠，在这条线之下的每对节点不相交。
- 算法关键：对前端的节点进行跟踪。通过为下一帧存贮“前端节点”，则测试可以从前端节点之上开始，从而节省包围体测试树上半部分的遍历时间。

限时碰撞检测(Time-critical collision detection)

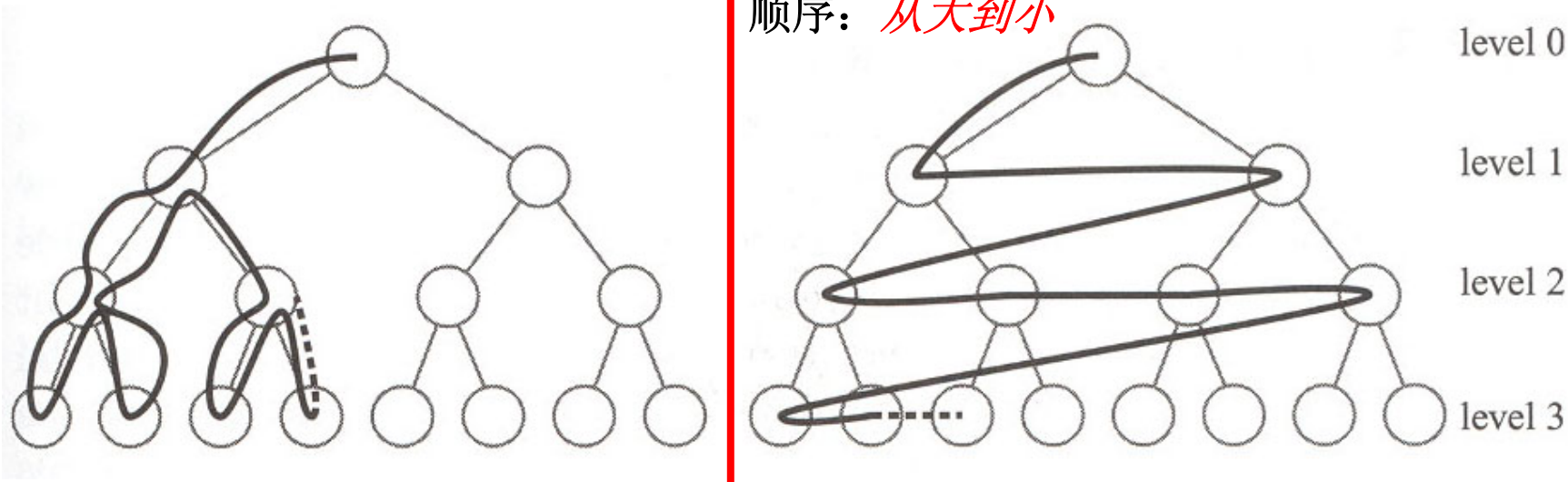
- 在指定的时间内完成碰撞检测。

- 如果绘制一幅画面所需的绘制时间最多为**20ms**，则碰撞检测部分可以用的最多时间为**20ms**。例如，如果一幅画面需要**15ms**进行绘制，则碰撞检测部分只能使用**5ms**的时间。

- 基本思想：以宽度优先顺序来遍历层次包围盒。也就是说，在进入树的下一级时必须将当前层次的所有节点访问一遍（见下页图）。

限时碰撞检测(Time-critical collision detection)

包围盒之间的测试
顺序：*从大到小*



深度优先遍历
由于算法可能超时，只能访问左子树

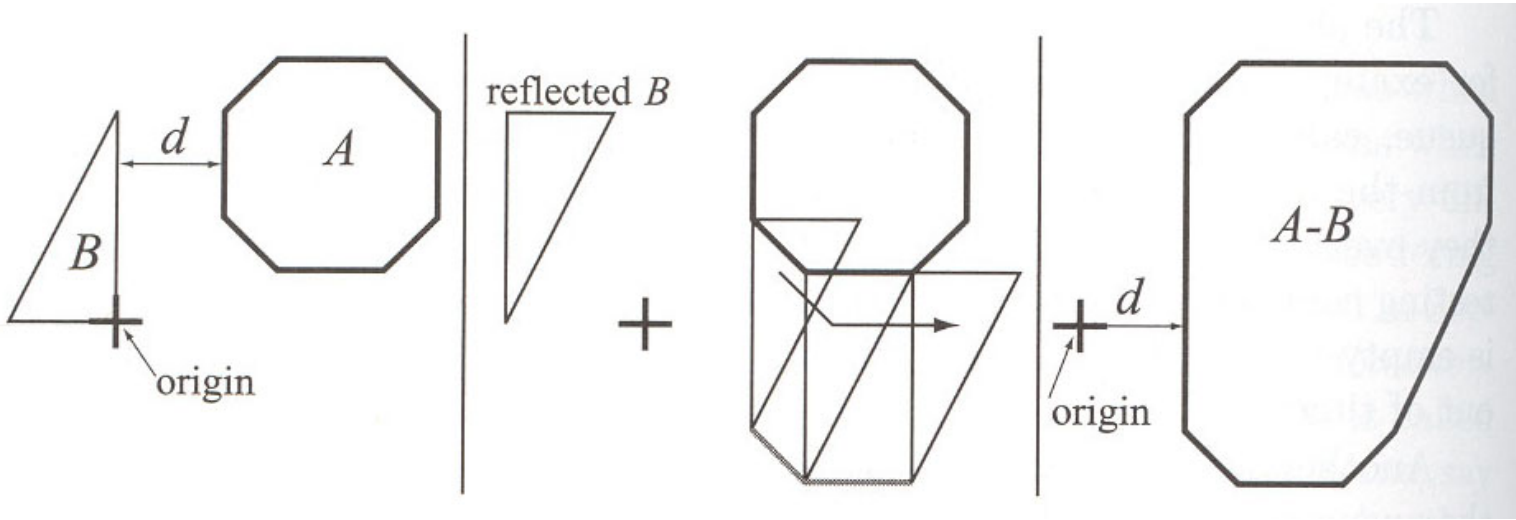
宽度优先遍历
对左右节点都进行访问

距离查询

- 在某些应用中，可能希望测试物体与环境之间的距离。
- 计算碰撞相应需要知道贯穿深度，从而可以用这个距离将物体回移，并计算相应的物理量。
- 给定一个物体的速度和加速度，可以用最小距离来估算碰撞时间的下限，从而在这个时间之前避免进行碰撞检测。
- 汽车设计中，不同体形乘客舒适感的评估等

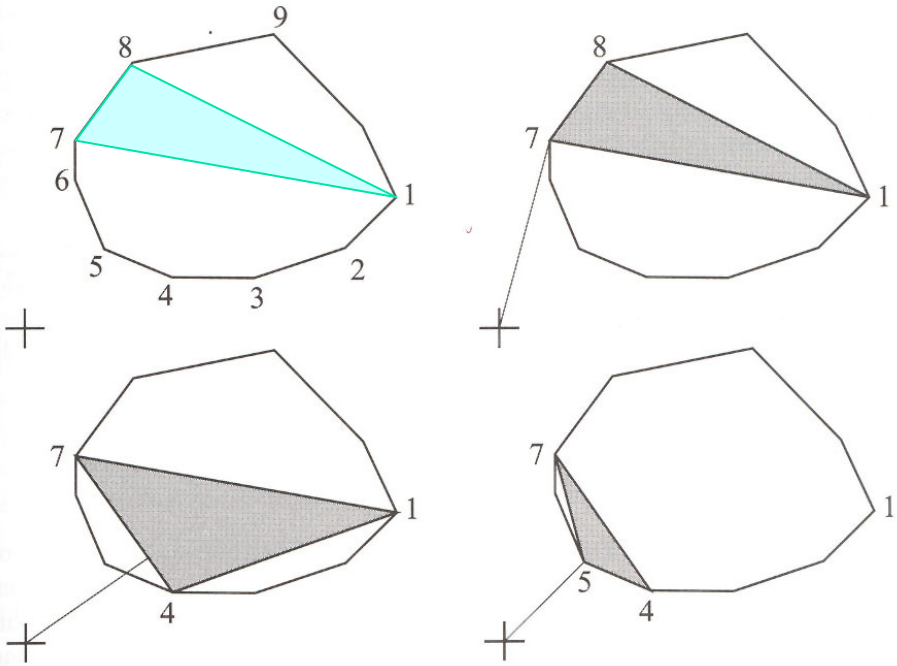
两个凸多面体之间的最小距离

- 基本思想：把计算A和B之间的最小距离问题转化为计算原点与(A-B)(Minowksi和)之间的最小距离问题。



左图为两个凸物体A和B。为了构造A-B，首先移动A和B使得参考点位于原点（左图所示已经完成这一点），然后对B进行反射，如中图所示，将B上的参考点放置在A的表面上，随后围绕A对B进行扫描，生成A-B。

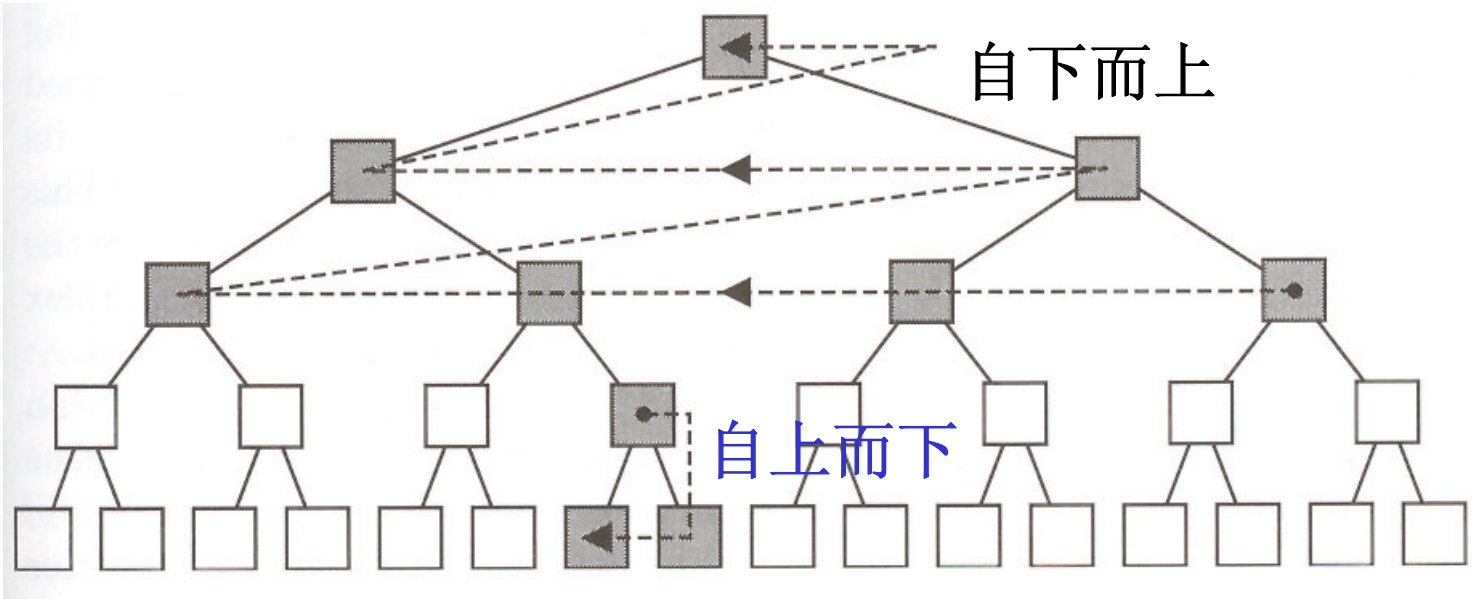
计算点与多面体之间的最小距离（二维）



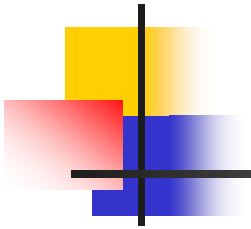
左上：计算原点与多边形之间的最小距离。右上：选择任意一个三角形作为算法的起始点，并计算出原点与三角形的最小距离，顶点7最近。左下：将所有顶点投影到从原点到顶点7之间的直线上，用最近的顶点替换在三角形中投影最远的点，这样顶点4就替换了8。然后，找到三角形上最近的点，这个点位于从顶点4到顶点7所在的边上。右下：将顶点投影到从原点到上一步中最近点之间的连线上，用顶点5取代顶点1（顶点1是投影上最远的点，顶点5是三角形上最近的点）。当把顶点投影在原点与顶点5之间的连线上时，会发现顶点5是这个三角形上最近的点。因此，最近的点是5。

可变形模型

- 前面讨论的物体都是刚体，但自然界中还存在布之类的可变形物体。
- 假设物体在变形过程中，网格的连接关系保持不变。想办法利用这种连接关系。
- 因为大多数包围体在碰撞查询中一般不会用到，因此树中需要更新的包围体非常少。
- 采用“自上而下/自下而上”的混合更新方法。基本思想：对高层节点（包括根节点）使用一种自下而上的更新思想（理由：较高层次的包围体通常会裁剪掉大部分物体）



自下而上/自上而下的混合更新方法。使用自下而上的策略对上层的节点进行更新；仅仅对在遍历过程中所访问树中比较深的下部节点采用自上而下的方法进行更新。



- 对于更新后的层次和其它树进行重叠测试。对于不重叠的节点来说，可以跳过对其子树的更新，这样可以节省很多计算量。

- 对于发生重叠的节点，使用一种自上而下的策略来对子树中的节点进行更新。

- 理由：*重叠测试速度比包围体更新速度快！*

对于可变形物体更一般的方法

- 计算出进行碰撞测试的两个物体的最小**AABB**
- 如果彼此重叠，则计算出重叠的**AABB**区域，即**AABB**的相交体积。
- 创建该区域内的所有三角形列表，建立包含这些场景的八叉树。
- 在八叉树节点中插入三角形，如果两个物体的三角形在同一个叶节点中，则对这些三角形进行相交测试。

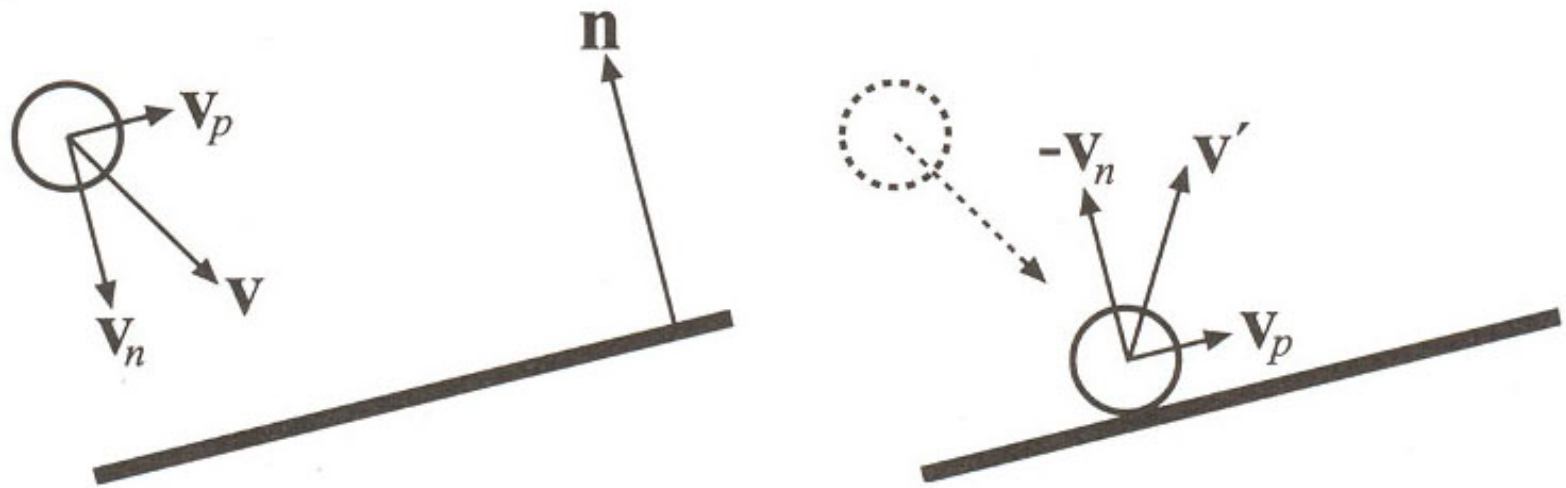
碰撞响应

- 碰撞响应作为一种动作，可以避免物体之间的相互贯穿。

- 球体和平面之间相互碰撞后的响应。
- 假设一个球体向一个平面运动，速度矢量为**v**，平面为Π: **n.x+*d*=0**，其中**n**为单位化向量。
- 为了计算碰撞响应，可将速度向量表示为：

  $\mathbf{v}=\mathbf{v}_n+\mathbf{v}_p$，其中$\mathbf{v}_n=(\mathbf{v}.\mathbf{n})\mathbf{n}$, $\mathbf{v}_p=\mathbf{v}-\mathbf{v}_n$
- 得到碰撞之后的速度向量**v‘**为

$$\mathbf{v}'=\mathbf{v}_p-\mathbf{v}_n$$



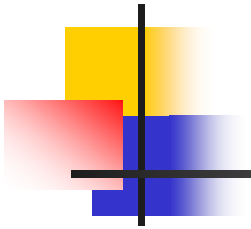
完全弹性碰撞。对于非完全弹性碰撞，可以将$-\mathbf{v}_n$长度减小。

免费开源软件

Ming C. Lin的主页 http://www.cs.unc.edu/~lin/

Collision Detection/Proximity Query Packages

- **I-COLLIDE**
- **RAPID**
- **V-COLLIDE**
- **PQP**
- **H-COLLIDE**
- **SWIFT**
- **PIVOT**
- **SWIFT++**
- **DEEP**



感谢大家选这门课！