二维刚体小球碰撞检测与响应的算法技术报告

1. 引言

在物理模拟与游戏开发中，多小球碰撞计算是常见的计算任务。每次碰撞的计算可分为两个主要任务：碰撞检测与碰撞响应。本文将首先介绍小球碰撞响应的实现方法，随后讨论多小球碰撞检测的几种优化算法。

2. 碰撞响应

在本实验中，我们假设两个小球均为刚体，且碰撞为理想弹性碰撞，遵循动量守恒定律。碰撞响应的核心任务是计算碰撞后小球的速率及方向。

2.1 计算速率

假设小球 1 和小球 2 的当前位置分别为 (x_1, y_1) 和 (x_2, y_2)，当前在x轴和y轴方向的速度分量分别为 (v_{x1}, v_{y1}) 和 (v_{x2}, v_{y2})。可以得到速率分别为$v_1 = \sqrt{v_{x1}^2 + v_{y1}^2}$，$v_2 = \sqrt{v_{x2}^2 + v_{y2}^2}$。

2.2 计算碰撞角度和当前运动方向

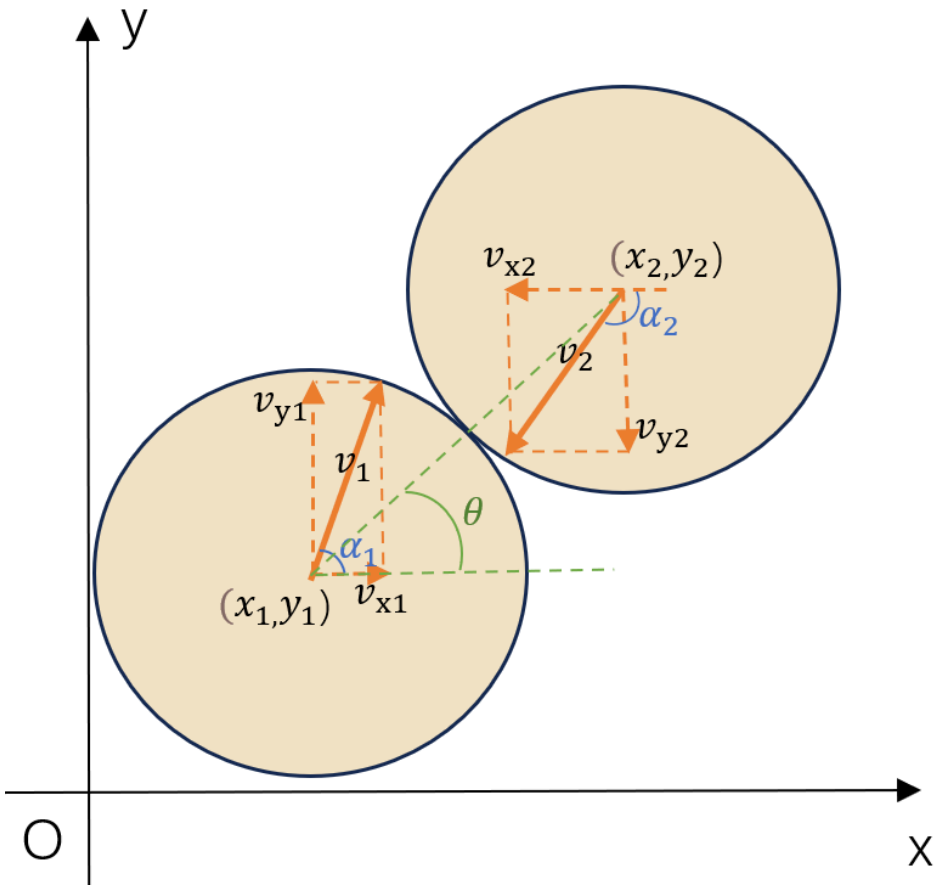
令碰撞角度为 θ，两小球的位置差为$dx = x_2 - x_1$,$dy = y_2 - y_1$。我们引入 $atan2(y, x)$ 函数，该函数可以计算从 x 轴正方向到点 (x, y) 的极角，范围为 $[-\pi, \pi]$。在此处将位置差传入 $atan2$，根据 dx 和 dy 的符号来确定象限，从而得到碰撞角度，即 $\theta = atan2(dy, dx)$。

$$
atan2(y, x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y < 0. \end{cases}
$$

我们还可以用$atan2$函数计算小球的运动方向，$\alpha_1 = atan2(v_{y1}, v_{x1})$ ，$\alpha_2 = atan2(v_{y2}, v_{x2})$。

2.3 速度分量的分解

对于碰撞，我们需要将速度再沿碰撞线及其法线方向分解，通过α和θ作差可以得到这个夹角。对于小球1，有$v'_{x1} = v_1 \cos(\alpha_1 - \theta), \quad v'_{y1} = v_1 \sin(\alpha_1 - \theta)$，小球2同理。图示如下。



2.4 速度更新

基于动量守恒定律，对于小球1在碰撞线方向以及其法线方向速度分量的更新如下。

$$v''_{x1} = \frac{(m_1 - m_2)v'_{x1} + 2m_2 v'_{x2}}{m_1 + m_2}, \quad v''_{y1} = v'_{y1}$$

对于小球2在碰撞线方向以及其法线方向速度分量的更新如下。

$$v''_{x2} = \frac{(m_2 - m_1)v'_{x2} + 2m_1 v'_{x1}}{m_1 + m_2}, \quad v''_{y2} = v'_{y2}$$

最终根据沿碰撞线方向以及其法线方向的速度分量，计算更新后的沿x轴和y轴方向的速度分量。小球1的最终速度如下，小球2的速度更新同理。

$$v'''_{x1} = v''_{x1}\cos(\theta) + v''_{y1}\cos\left(\theta + \frac{\pi}{2}\right)$$

$$v'''_{y1} = v''_{x1}\sin(\theta) + v''_{y1}\sin\left(\theta + \frac{\pi}{2}\right)$$

3. 碰撞检测

碰撞检测分为 Broad-Phase 和 Narrow-Phase 两个阶段。Broad-Phase 旨在筛选出可能发生碰撞的物体，而 Narrow-Phase 则在更精确的范围内进行详细检测。

3.1 Narrow-Phase

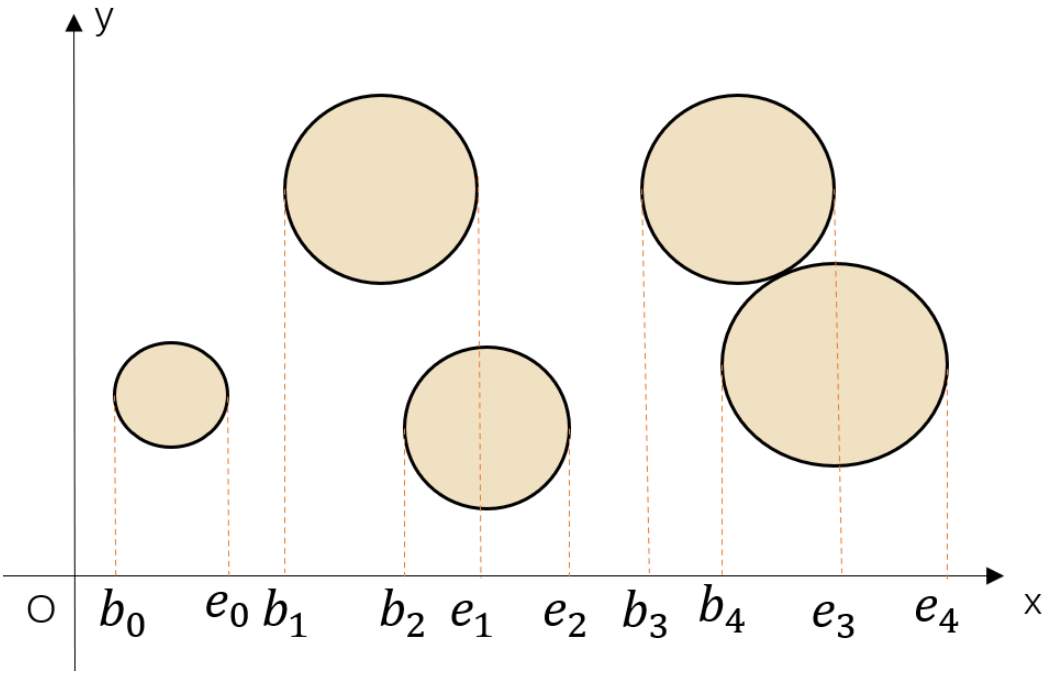
对于两个小球，碰撞检测的条件是两球之间的距离小于等于它们半径之和，即
$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$，$d \leq r_1 + r_2$时两球相撞。

扩展到多个小球的碰撞检测。我们自然而然能想到最朴素的方法：两两比较，时间复杂度为$O(n^2)$，当小球数量较多时效率较低。此时我们就要设计 Broad-Phase ，筛选出可能互相碰撞的小球。接下来讲解几种 Broad-Phase 的优化方法。

3.2 Broad-Phase

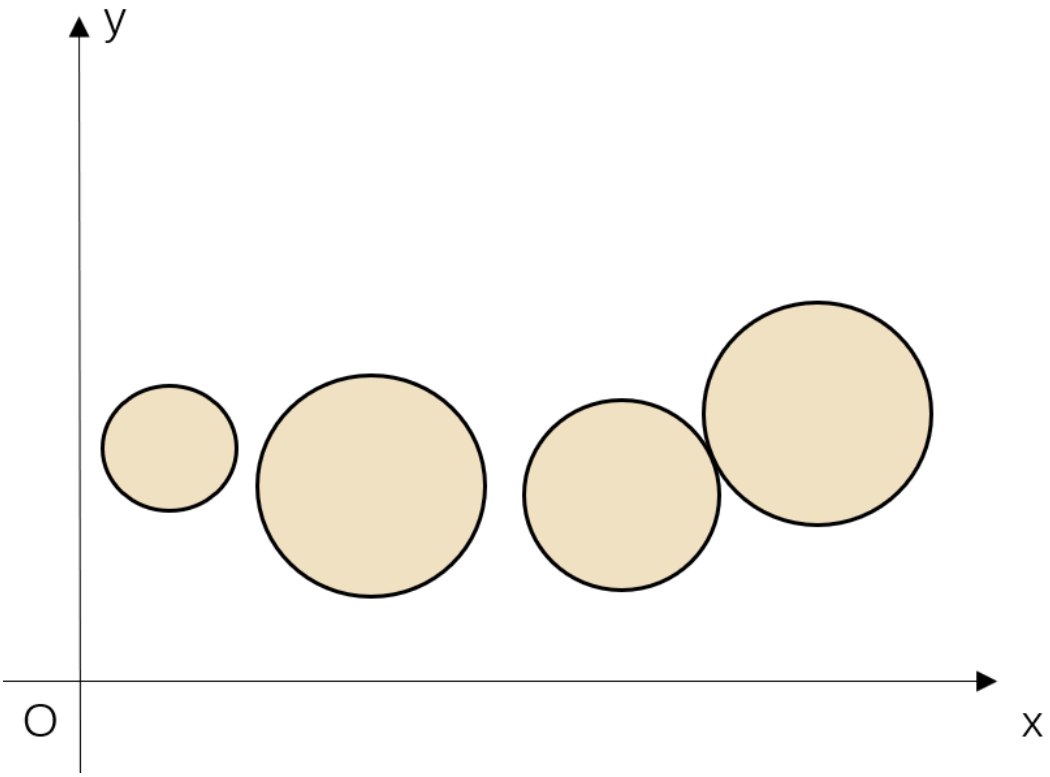
3.2.1 Sort and Sweep

为了优化这一时间，可以采用 Sort and Sweep 方法，首先把小球投影到轴上，就得到了一个起止点数组，如下图，b为起点，e为结束点。如果两个b相邻，说明在两球在这个轴上的投影有重叠。在x、y轴上各进行一次，即可得到各球的重叠情况。
这个算法需要将各球的起点进行排序，时间复杂度为$O(nlogn)$，扫描过程的时间复杂度为$O(n)$，排序和扫描分先后进行，所以该方法的总体时间复杂度为$O(nlogn)$。
在实际的碰撞检测中，由于大部分物体的相对位置不会发生变化，在后续的帧中，只需更新有变化的球的位置，可以通过插入排序进行更新。



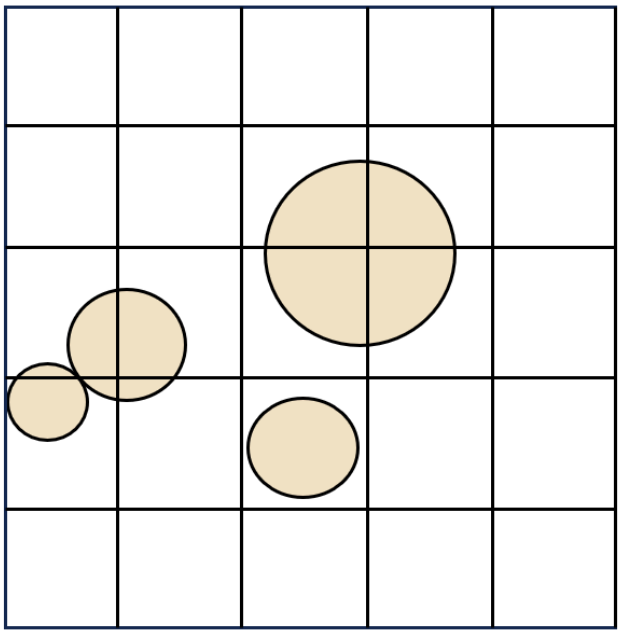
在某些情况下，小球会在某个轴上聚集，会导致该轴进行的筛选结果很差。在这种情况下可以考

虑舍弃该轴的检测。在下图中就可以舍弃对y轴的检测。



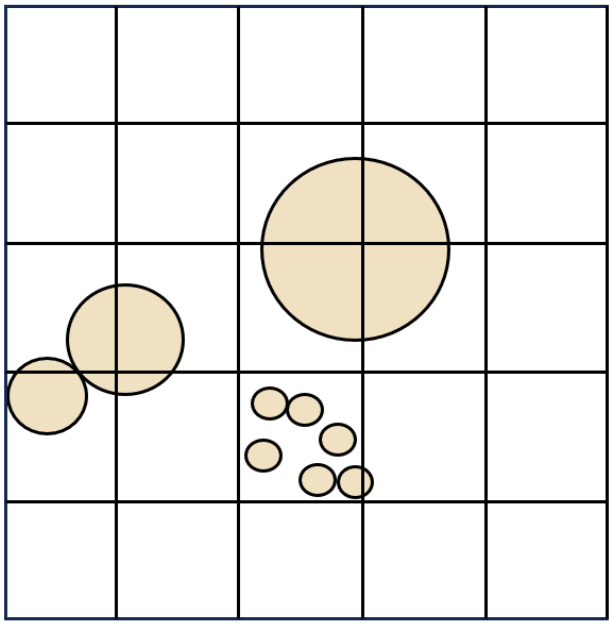
3.2.2 网格划分

网格划分方法将空间划分为多个格子，如果一个小球与某个格子相交，则该格子记录该小球。最终，只有位于同一个格子的物体需要进行碰撞检测。该方法的优势在于可以避免将所有物体进行两两比对。



这种方法需要维护网格、额外保存小球信息。同时，选择一个合适的格子大小至关重要,太大和太小都会导致性能下降。合适的格子大小应该是跟碰撞体大小接近，如果碰撞体的大小本身就差异很大，则很难选择一个合适的大小。

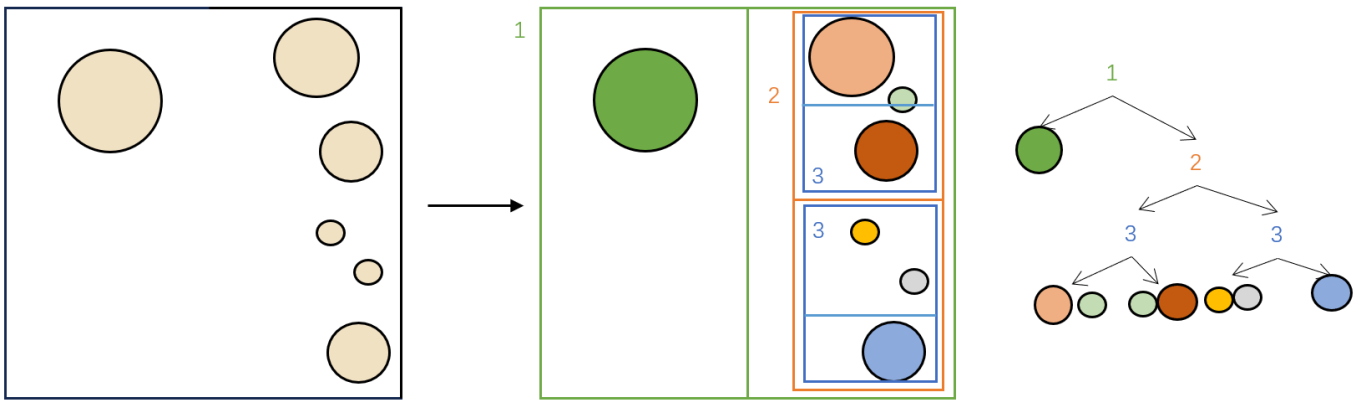
对于碰撞体大小差异很大的，可以使用分层网格，即建立多个不同大小的网格，每个碰撞体都能找到一个适合自己大小的网格。



格内可以使用 Sort and Sweep 进行进一步优化。

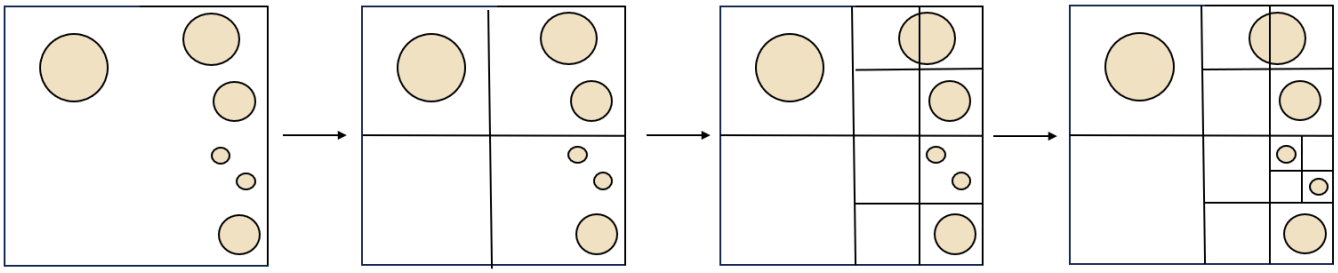
3.2.3 四叉树

在介绍四叉树前，先讲下下图这种分块的思想。与网格固定的划分不同，下图的思路是递归地对当前矩形寻找一个合适的分割轴，尽可能的将碰撞体划分开。其实这个分割操作与上文提到的 Sort and Sweep 很相像，只是前者是轴投影到碰撞体上，后者是碰撞体投影到轴上。
遍历这个通过递归操作生成的二叉树，其实就是在减少需要两两比较的碰撞体个数。



然而上图找分割轴的操作也有点复杂，可以仅递归的将空间划分为4块。这样得到的树会是一棵

四叉树。



4. 实验

基于 pygame 进行图像绘制，用帧率作为性能指标。表格内数据为几种优化手段在不同小球个数下的平均帧率。从表格数据来看各优化算法确实比朴素暴力算法要高效，至于各算法间的对比，该数据仅供参考，因为我只做了基本的实现，还没有做一些细节处理，可能我的实现不如算法理应达到的水平。

小球个数	Brute Force	Sort and Sweep	Grid	Grid & Sort and Sweep	Quadtree
100	138	350	285	285	235
400	13	140	130	145	70
800	4	60	58	65	30
1600	1	19	20	22	11

实验代码见：https://github.com/urlyy/algorithm_analysis_course_tech_report

5. 结论

本报告介绍了二维刚体小球碰撞检测与响应的算法，并对几种常见的优化方法进行了实验分析。通过对不同算法的比较，我们发现 Sort and Sweep、网格划分以及四叉树方法可以有效提高碰撞检测的性能，尤其在小球数量较多时，优化方法的优势更加明显。

6. 扩展

在本实验中，我们仅考虑了二维刚体小球碰撞检测与响应。实际应用中，碰撞检测问题更加复杂，涉及到三维物体、非刚体物体（如布料、液体）以及不规则形状物体的碰撞。这里进行一些知识的补充。

6.1 三维

从二维扩展到三维是比较简单的，原先对于 x、y 轴各进行一次的操作，此时需要再对 z 轴进行一次。而像网格划分也需要考虑 z 轴。对于四叉树，则需要改为八叉树以适应三维空间中的划

分。

6.2 非刚体

非刚体即布料、液体等受到外力会发生形变的物体，碰撞时需要考虑物体的弹性、塑性等材料属性，需要对物体的形变与恢复进行计算，涉及到更高级的物理模拟技术，尽管本报告主要聚焦于刚体小球的碰撞检测与响应，非刚体物体的碰撞处理无疑是物理模拟中的一个重要课题。

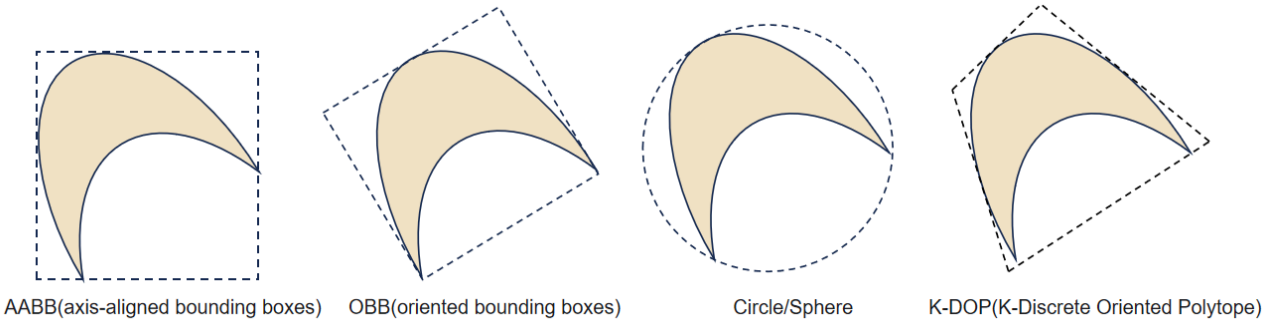
6.3 不规则物体

对于小球这种简单的碰撞体，Narrow-Phase 碰撞检测较为简单，仅需考虑圆心与半径。然而，扩展到更加复杂的形状，如三角形、矩形或不规则图形时，碰撞检测将变得更加复杂。这些不规则物体通常需要使用更加精确的碰撞体定义，并且在碰撞检测过程中需要额外的几何计算。

6.3.1 定义碰撞体形状

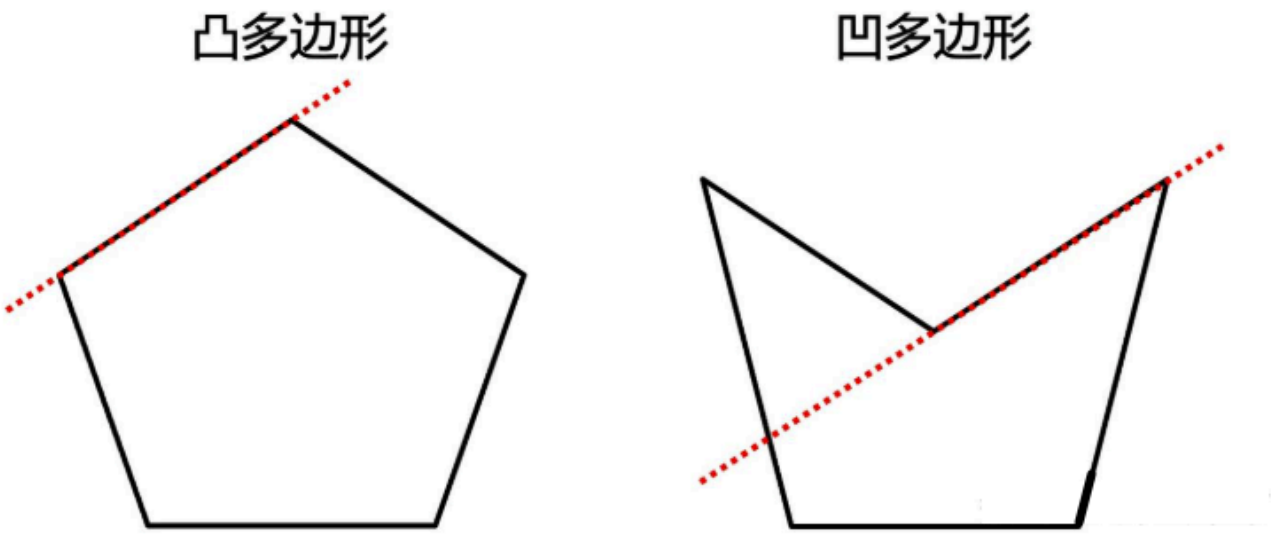
常见的碰撞体形状包括：

- AABB：直接基于 x 和 y 轴设置边界的矩形框。
- OBB：通过计算物体的旋转角度，使用旋转后的矩形框来减少碰撞盒的大小。
- Circle：将碰撞体看作一个圆形，适用于圆形物体。
- K-DOP：通过多个轴来拟合碰撞体的边缘，通常用于更复杂的几何形状。

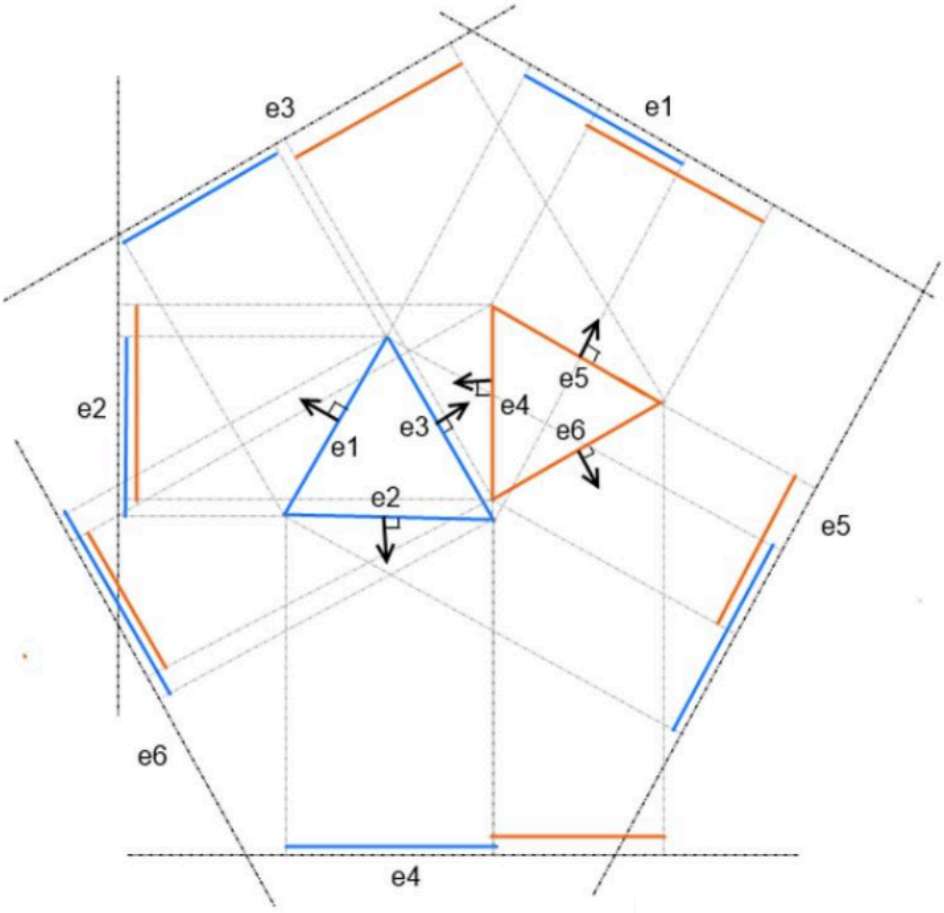


6.3.2 设计Narrow-Phase

凸多边形的定义非常重要：对于平面上的一个多边形，如果延长它的任何一条边，都使整个多边形位于一边延长线的同侧，这样的多边形叫做凸多边形。



1. 分离轴算法(Separating Axis Theorem, SAT)。只能用于凸多边形，在两个碰撞体比较时，使用边的垂线作为分离轴。如果存在一个投影不重叠,则可判定不相交；反之,如果所有边的垂线的投影都重叠,则判定相交。



2. GJK算法，基于闵可夫斯基差，较SAT更高效。由于 GJK 算法的细节较为复杂，本文不进行深入讨论。

6.3.3 碰撞穿透

在传统的离散碰撞检测中，碰撞检测是基于帧与帧之间的离散时间步长进行的，这可能导致一些物体由于运动速度过快而穿透其他物体，这种现象被称为隧穿问题。

例如，假设一个子弹以非常高的速度向墙壁移动，在当前帧中，子弹位于墙壁的左部，而在下一帧中，由于子弹的速度过快，它可能直接穿过墙壁到达墙壁右侧，导致没有发生预期的碰撞。这种问题在高速物体模拟中尤为常见。

为了解决这一问题，传统的离散碰撞检测方法需要扩展为连续碰撞检测。在连续碰撞检测中，我们不再仅仅判断物体是否发生碰撞，而是追踪物体的运动轨迹，检查物体在两个时间步之间是否穿透了其他物体。这样可以有效避免因离散化时间步长而导致的碰撞穿透问题。相关的知识体系较为复杂，本文不进行深入讨论。

7. 参考

- 降低维度，分而治之，解决多个小球碰撞问题——优雅永不过时_哔哩哔哩_bilibili
- 动手学运动规划:1.3 碰撞检测算法:AABB, SAT - 知乎
- 游戏物理引擎(二) 碰撞检测之Broad-Phase - 知乎