

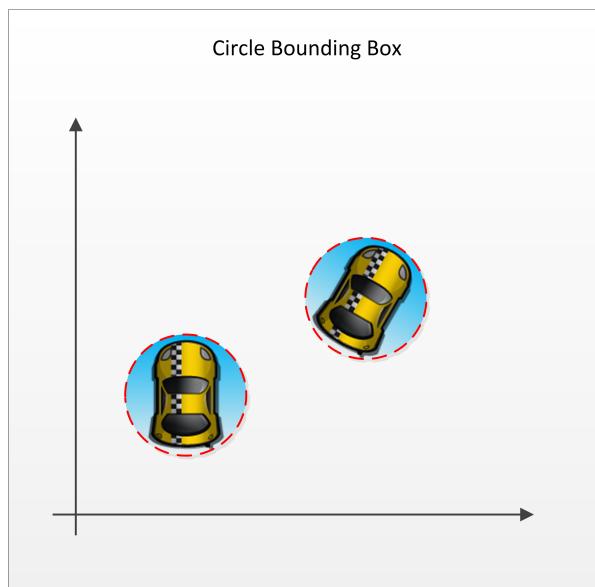
# 2d collision detection

-bobding 2014/10/29

## 引言

碰撞检测在游戏开发使用非常多，几乎每款游戏都有使用，只是根据游戏类型的的不同，使用方式也简繁不一。本文主要介绍几种常用的检测方法，有不足之处，还请大家指正。

## 包围圆



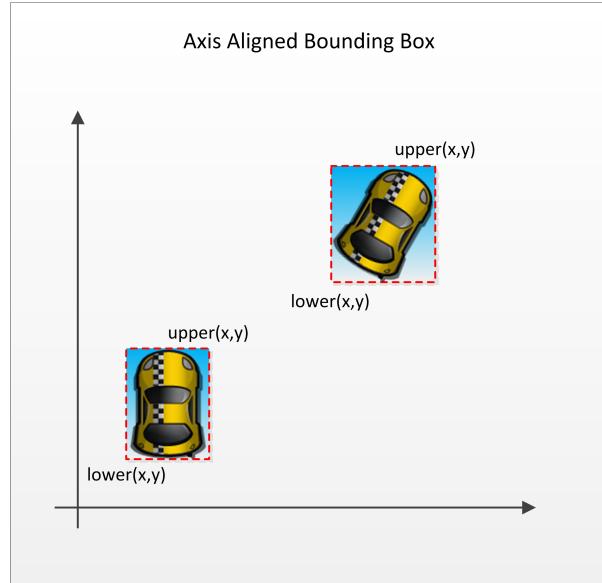
包围圆由一个圆心和半径构成，检测时根据两圆心距离与两半径和的大小确定是否相交。这种检测方式简单快速，但精确度较低。

包围盒数据结构为

```
struct circle {  
    vector2 pivot;  
    float radius;  
};
```

其相交判定公式为：  $\text{radius1} + \text{radius2} > \text{distance}(\text{pivot1}, \text{pivot2})$

## 坐标轴对齐包围盒 AABB(Axis Aligned Bounding Box)



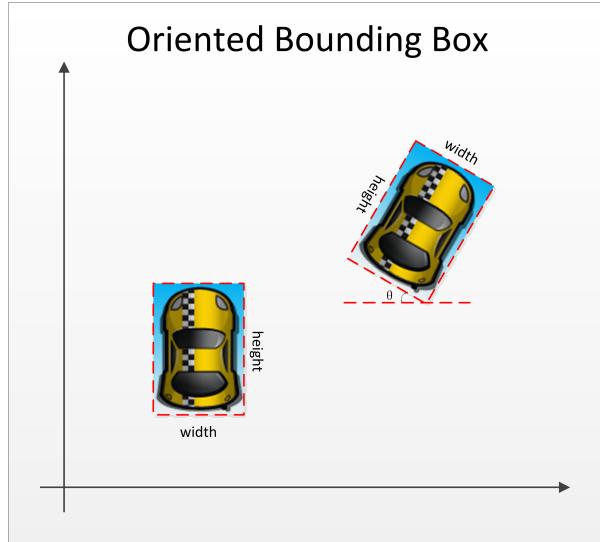
AABB 由平行于坐标轴的矩形框表示，我们这里使用左下右上两个顶点表示。和包围圆类似，这种检测方式简单，对非特定形状（非旋转矩形）精确度较低，例如上图中的车辆旋转后，其 AABB 区域被扩大。

包围盒数据结构为：

```
struct aabb {  
    vector2 lower;  
    vector2 upper;  
};
```

其相交判定公式为：  $(\text{lower1} - \text{upper2}).xy > 0 \ \&\& (\text{lower2} - \text{upper1}).xy > 0$

## 方向包围盒 OBB(Oriented Bounding Box)



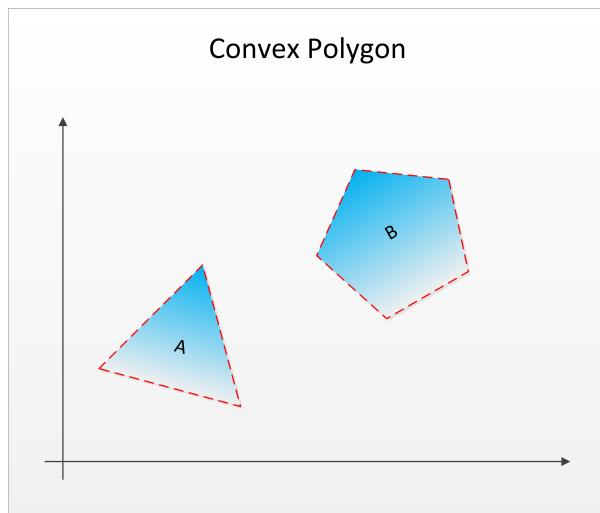
OBB 由一个可旋转的矩形框表示，和 AABB 相比能更精确的表示旋转物体的碰撞区域，其碰撞算法也稍复杂，后文再做详细介绍。

包围盒数据结构：

```
struct obb {
    vector2 pivot;
    vector2 size;
    float rotation;
};
```

相交判断主要用到分离轴定理 SAT(Separating Axis Theorem)

## 凸多边形包围盒 Convex Polygon



凸多边形可以比 OBB 有更多的边，因此可以更精确的表示物体碰撞区域，但其检测算法耗时也随边数呈线性增长。其相交判断原理和 OBB 一样。

凸多边形数据结构：

```
struct convex {  
    vector2* vertices;  
    size_t numVerts;  
};
```

相交判断主要用到分离轴定理 SAT(Separating Axis Theorem)

## 向量代数

为了更好的理解后文中的算法，我们要了解一些向量代数的知识。如果对向量运算很熟悉，可以略过这段。本文只讲解使用到的向量知识，不做扩展。

2d 向量数据结构：

```
struct vector2 {  
    float x;  
    float y;  
};
```

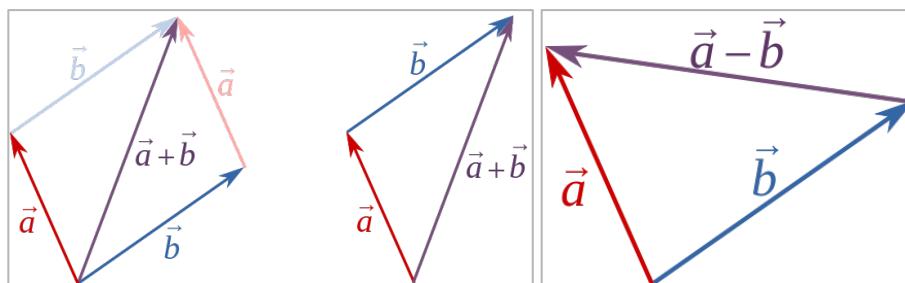
向量模：float magnitude = sqrt(v.x \* v.x + v.y \* v.y)，几何意义是向量长度。

**重要：**当向量模为 1 时，该向量称作基向量，本文中我们用来表示分离轴。

向量加法：vector2 result = vector2(v1.x + v2.x, v1.y + v2.y)

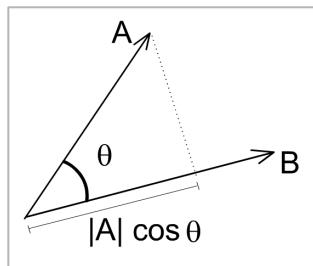
向量减法：vector2 result = vector2(v1.x - v2.x, v1.y - v2.y)

向量加减法的几何意义（图片来自维基百科）：



向量点乘： $\text{float result} = \text{v1.x} * \text{v2.x} + \text{v1.y} * \text{v2.y} = |\text{v1}| * |\text{v2}| * \cos(\theta)$ , 其中 $\theta$ 为两向量夹角。

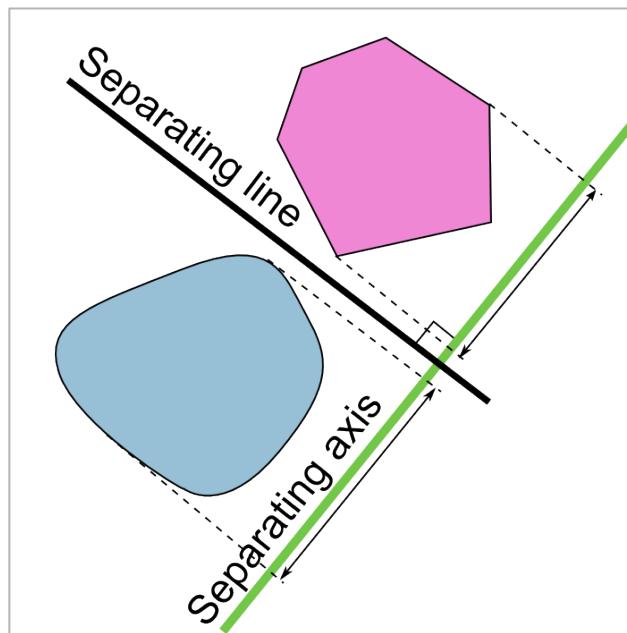
向量点乘的几何意义（图片来自维基百科）：



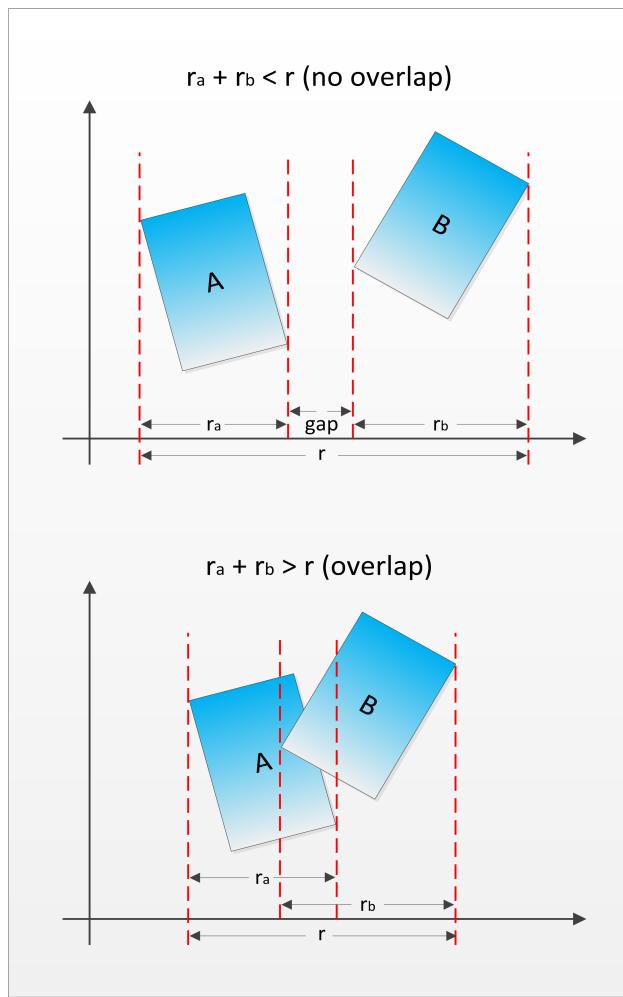
重要：当 v2 为基向量时，v1 和 v2 的点积就是 v1 在 v2 上的投影长度。

### 分离轴定理 SAT(Separating Axis Theorem)

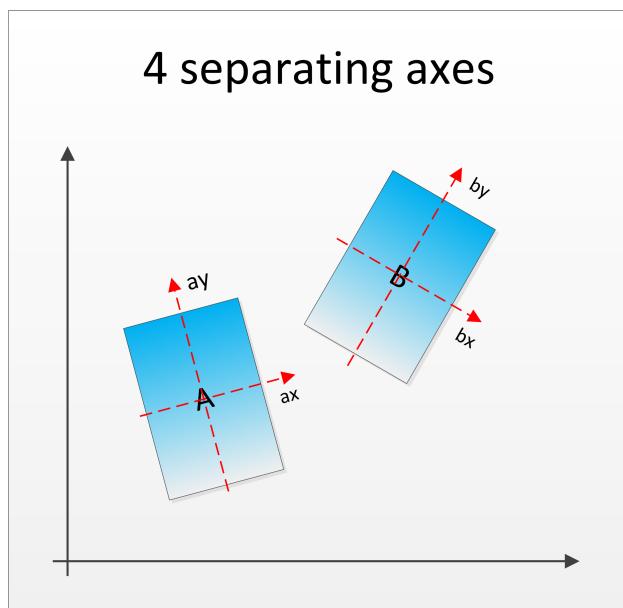
定理描述：如果存在一条轴使两个凸多边形上的点处于轴两侧，则这两个凸多边形不相交。（图片来自维基百科）



对 OBB 使用 SAT 进行相交检测



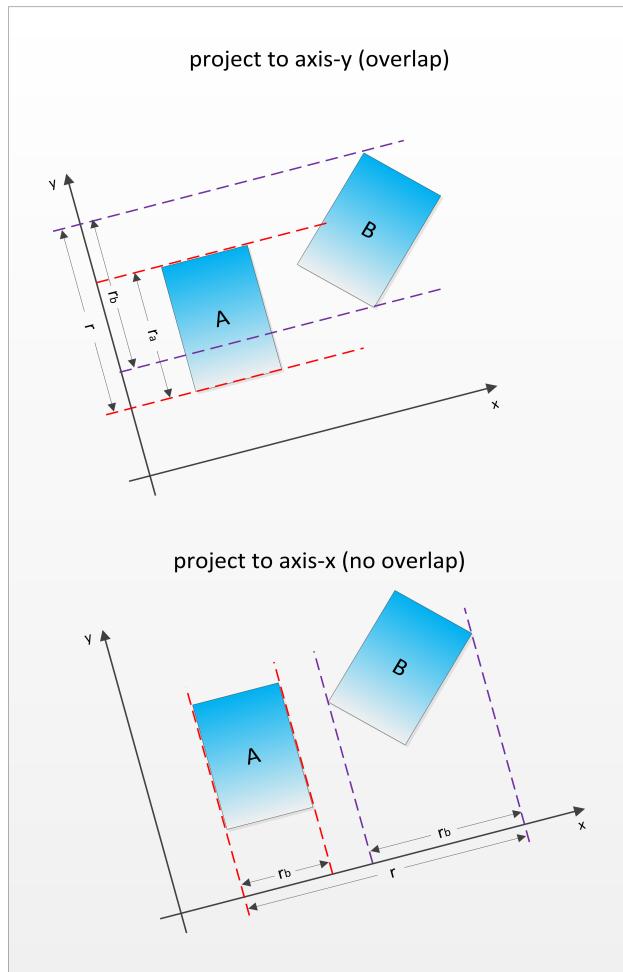
## 分离轴选取



根据 SAT 定理的描述，若两个凸多边形不相交则存在多个分离轴，我们不能一一验证。根据凸多边形的性质：凸多边形上点在边的同侧，我们可以只对边法线进行检测，这样的轴共有 8 条，但两两平行，所以只需要检测 4 条轴。

### 检测步骤（可以对应下图理解）

1. 计算 4 个分离轴；
2. 计算旋转矩形 A 的四个顶点在一个分离轴上的投影，确定最大和最小值；
3. 计算旋转矩形 B 的四个顶点在一个分离轴上的投影，确定最大和最小值；
4. 计算 A, B 在分离轴上的投影长度  $r_a, r_b$ , 计算 A, B 在分离轴上的投影和  $r$ ；
5. 如果  $r_a + r_b > r$ , 在该分离轴上相交，到步骤 2 检测下一分离轴；
6. 如果  $r_a + r_b < r$ , 则不相交，退出检测；
7. 直到所有分离轴都检测完成，并且都有相交，则这两个凸多边形相交。



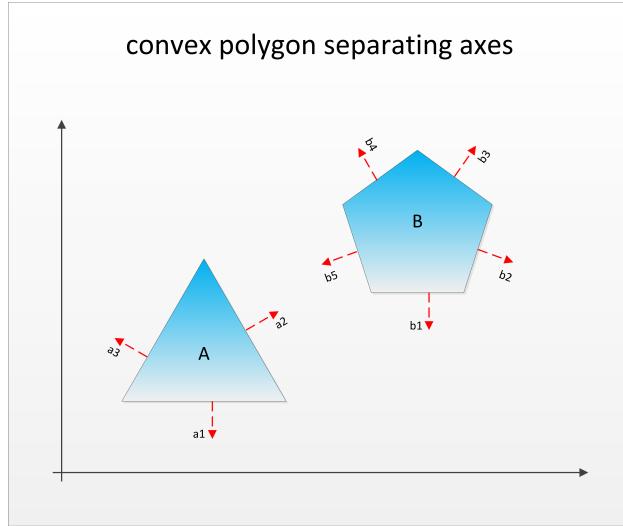
分离轴（边法线）的计算（代码取自 demo，由 js 编码）

```
// find axes  
  
var axes = new Array(4);  
  
axes[0] = new vector2( Math.cos(o1.rotation), Math.sin(o1.rotation));  
axes[1] = new vector2(-Math.sin(o1.rotation), Math.cos(o1.rotation));  
axes[2] = new vector2( Math.cos(o2.rotation), Math.sin(o2.rotation));  
axes[3] = new vector2(-Math.sin(o2.rotation), Math.cos(o2.rotation));
```

投影长度的计算：float result = vertices[j].dot(axis[i])。如果不明白，请参考上文的向量点积的内容。

### 对图多边形使用 SAT 进行相交检测

同理，我们可以将 SAT 从 OBB 扩展到凸多边形上，检测步骤则完全一样。分离轴的选取则是每一条边的法线，如下图：



convex 的分离轴计算方法（代码取自 demo，由 js 编码）

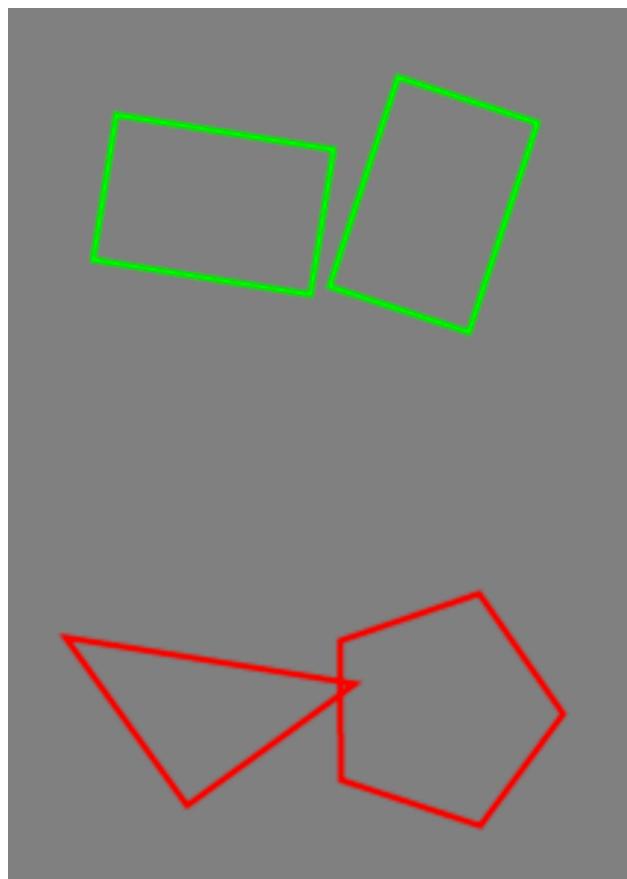
```
// find axes  
  
var numVerts = c1.numVerts + c2.numVerts;  
  
var axes = new Array(numVerts);  
  
var i = 0, j = 0;
```

```
for (i = 0; i < c1.numVerts; ++i, ++j)
{
    var v = c1.vertices[i].sub(c1.vertices[(i + 1) % c1.numVerts]);
    v.normalize();
    axes[j] = new vector2(-v.y, v.x);
}

for (i = 0; i < c2.numVerts; ++i, ++j)
{
    var v = c2.vertices[i].sub(c2.vertices[(i + 1) % c2.numVerts]);
    v.normalize();
    axes[j] = new vector2(-v.y, v.x);
}
```

## 代码演示

Talk is cheap, let's show the code. 原先的代码有 C++ 构成，但依赖于游戏逻辑，所以将其抽取并使用 javascript 改编。虽然和原始代码稍有不同，但原理不变。这里给出运算结果：如果多边形相交，则显示红色，否则现实绿色。



## 算法优化

针对 obb 的检测，对每个分离轴我们使用了 8 个顶点的点积运算，这里可以改为计算长短半轴的投影和两矩形中心点距离相比较。这样能减少为 4 次点积运算。

## 进一步的优化

我们可以为物体建立多套包围盒，比如同时有包围圆, AABB, OBB，这样首先使用包围圆进行粗略剔除，然后看是否有旋转，若物体没有旋转则使用 AABB，最后才使用 OBB，这样在一定程度上可以提升性能。

再进一步的优化，则是对物体进行空间分割，这项技术暂不在这儿展开，后面有空再写。

以上就是本文的全部内容，还请大家指点。更详细的细节请参见附件中的代码。