数学美 之 判断线段相交的最简方法

向量 计算几何 数学 **hsfzxjy** 2016年02月19日发布

> 首发于我的博客 **转载请注明出处**

> 解析几何的巅峰
> 是 向量
> 那无关过程的狂妄与简洁
> 映射着大自然无与伦比的美

引子

如何判断两条直线是否相交？

这很容易。平面直线，无非就是两种关系：相交 或 平行。因此，只需判断它们是否平行即可。而直线平行，等价于它们的斜率相等，只需分别计算出它们的斜率，即可做出判断。

但倘若我把“直线”换成“线段”呢——如何判断两条线段是否相交？

这就有些难度了。和 直线 不同，线段 是有固定长度的，即使它们所属的两条直线相交，这两条线段也不一定相交。

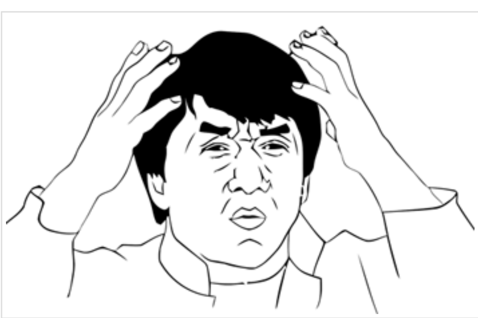
也许你会说：分情况讨论不就行了嘛：

- 先计算两条线段的斜率，判断是否平行。若平行，则一定不相交。
- 若不平行，求出两条线段的直线方程，联立之，解出交点坐标。
- 运用定比分点公式，判断交点是否在两条线段上。

的确，从理论上这是一个可行的办法，这也是人们手动计算时普遍采用的方法。

然而，这个方法并不怎么适用于计算机。原因如下：

- 计算中出现了除法（斜率计算、定比分点），因此每次计算前都要判断除数是否为 0（或接近 0）。这很麻烦，严重干扰逻辑的表达。
- 浮点精度丢失带来的误差。人类计算时可以采用分数，但计算机不行。计算机在储存浮点数时会有精度丢失的现象。一旦算法的计算量大起来，误差会被急剧放大，影响结果准确性。
- 效率低下。浮点乘除会十分耗时，不适用于对实时性要求较高的生产环境（如 游戏）。



那么，有更好的方法？

当然有。

类型预定义

本文的算法将用 python 描述，主要用到两个数据类型：

```
# 点
class Point(object):

    def __init__(self, x, y):
        self.x, self.y = x, y

# 向量
class Vector(object):

    def __init__(self, start_point, end_point):
        self.start, self.end = start_point, end_point
        self.x = end_point.x - start_point.x
        self.y = end_point.y - start_point.y
```

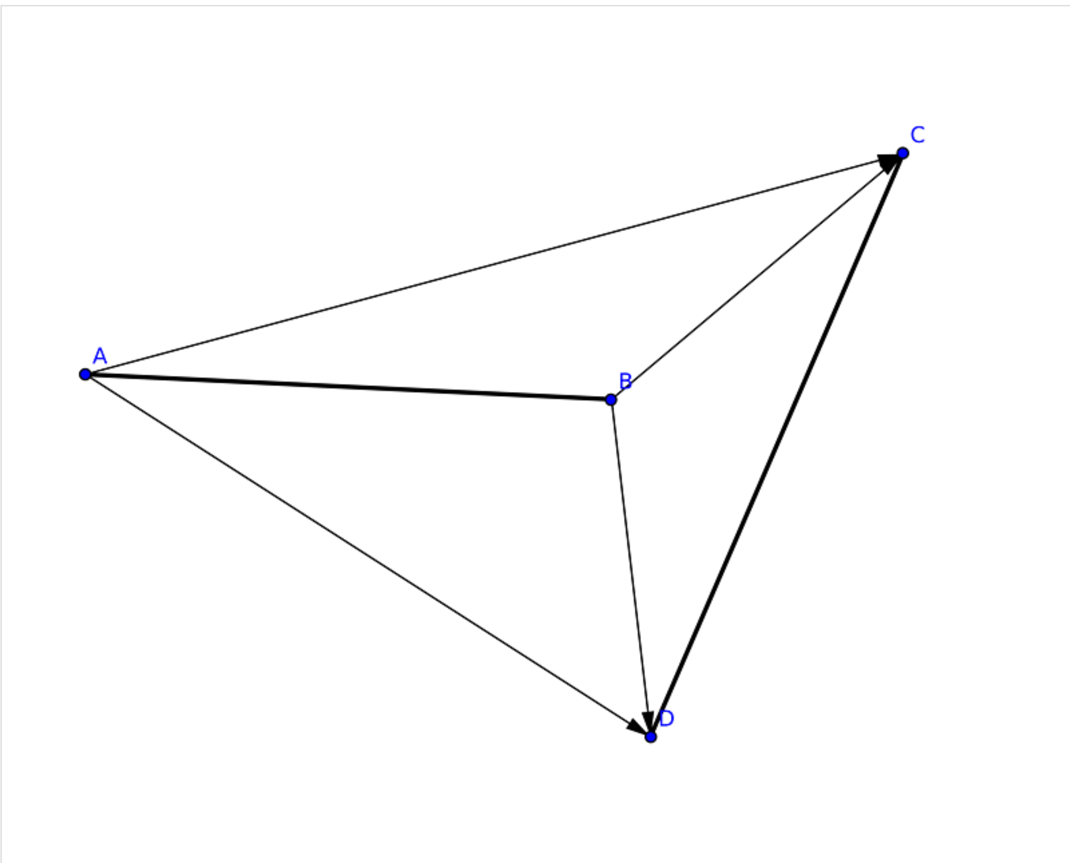
先在此处说明。

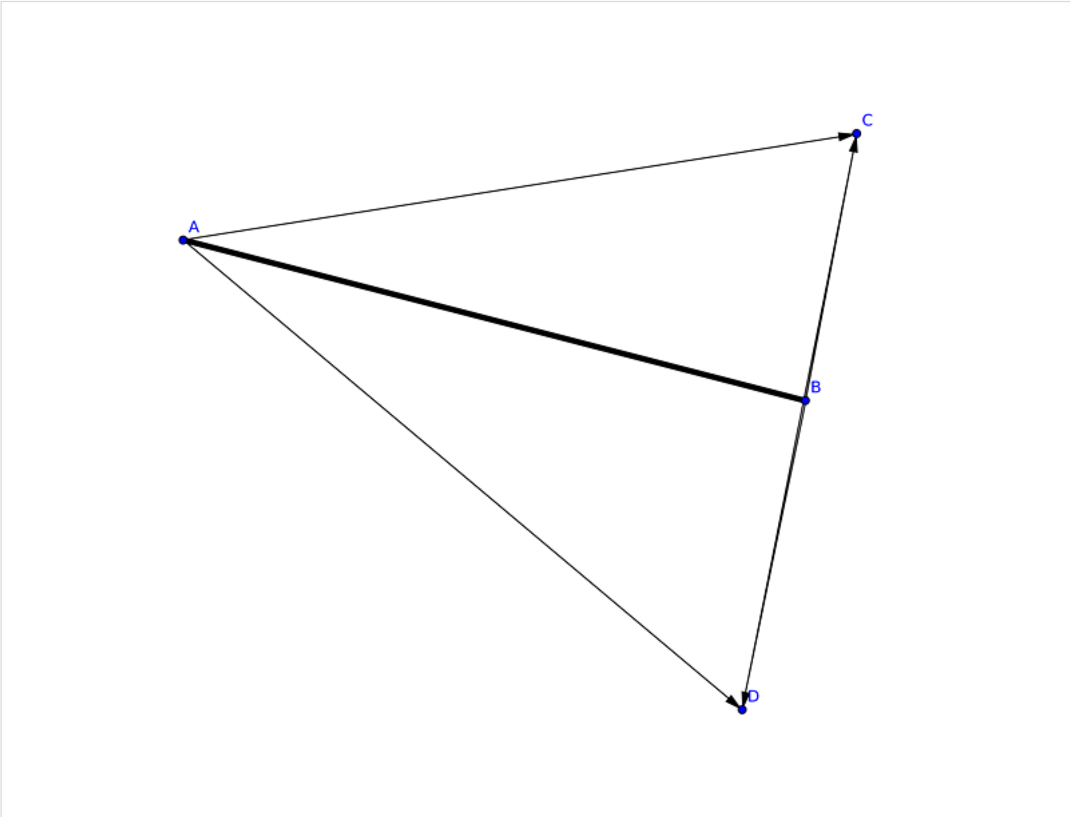
问题分析

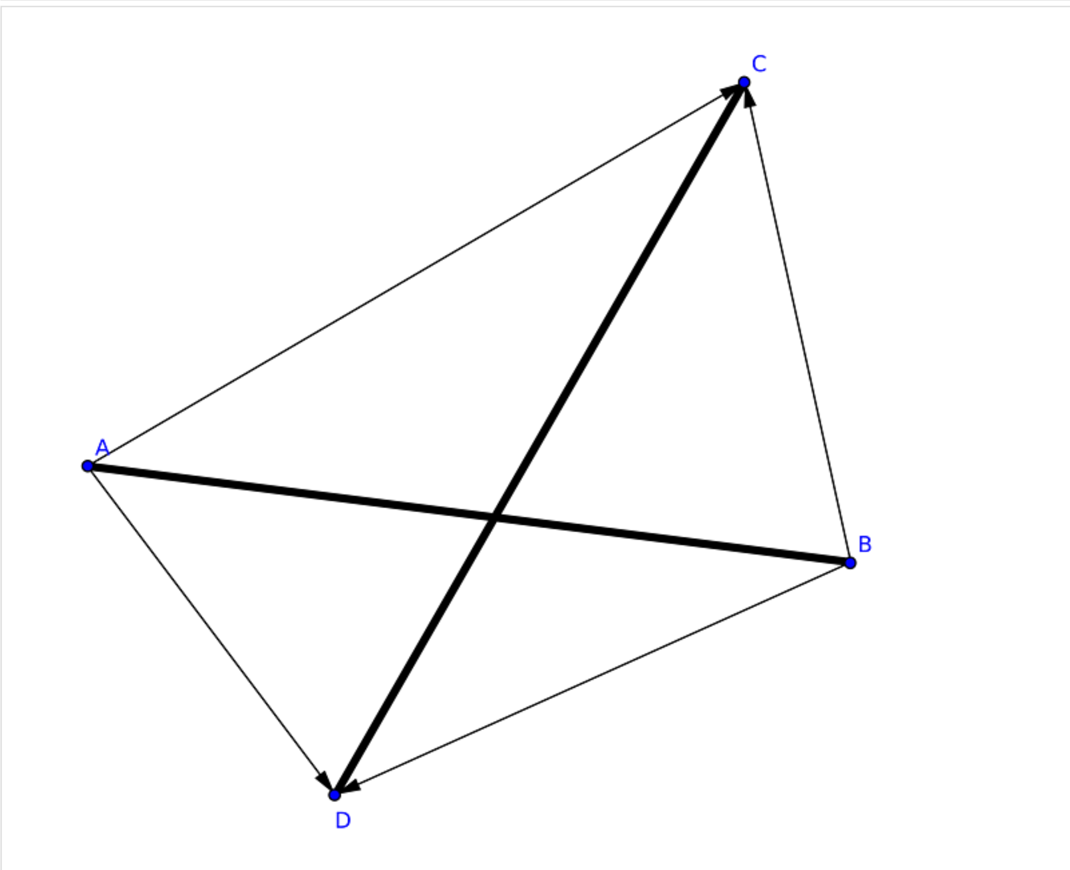
对于“判断两条直线是否相交”这个问题，我们之所以能迅速而准确地进行判断，是因为“相交”与“不相交”这两个状态有着明显的不同点，即 **斜率是否相等。**

那么现在，为了判断两条线段是否相交，我们也要找出“相交”与“不相交”这两个状态的不同点。

假设现在有两条线段 AB 和 CD，我们画出它们之间的三种关系：

1. 

2. 

3. 

其中，情况 1 为不相交，情况 2、3 为相交。

作出向量 AC、AD、BC、BD。

首先介绍一个概念： **向量有序对的旋转方向。**这个概念指：对于共起点有序向量二元组 `(a, b)` ，其旋转方向为 **使 a 能够旋转一个小于 180 度的角并与 b 重合的方向**，简记为 `direct(a, b)` 。若 `a` 和 `b` 反向共线，则旋转方向取任意值。

举个例子：图一中， `direct(AC, AD)` 为顺时针方向。

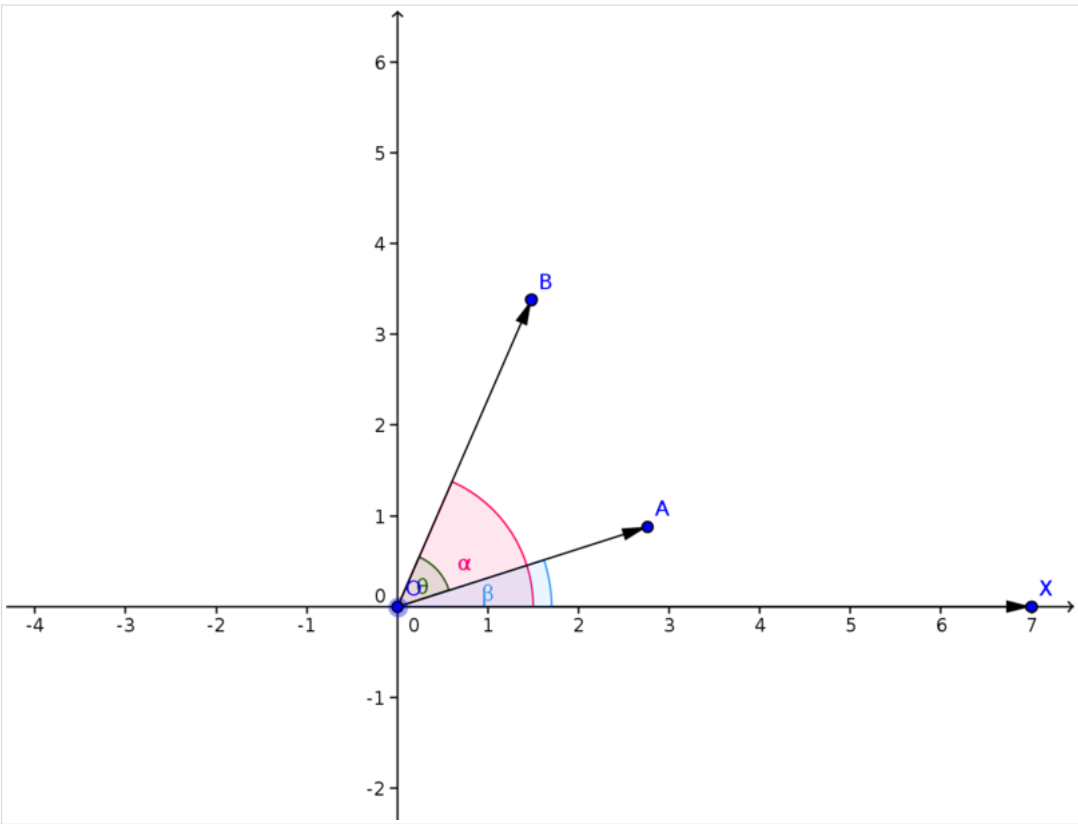
接下来我们要分析四个值： `direct(AC, AD)` 、 `direct(BC, BD)` 、 `direct(CA, CB)` 、 `direct(DA, DB)` 。

1. 对于图一， `direct(AC, AD)` 和 `direct(BC, BD)` 都为顺时针， `direct(CA, CB)` 为逆时针， `direct(DA, DB)` 为顺时针。
2. 对于图二， `direct(AC, AD)` 为顺时针， `direct(BC, BD)` 为任意方向， `direct(CA, CB)` 为逆时针， `direct(DA, DB)` 为顺时针。
3. 对于图三， `direct(AC, AD)` 、 `direct(DA, DB)` 为顺时针， `direct(BC, BD)` 、 `direct(CA, CB)` 为逆时针。

不难发现，两条线段相交的充要条件是： `direct(AC, AD) != direct(BC, BD)` 且 `direct(CA, CB) != direct(DA, DB)` 。这便是“相交”与“不相交”这两个状态的不同点。

然而你可能会觉得：旋转方向这么一个虚无飘渺的东西，怎么用程序去描述啊？

再来看一幅图：



再来定义有向角：

> 有向角 `<a, b>` 为 向量 `a` **逆时针** 旋转到与 向量 `b` 重合所经过的角度。

不难看出，对于向量 `a` 、 `b` ：

- 若 `direct(a, b)` 为逆时针，则 `0 <= <a, b> <= 180` ，从而 `sin<a, b> >= 0` 。
- 若 `direct(a, b)` 为顺时针，则 `180 <= <a, b> <= 360` ，从而 `sin<a, b> <= 0` 。

这样一来，我们可以将旋转方向的问题转化为 **求有向角正弦值** 的问题。而这个问题，是很容易的。

如上图，记

$$OA = (x_1, y_1), OB = (x_2, y_2)$$

$$|OA| = r_1, |OB| = r_2$$

则

$$sin(ltOA, OBgt)$$

$$= sintheta$$

$$= sin(alpha - beta)$$

$$= sinalphacosbeta - sinbetacosalpha$$

$$= frac(sinalphacosbeta - sinbetacosalpha) * r_1 * r_2 r_1 * r_2$$

$$= fracx_1 * y_2 - x_2 * y_1 r_1 * r_2$$

而这里，我们要的只是 `sin(<OA, OB>)` 的符号，而 `r1` 和 `r2` 又都是恒正的，因此只需判断 `x1 * y2 - x2 * y1` 的符号即可。

这个方法的数学背景是 **叉乘**，可以前往 Wikipedia 了解更多。

思路小结

- 由点 A，B，C，D 计算出向量 AC，AD，BC，BD
- 计算 `sin(<AC, AD>) * sin(<BC, BD>)` 和 `sin(<CA, CB>) * sin(<DA, DB>)`，若皆为非正数，则相交；否则，不相交。

实现

终于到代码部分了，想必大家都已不耐烦了吧。

在向量的辅助下，代码显得异常简单。

```
ZERO = 1e-9

def negative(vector):
    """取反"""
    return Vector(vector.end_point, vector.start_point)

def vector_product(vectorA, vectorB):
    '''计算 x_1 * y_2 - x_2 * y_1'''
    return vectorA.x * vectorB.y - vectorB.x * vectorA.y

def is_intersected(A, B, C, D):
    '''A, B, C, D 为 Point 类型'''
    AC = Vector(A, C)
    AD = Vector(A, D)
    BC = Vector(B, C)
    BD = Vector(B, D)
    CA = negative(AC)
    CB = negative(BC)
    DA = negative(AD)
    DB = negative(BD)

    return (vector_product(AC, AD) * vector_product(BC, BD) <= ZERO) \
        and (vector_product(CA, CB) * vector_product(DA, DB) <= ZERO)
```

一气呵成，没有恼人的除法，没有情况讨论，只是纯粹的简单运算。

2016年02月19日发布 更多

特征值与特征向量的几何含义(转) 4 收藏，1.2k 浏览

---

7 条评论 默认排序

丁亚光 · 2016年03月07日

没有考虑两条线段在一条直线上的情况

赞 +1 回复

桂林的小河 · 2016年02月22日

厉害

赞 回复

---

hsfzxjy 作者 · 2016年02月22日

(^ω^)退役竞赛党，以后会常分享这类知识

赞 回复

---

bf · 2016年05月31日

不过传统方法其实也可以用向量哦

```
// segmentA: pointA1 = p, pointA2 = p + r
// segmentB: pointB1 = q, pointB2 = q + s
function segmentIntersection(p, r, q, s) {
    let rxs = Vec.cross(r, s);

    if (rxs === 0) {
        return false; // intersection may not be a point
    } else {
        let pq = Vec.minus(q, p);
        let pqxr = Vec.cross(pq, r);
        let pqxs = Vec.cross(pq, s);
        let u = pqxr / rxs;
        let t = pqxs / rxs;

        return 0 <= u && u <= 1 && 0 <= t && t <= 1;
    }
}
```

无论从代码量上还是运算量上似乎都比你的方法更简单哦

赞 回复

---

bf · 2016年05月31日

以及按照向量外积的性质，你的代码似乎可以简化成

```
def is_intersected(A, B, C, D):
    '''A, B, C, D 为 Point 类型'''
    AC = Vector(A, C)
    AD = Vector(A, D)
    BC = Vector(B, C)
    BD = Vector(B, D)

    return (vector_product(AC, AD) * vector_product(BC, BD) <= ZERO) \
        and (vector_product(AC, BC) * vector_product(AD, BD) <= ZERO)
```

赞 回复

Chivalee · 2016年09月22日

请问这篇blog 可有参考文献？

赞 回复

---

hsfzxjy 作者 · 2016年09月22日

无，纯冥想

赞 回复

文明社会，理性评论

发布评论

hsfzxjy

16.3k 声望

关注作者

发布于专栏

Thoughts of hsfzxjy

“那，我要怎么做呢" “要多想” “然后呢” “孩子，这就足够了”

28 人关注

关注专栏

分享扩散：