一种光标查找算法

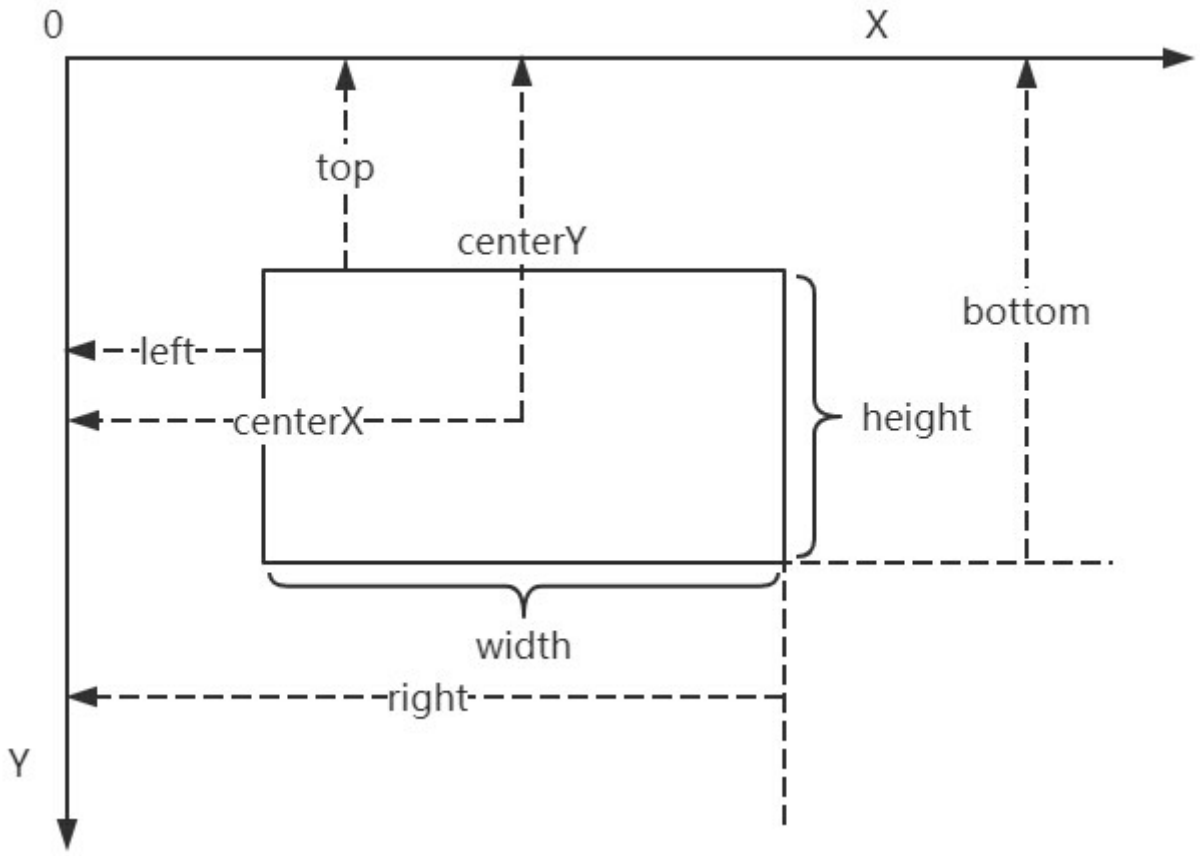
本文描述一种光标移动过程中自动查找下一个最佳光标的算法。

目的

解决光标移动规则需要事先编写且编写繁琐的问题。

定义

- 光标：即按钮，表示页面当前焦点位置所呈现的视觉效果组件，常见效果如光标放大、边框发光等。每个光标都有如下几个属性：
- left：光标左边线X坐标。
- right：光标右边线X坐标，right = left + width。
- top：光标上边线Y坐标。
- bottom：光标下边线Y坐标，bottom = top + height。
- centerX：光标中点X坐标，centerX = left + width/2。
- centerY：光标中点Y坐标，centerY = top + height/2。
- width：光标宽度。
- height：光标高度。

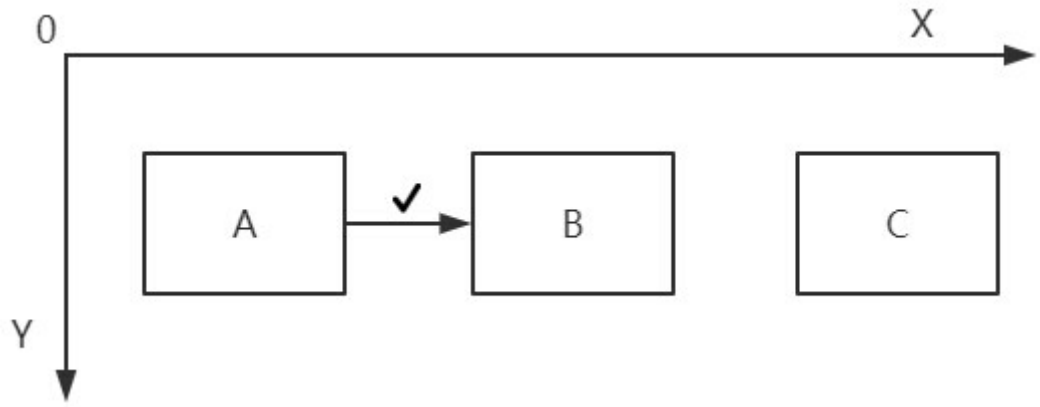


约定

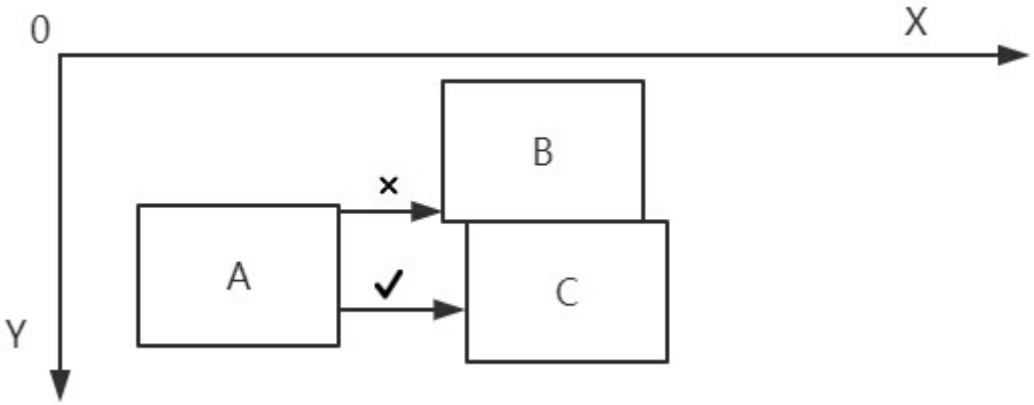
以下示例均以光标向右移动为准。

尝试过的算法

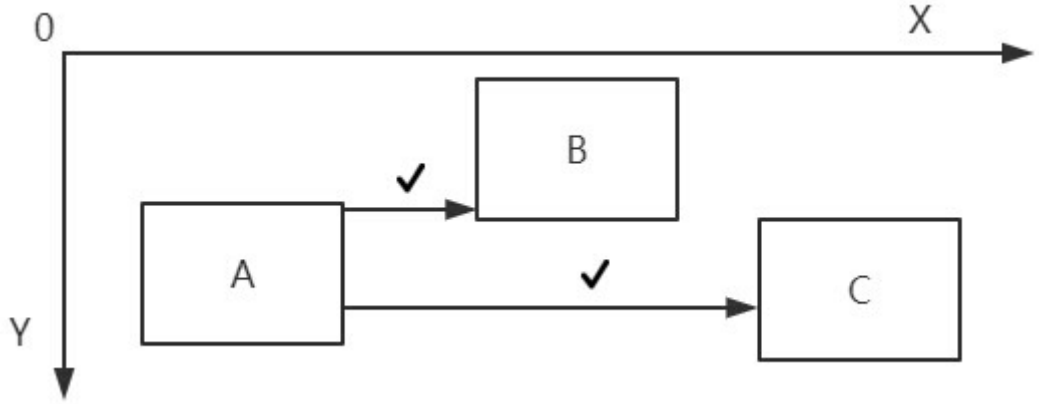
- **就近原则**：即光标向右移动后与之最早产生碰撞的光标即为最佳光标，如下图所示：



A向右移动时与之最早产生碰撞的是B，故B为目标光标，但是出现以下布局时该原则不成立。

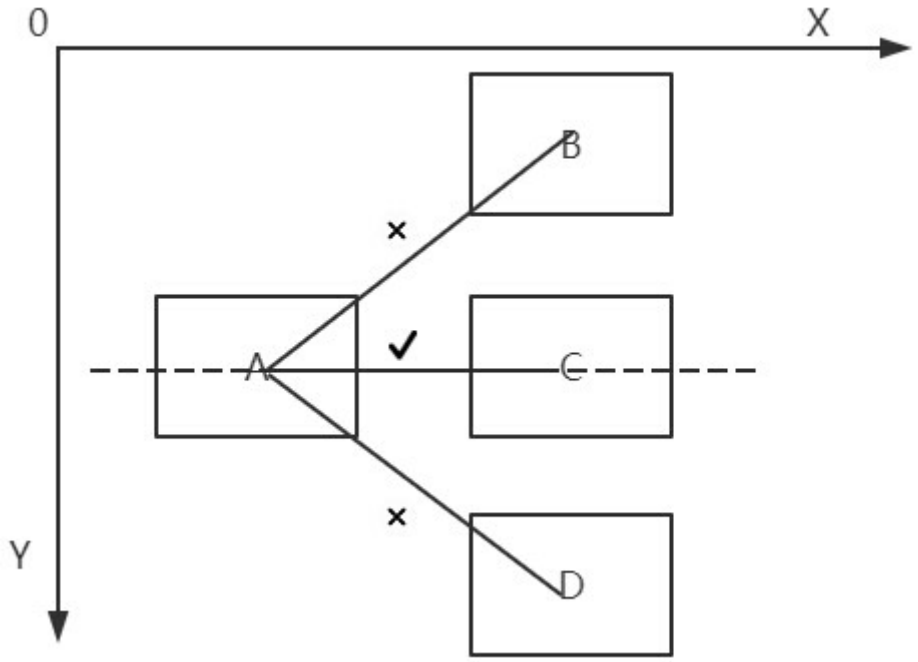


虽然A向右移动最早与之碰撞的是B，但是明显不合理，因为C与A同处水平方向，且产生的交集更大，但是出现以下布局时情况又不明朗了。

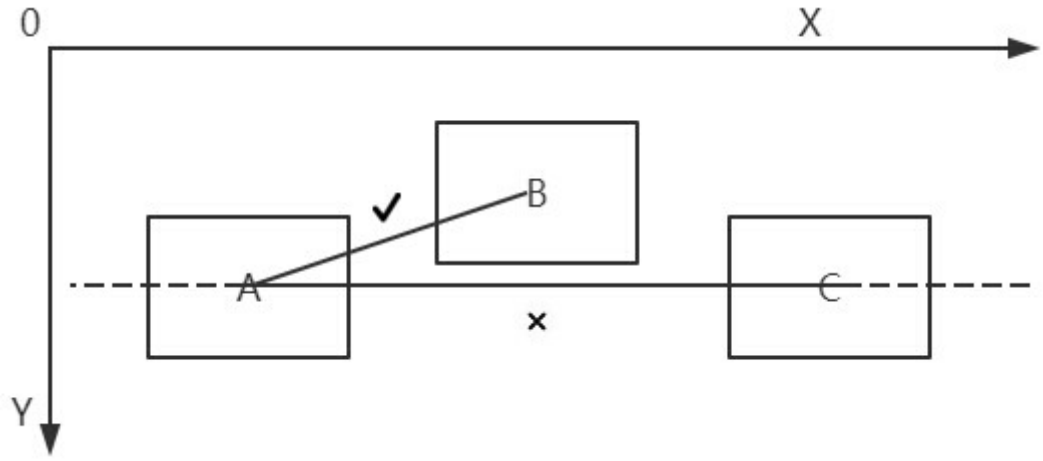


A向右移动后选B或选C都合理。

- **中点连线夹角最小原则**：即计算两个光标中点连线的水平夹角，越小则表示两个光标同处水平方向越大。

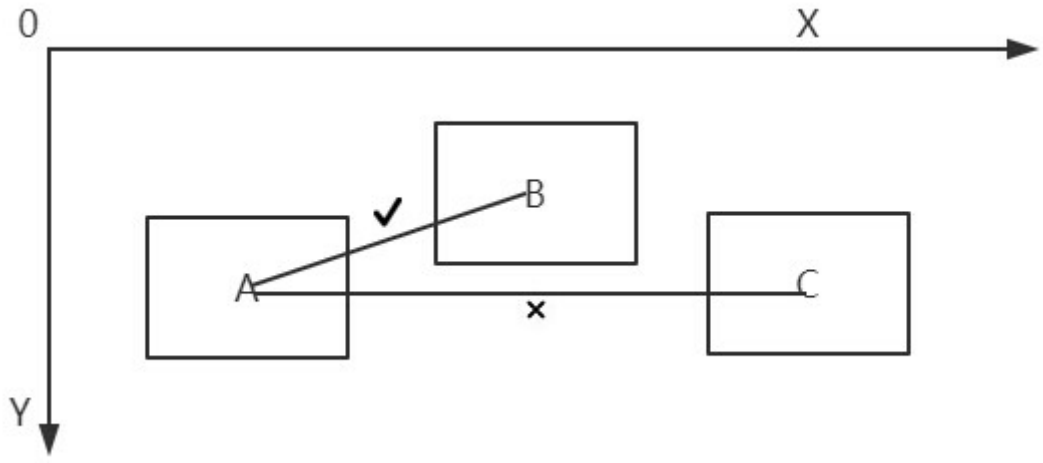


因为A与C的中点连线夹角几乎为0，所以最佳光标为C，但是当出现这种布局时该原则不成立。

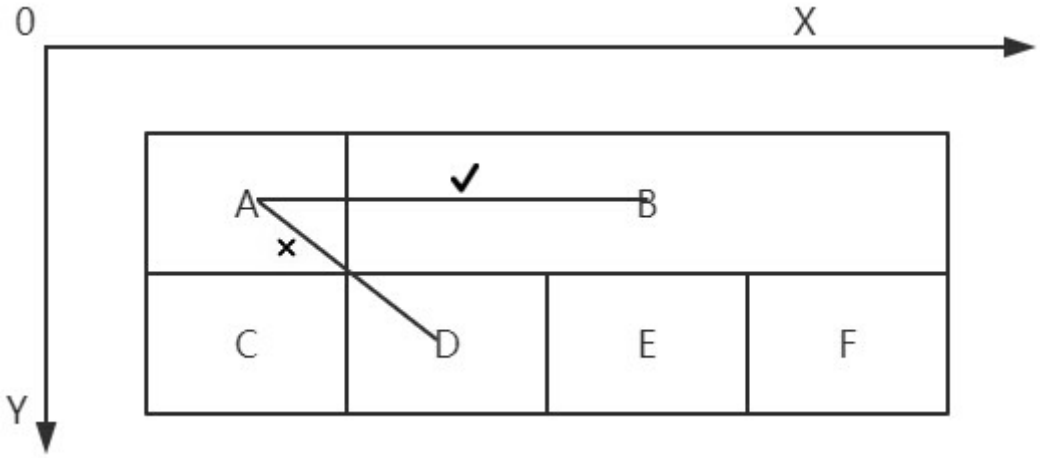


虽然A与C夹角几乎为零，但是由于B挡在中间，所以A向右移动时最佳的光标应为B。

- **中点连线最短原则**：即计算两个光标的中点连线大小，最短的就是最佳的光标。



AB中点连线长度比AC中点连线长度小，所以最佳光标为B，但是出现下面这种布局则该原则不成立。

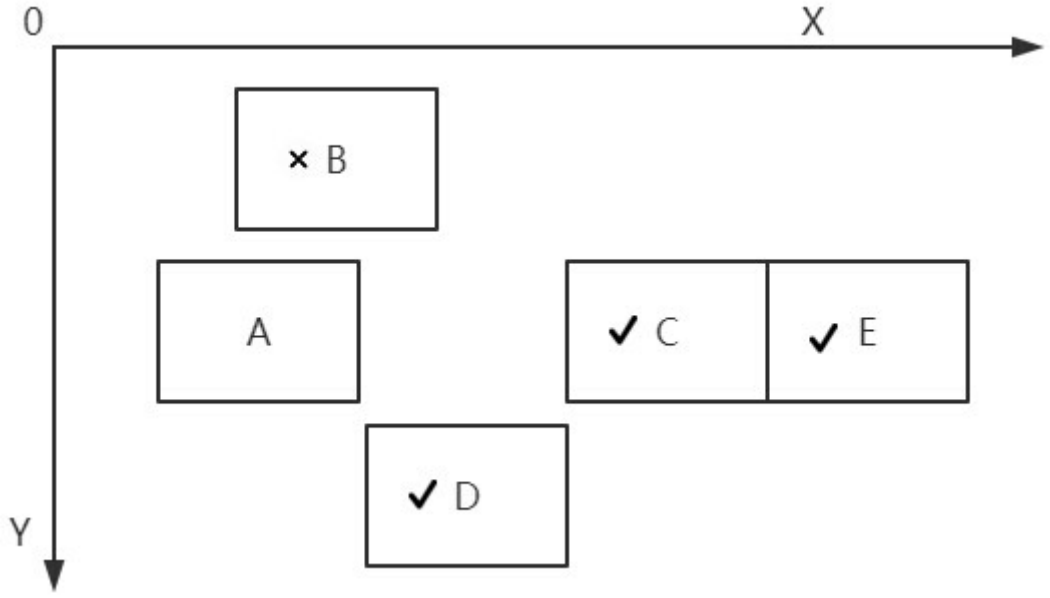


虽然AD中点连线长度比AB中点连线长度短，但是最佳光标应为B。

由此可见前面所列的几种算法都有一些局限性，无法最大化满足所有需求，要做到最大化满足所有需求单单使用一种算法明显是不够的，所以本算法会混合采用两种算法以求达到最大化的目的。

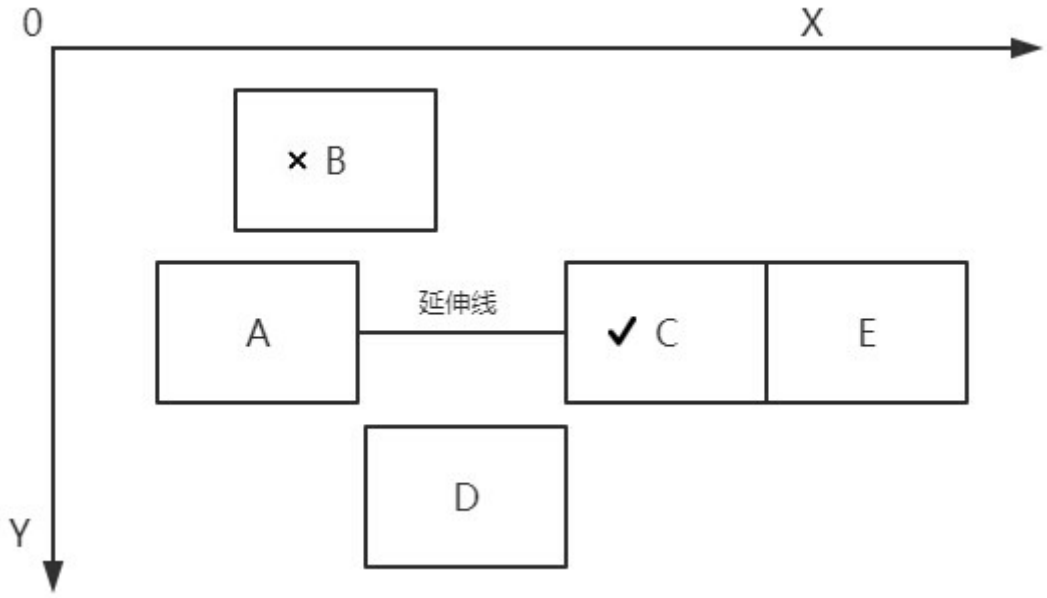
布入正题

1. 首先定义候选光标条件，以向右移动为例，假设当前光标为A，其它光标为B，则满足B.centerX>A.right的都为候选光标。

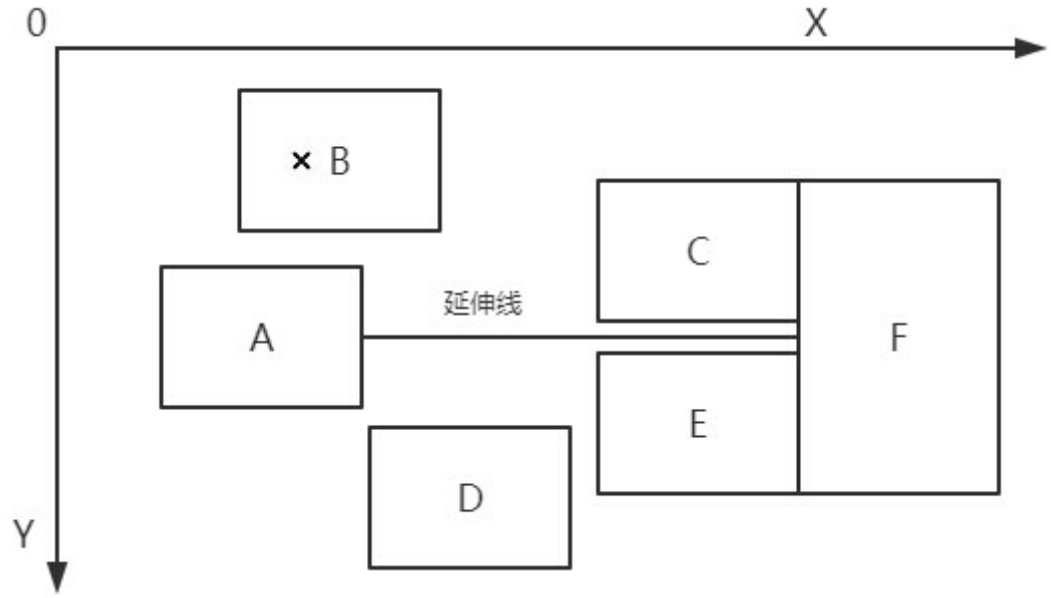


候选光标：D、C、E

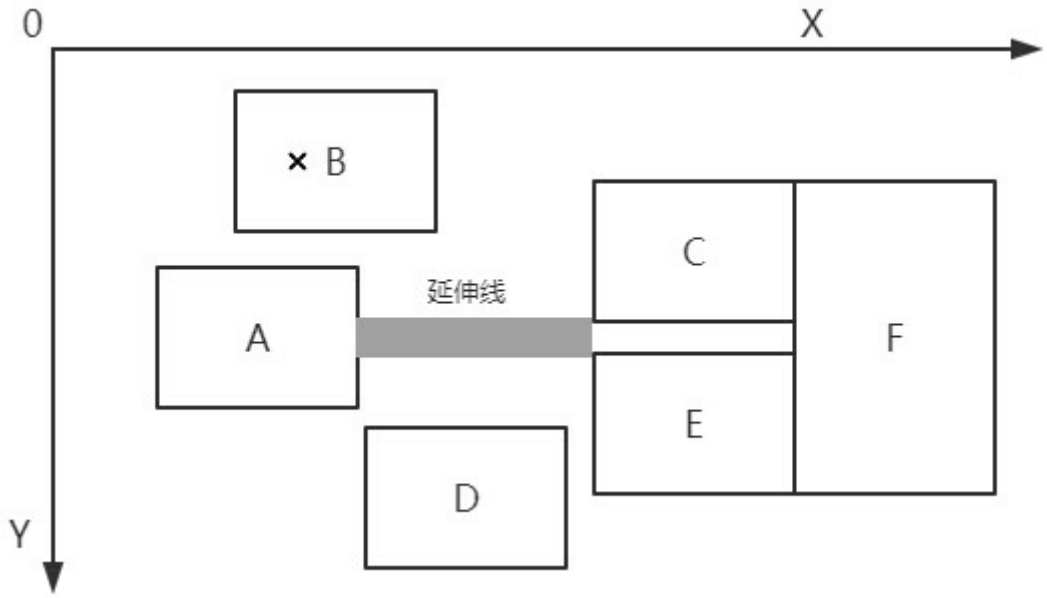
1. 将候选光标放入数组buttons，假设其它光标的左边线与当前光标的右边线的距离为distance，distance=Math.abs(a.right-b.left);
2. 对数组buttons，按distance由小到大排序，即距离越近的排在前面，此时buttons=[D,C,E]。
3. 从A.centerY处向右延伸出一条直线，看看与谁最早产生交集，若有交集则找到最佳光标。



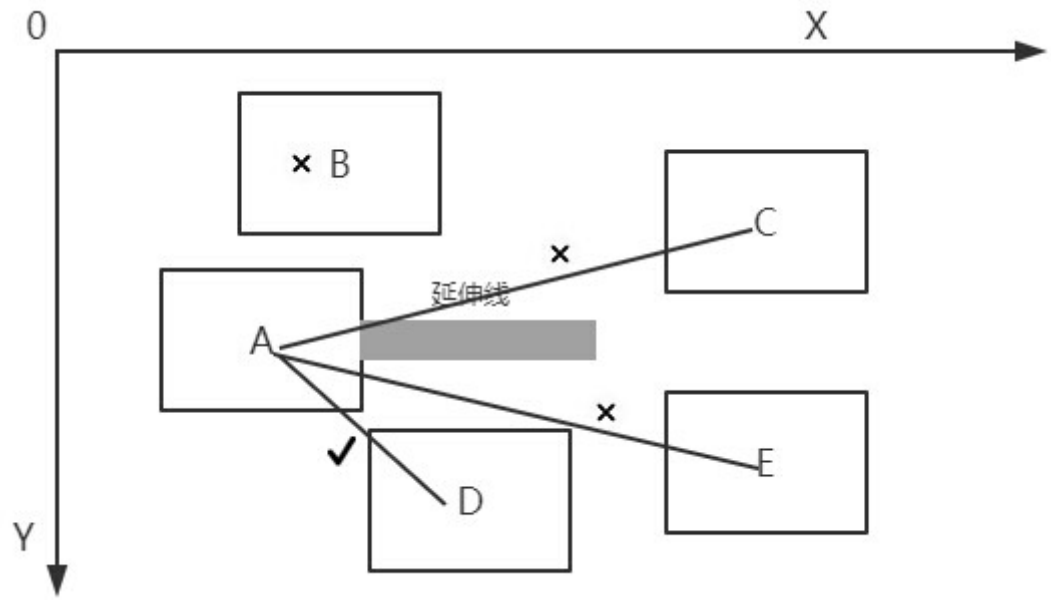
此时C为最佳光标，结束查找，但是也有例外情况，如下图：



由于延伸线太细，而CE之间又有缝隙，导致F成了最佳光标，这样明显不合理，所以需要对延伸线做加粗处理，可以设置一个比例因子ratio，ratio=A.height*0.25（或0.33等）



1. 如果通过延伸线没有找到产生交集的光标，则继续采用中点连线最短算法，计算其它光标与当前光标的中点连线长度，pointDistance=Math.sqrt(Math.pow(a.centerX-b.centerX,2)+Math.pow(a.centerY-b.centerY,2))



1. 对buttons按pointDistance由小到大排序，取第一个元素作为最佳光标。

以上就是本算法，**本算法不是完美的，只是力求做到最大化。**初次发文，小心拍砖，欢迎指正补充！

参考实现：http://172.16.4.213:8080/page/preview?path=/ott-blkg-all/column/home.json