

• 接口形式

python层面:

```
def nms_rotated(boxes, scores, threshold):  
    """  
    NMS for obbs  
  
    Args:  
        boxes (np.ndarray): (N, 5), xywhr.  
        scores (np.ndarray): (N, ).  
        threshold (float): IoU threshold.  
  
    Returns:  
        valid_id(list)  
    """
```

c++层面:

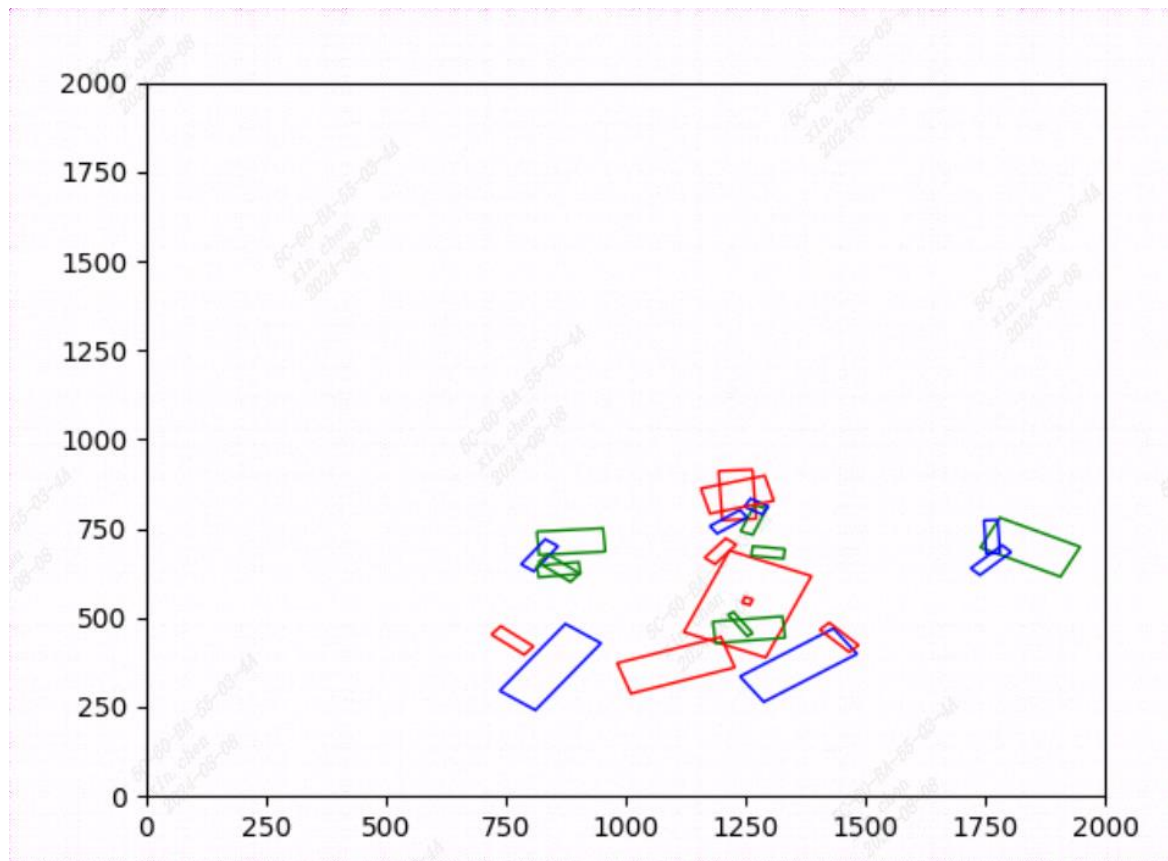
```
std::vector<int> nms_rotated(std::vector<std::vector<float>>, std::vector<float>, float)
```

Part1 旋转nms算法

Part2 问题与优化方案

旋转nms算法

- 是啥?
 - 框不是水平的情况下做nms

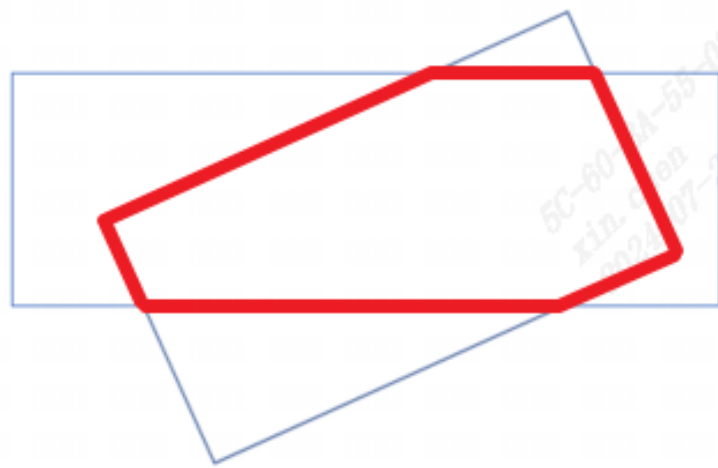


旋转nms算法

- 基本步骤：
 1. 按照分数将框排序
 2. 按照上述顺序，取一个框A作为基准框，保留A的id。用A与其他框B算IOU，如果超过阈值就扔掉B。（相当于直接删除B）
 3. 重复2，直到遍历所有框，返回所有保留框的id。
- 注：
 - 实际实现中只根据分数排序索引就可以，根据排好序的索引取对应框取操作。

旋转nms算法

- 计算IOU:
- 1. 求矩形顶点坐标
- 2. 求公共多边形的顶点坐标
- 3. 将多边形顶点排序与过滤
- 4. 求多边形面积
- 5. 得到IOU



旋转nms算法

- 1. 求矩形顶点坐标: $[x_ctr, y_ctr, w, h, \theta] \rightarrow 4*(x, y)$

- 未旋转时, 以原点为矩形中心的四个顶点是:

$$\left(\frac{w}{2}, \frac{h}{2}\right), \left(-\frac{w}{2}, \frac{h}{2}\right), \left(-\frac{w}{2}, -\frac{h}{2}\right), \left(\frac{w}{2}, -\frac{h}{2}\right)$$

- 旋转矩阵是:

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

旋转矩形的推导可以参考:

<https://blog.csdn.net/csxiaoshui/article/details/65446125>

- 旋转后的坐标:

$$\begin{pmatrix} x'_i \\ y'_i \end{pmatrix} = R \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_i \cos(\theta) - y_i \sin(\theta) \\ x_i \sin(\theta) + y_i \cos(\theta) \end{pmatrix}$$

旋转nms算法

- 1. 求矩形顶点坐标: $[x_ctr, y_ctr, w, h, \theta] \rightarrow 4*(x, y)$
 - 带入顶点坐标, 以右上顶点为例:

$$\left(\frac{w}{2}, \frac{h}{2} \right)$$



$$\begin{aligned} x'_1 &= \frac{w}{2} \cos(\theta) - \frac{h}{2} \sin(\theta) \\ y'_1 &= \frac{w}{2} \sin(\theta) + \frac{h}{2} \cos(\theta) \end{aligned}$$

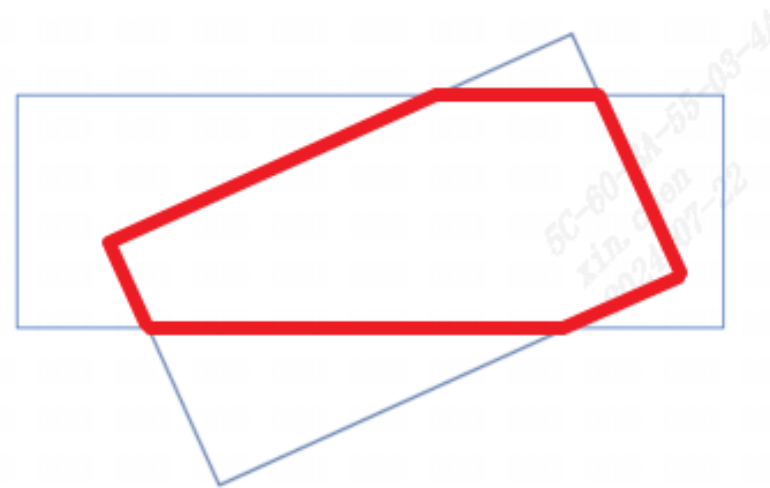
- 加上中心点坐标 (原点偏移量), 得到所有公式

$$\begin{aligned} x''_i &= x + x'_i \\ y''_i &= y + y'_i \end{aligned}$$

注: 对称点的坐标公式可以由中心点坐标求得
 $2*(x_ctr, y_ctr) = (x, y) + (x', y')$

旋转nms算法

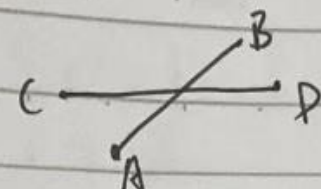
- 2. 求公共多边形的顶点坐标
 - 顶点类型:
 - 交点 (两边交点)
 - 原矩形顶点 (顶点在另一个矩形内部)



旋转nms算法

- 2. 求公共多边形的顶点坐标
 - 交点（两边交点）

Date

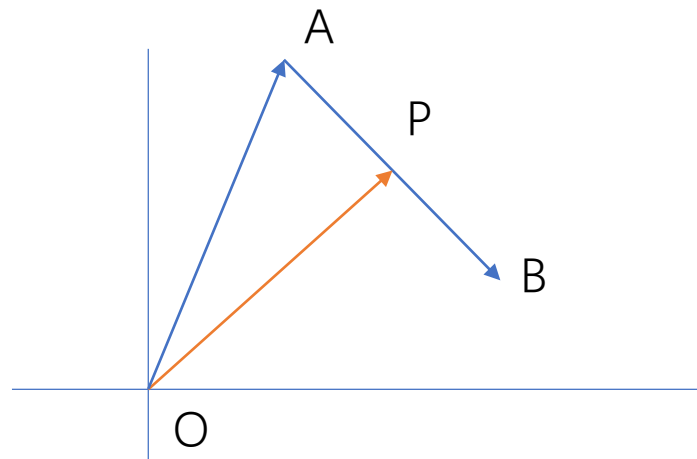


设 $d_1 = \vec{AB} = B - A$
 $d_2 = \vec{CD} = D - C$ $\Rightarrow \text{cross}(d_1, d_2) \neq 0$

线段AB上的点可以表示为:
 $P_1(t_1) = A + t_1 \cdot d_1 \quad (t \in [0, 1])$

线段CD上的点可以表示为:
 $P_2(t_2) = C + t_2 \cdot d_2 \quad (t \in [0, 1])$

交点:
 $A + t_1 \cdot d_1 = C + t_2 \cdot d_2$
 $t_1 \cdot d_1 - t_2 \cdot d_2 = C - A$



求 t_1 : 两边叉乘 d_1 或 d_2 ($d_2 \times d_2 = 0$)

$$t_1 \cdot \text{cross}(d_1, d_2) = \text{cross}(C - A, d_2)$$
$$t_1 = \frac{\text{cross}(C - A, d_2)}{\text{cross}(d_1, d_2)} \in [0, 1]$$

同理 $t_2 = \frac{\text{cross}(C - A, d_1)}{\text{cross}(d_1, d_2)} \in [0, 1]$

如果满足 $0 \leq t_1 \leq 1$ && $0 \leq t_2 \leq 1$, 则有交点:

$$P = A + t_1 \cdot d_1$$

旋转nms算法

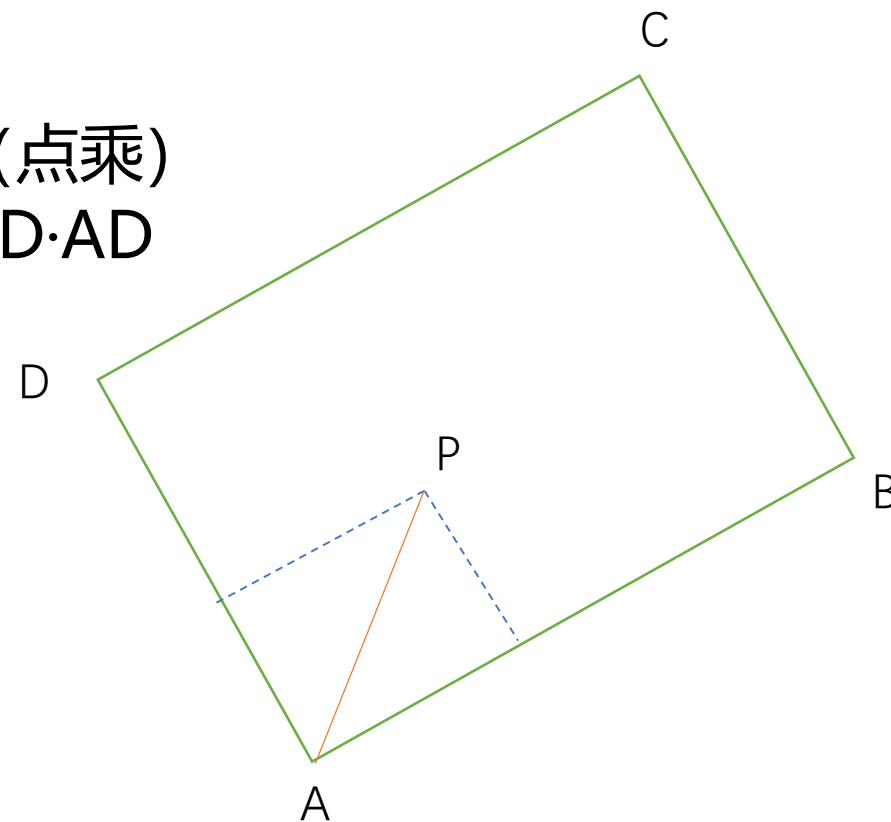
- 2. 求公共多边形的顶点坐标

- 原矩形顶点（顶点在另一个矩形内部）

- 判断方式：

- 顶点在矩形两边上的投影是否在两边的范围内（点乘）

- $0 \leq \mathbf{AP} \cdot \mathbf{AB} \leq \mathbf{AB} \cdot \mathbf{AB} \ \&\& \ 0 \leq \mathbf{AP} \cdot \mathbf{AD} \leq \mathbf{AD} \cdot \mathbf{AD}$



- 2. 求公共多边形的顶点坐标
 - 整体流程：
 - 交点：两个矩形各取一条边求交点（嵌套for循环）： $4*4=16$ 种
 - 原矩形顶点：两个矩形各个顶点都有在另一个矩形内的可能：8种
 - 最终得到交点坐标数组intersection[24]，和有效点个数num

旋转nms算法

- 3.将多边形顶点排序与过滤
 - Graham Scan
 - 求包含点集的最小凸边型

注:

- 2. Sort: 叉乘 $a \times b > 0$, 则向量 a 的方向在向量 b 的逆时针方向, 排序向量 $p_0 p_i$
- 2. Farthest: 通过点乘 $p_i \cdot p_i$ 确定
- 8. Next-to-top: 栈顶的第二个元素
- 8. Nonleft turn: 右拐或直行

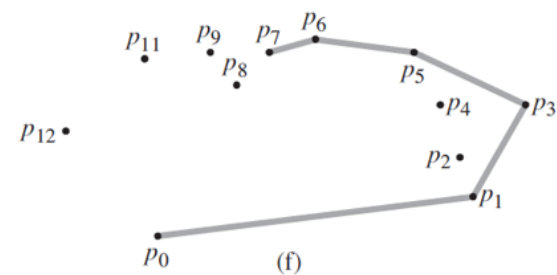
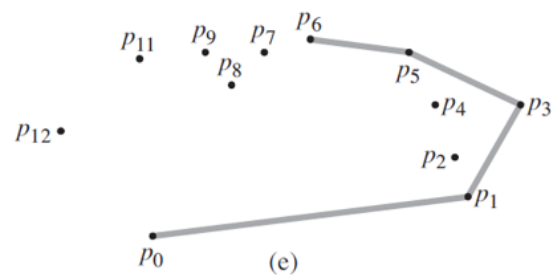
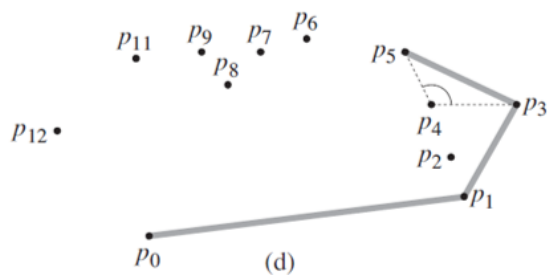
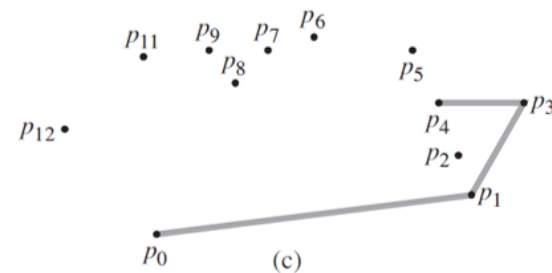
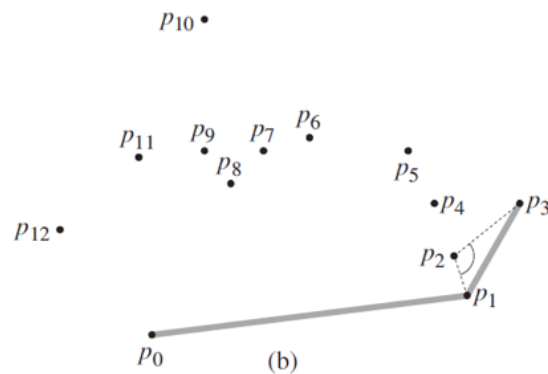
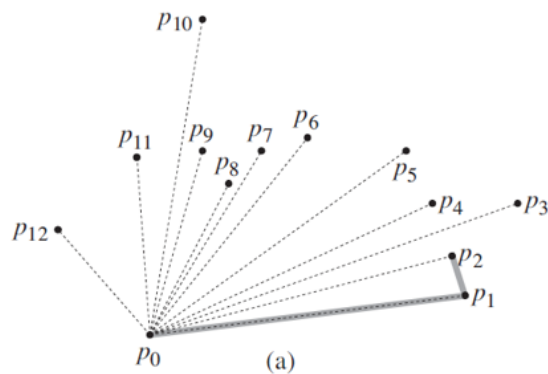
GRAHAM-SCAN(Q)

- 1 let p_0 be the point in Q with the minimum y -coordinate, or the leftmost such point in case of a tie
- 2 let $\langle p_1, p_2, \dots, p_m \rangle$ be the remaining points in Q , sorted by polar angle in counterclockwise order around p_0 (if more than one point has the same angle, remove all but the one that is farthest from p_0)
- 3 let S be an empty stack
- 4 PUSH(p_0, S)
- 5 PUSH(p_1, S)
- 6 PUSH(p_2, S)
- 7 **for** $i = 3$ **to** m
- 8 **while** the angle formed by points NEXT-TO-TOP(S), TOP(S), and p_i makes a nonleft turn
- 9 POP(S)
- 10 PUSH(p_i, S)
- 11 **return** S

https://blog.csdn.net/weixin_38442390

旋转nms算法

• 3.将多边形顶点排序与过滤

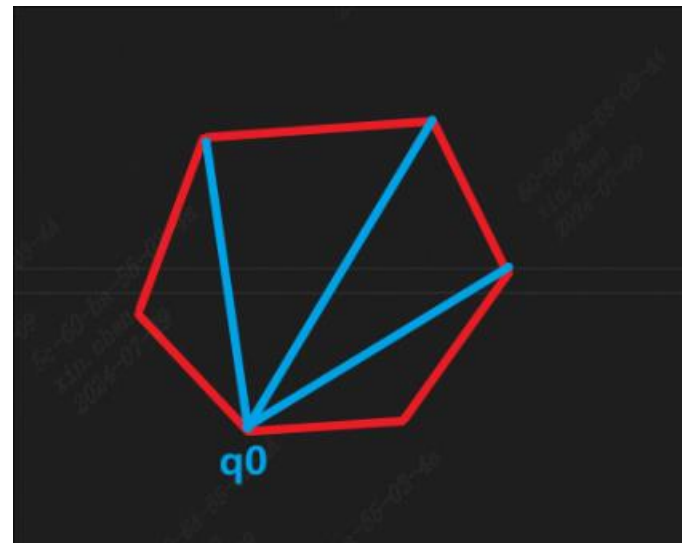


旋转nms算法

- 4. 求多边形面积
 - 逆时针向量叉乘求面积

注：

两向量叉乘结果是由两向量组成的平行四边形面积



旋转nms算法

- 5. 求IOU
 - $\text{Intersection} / (\text{area1} + \text{area2} - \text{intersection})$

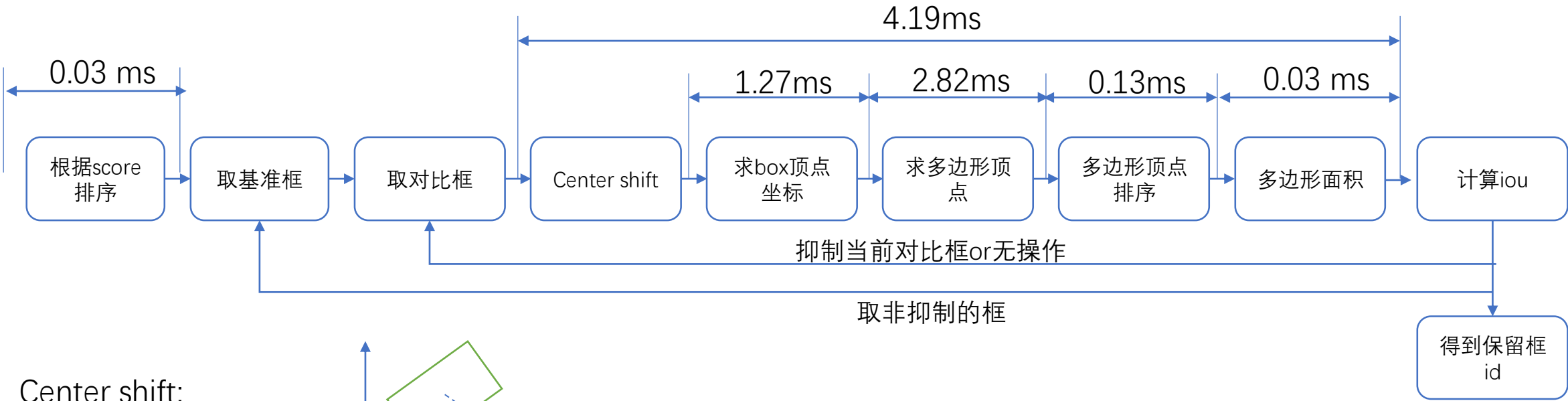
Part1 旋转nms算法

Part2 问题与优化方案

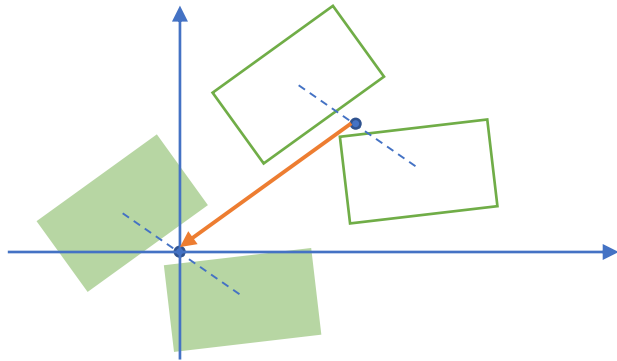
问题与优化方案

- 问题1：接口能否再提速
- 问题2：ground truth有误

问题与优化方案



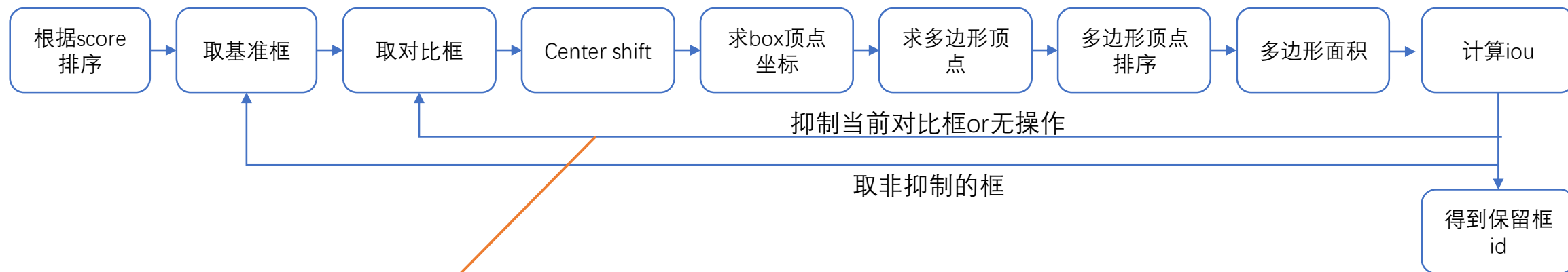
Center shift:
是一对一的关系



问题与优化方案

- 问题1：接口能否再提速

- 想法1：线程池



拆分子任务，并行执行
对同一基准框，不同对比框的结果不相互影响

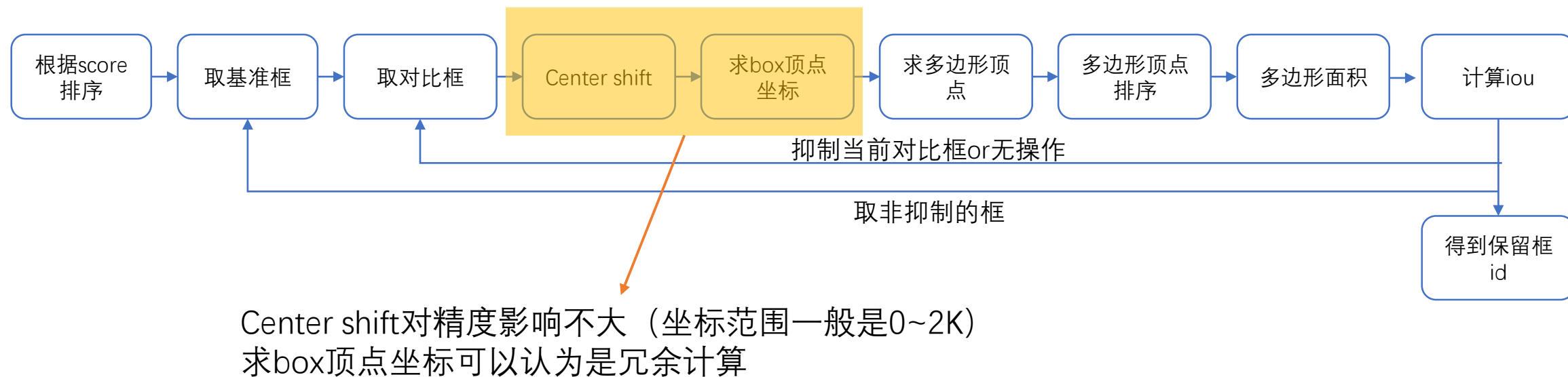
实验结果：
效果不好，速度更慢。每个线程负责
的子任务数量可能较少，导致并行优
势不足以掩盖线程开销的劣势

问题与优化方案

- 问题1：接口能否再提速
 - 想法2：语句优化
 - 减少除法等耗时语句
 - 使用neon加速（批量数据处理，但逻辑复杂，未尝试）
- Neon开发时间较长，未尝试
- 语句优化已包含在O3，时间没有太大区别

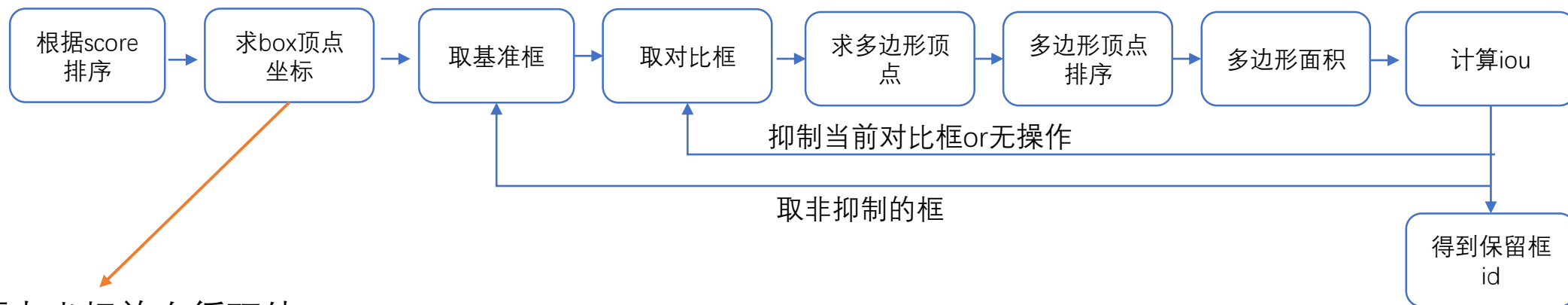
问题与优化方案

- 问题1：接口能否再提速
 - 想法3：算法调整



问题与优化方案

- 问题1：接口能否再提速
 - 想法3：算法调整



求顶点坐标放在循环外

最坏情况： $O(n^2) \rightarrow O(n)$, n 为框的个数

$O(n^2)$: 每次都没有需要抑制的框, 即 $((n-1)+2)(n-1)/2$

实验结果:
总时长2.8ms

问题与优化方案

- 问题2: ground truth有误

- 现象:

- 上述优化后, 得到的框数量比gt少一个。
 - 打印所有gt后发现有一个框和第一个框只有宽有一点点不一样, gt存在错误
 - (198为基准框, 235错误保留)

```
198: [ 7.413525e+04 7.428150e+04 1.082500e+02 8.018750e+01 -1.204277e+00]
201: [ 7.413525e+04 7.428150e+04 1.082500e+02 8.050000e+01 -1.204277e+00]
235: [ 7.413525e+04 7.428150e+04 1.088750e+02 8.018750e+01 -1.204277e+00]
303: [ 7.413525e+04 7.428150e+04 1.085625e+02 8.006250e+01 -1.204277e+00]
357: [ 7.413525e+04 7.428150e+04 1.080000e+02 8.000000e+01 -1.204277e+00]
```

问题与优化方案

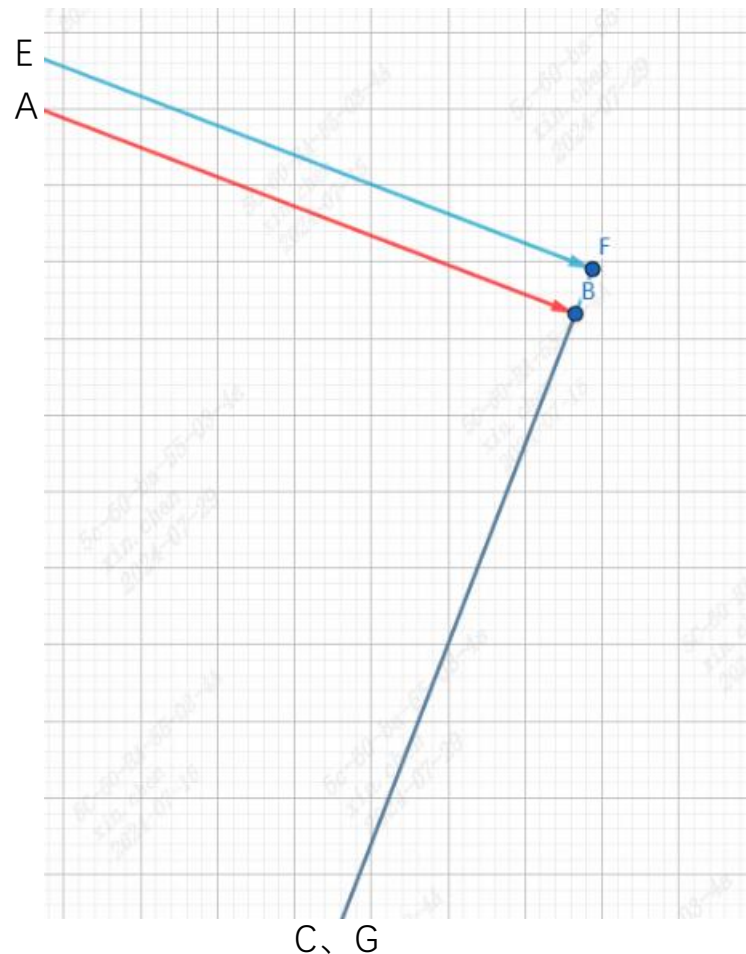
- 问题2: ground truth有误

- 原因:

- 客户给的数据集不是真实的
 - 中心点是0~100000的随机分布
 - 固定中心点随机修改宽高或角度

- 框高重合度, 可表示的精度不足
算多边形顶点时错误扔掉多个交点, 例如:

- $BC=108.25$
 - $FG=108.875$
 - $BF=0.3125$
 - 交点是B, 对于AB向量, 算得 $t_1=1.0000002$, 舍弃
 - 当交点个数小于2时, 认为没有相交多边形, $iou=0$
 - 实际上 $iou>0.994$



问题与优化方案

- 问题2: ground truth有误
 - 总结:
 - 算法实现时不够鲁棒
 - 对 t_1, t_2 的范围判断时可以添加极小值 (类比判断是否平行)
 - 使用更高精度处理 (float- \rightarrow double) , 略微影响速度 (0.3ms)