

The background is a complex, layered illustration. At the top, there are concentric circles and a grid pattern, resembling a technical or astronomical diagram. A large, dark magnifying glass is positioned over the center, focusing on a sunset scene. The sunset features a silhouette of a person standing on a path, with stylized trees and foliage in the foreground. The sky is a mix of warm colors (orange, red, yellow) and cool colors (blue, purple). Various geometric shapes like triangles, circles, and lines are scattered throughout the composition, adding to the abstract feel.

Computational Geometry

From Foundation To Giving Up

crazy_cloud @ SJTU

qq: 1607548255

E-mail: a_crazy_czy@163.com

Content

- *Basic Structure & Operation*
- *Basic Algorithm*
 - *Convex Hull*
 - *Scan Line Algorithm*
 - *Pick's Theorem*
- *Advanced Algorithm*
 - *Halfplane Intersection*
 - *Minkowski Sum*
 - *Nearest Neighbor*
 - *Basic 3D Geometry*
 - *Numerical Algorithm*
 - *Randomized Algorithm*
 - *Dual*
- *Complex Algorithm*
 - *3D Convex Hull*
 - *Voronoi Diagram*
 - *Delaunay Triangulation*

A large satellite dish antenna is the central focus, mounted on a complex metal structure. It is set against a dramatic sunset sky with warm orange and yellow hues. In the foreground, a person stands on a rocky outcrop, looking at a tablet. The background features a mountainous landscape with evergreen trees and a small building with a chimney. The overall scene is a blend of technology and nature.

Basic Structure & Operation

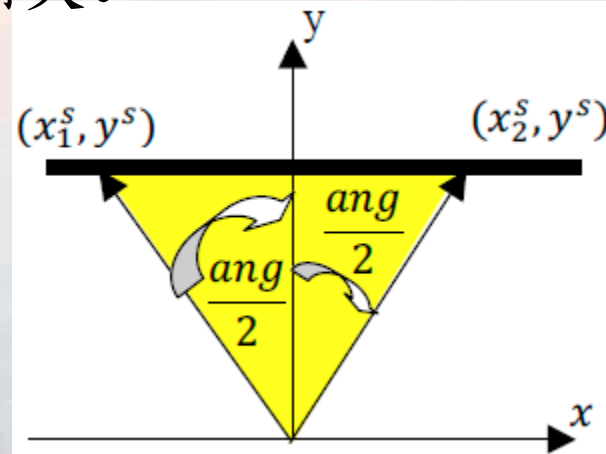
让我们先从一道基础计算几何题开始。

投影面积

JSTSC2015 Round 2 Day 2, light

- 在二维平面上，有一个位于原点的光源，可以朝 y 轴正方向发出对称于 y 轴的张角为 ang 的光束。
 - 投影屏是连接 (x_1^s, y^s) 和 (x_2^s, y^s) 的一条线段。
 - 一共有 n 个以平面上线段形式存在的障碍物，保证其 y 值严格小于 y^s 。
 - 所有障碍物之间互相不接触，也不会和投影屏以及光源接触。
 - 这些障碍物有反光和不反光两种。当光线射到反光的障碍物上时，会按照光学原理反射，否则会被完全吸收。反光的障碍物双面都是反光的。投影屏是不反光的。由于散射作用，光线在传播了 len 的长度之后会消失。
 - 现在请你计算投影屏被照亮的区域占投影屏总长度的比例。
- $0 \leq n \leq 10, 0 < ang \leq 150, 0 < len \leq 10^3, |x_i|, |y_i| \leq 10^3$

<https://gmoj.net/senior/#main/show/4065>



Point & Vector

- 点和向量是等价的，可以直接用 (x, y) 表示。
- 向量相关操作有加减、数乘、点乘和叉乘。

```
struct P
{
    db x , y;

    inline P (db x_ = 0. , db y_ = 0.){x = x_ , y = y_;}

    inline P operator + (P const p)const{return P(x + p.x , y + p.y);}
    inline P operator - (P const p)const{return P(x - p.x , y - p.y);}
    inline P operator * (db const k)const{return P(x * k , y * k);}

    inline db operator * (P const p)const{return x * p.x + y * p.y;}
    inline db operator ^ (P const p)const{return x * p.y - y * p.x;}
};
```

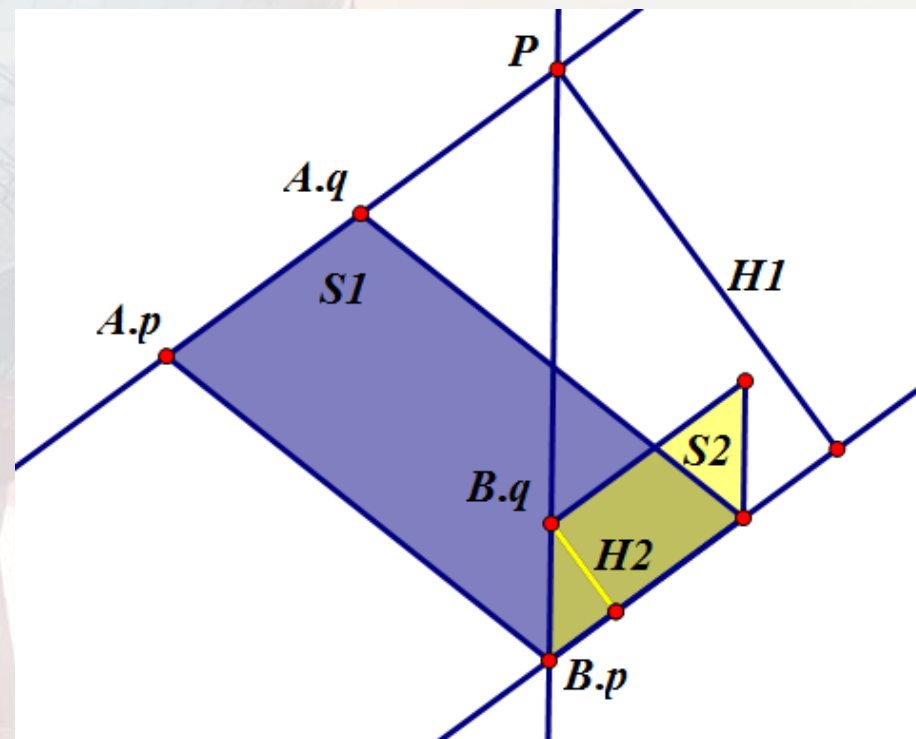

Segment & Half-line & Line

- 不论是线段/射线/直线都可以写成向量式 $q = p + \lambda v$, 所以可以用 (p, v) 表示。
- 当 (p, q) 表示的是线段时, $0 \leq \lambda \leq 1$;
- 当 (p, q) 表示的是射线时, $\lambda \geq 0$;
- 当 (p, q) 表示的是直线是, $\lambda \in R$ 。

```
struct L
{
    P p , v;

    inline L (){}
    inline L (P p_ , P v_){p = p_ , v = v_;}
};
```

- 用向量式表示的直线如何求交?
- 利用相似三角形的比例关系!
- 注意特判两条直线平行的情况。



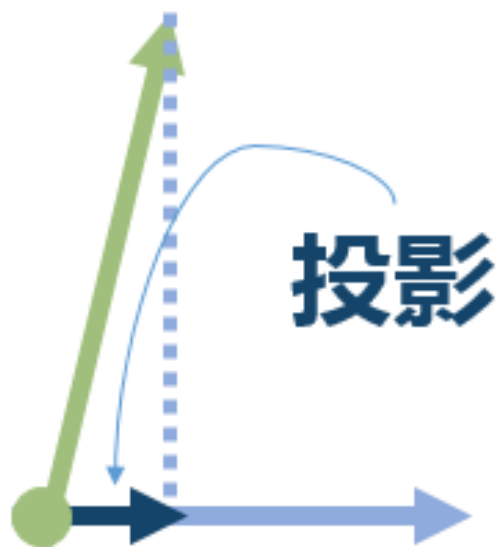
```
inline P ict(L a , L b){return b.p + b.v * ((a.v ^ (a.p - b.p)) / (a.v ^ b.v));}
```

Intersection Of Lines

Intersection Of Segments

- 先做一次直线求交，求出交点。
- 然后在判断这个交点是否同时在两条线段上。
- 判断点 P 是否在线段 (p, q) 上？
- 只需要判断 $|\overrightarrow{Pp}| + |\overrightarrow{Pq}| = |\overrightarrow{pq}|$ 是否成立即可。
- 当然，还有一种不需求出交点的快速排斥实验 + 跨立实验法，相对而言较为麻烦，在这里不作介绍。

Projection

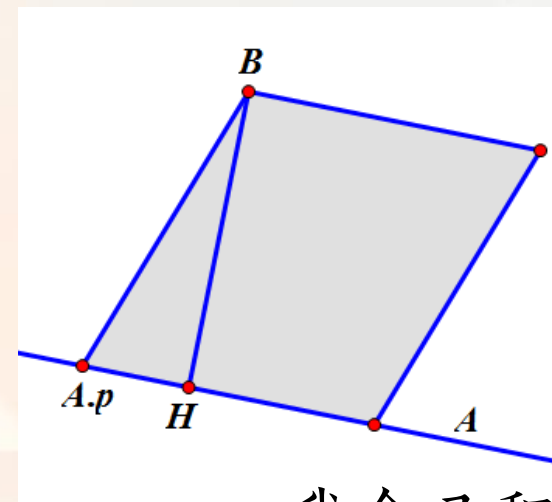


点积 = 投影 * 原长

```
inline db shade(P p , P q){return (p * q) / mod(q);}
```

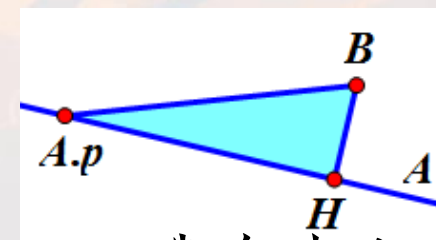
Distance

Point vs Line



我会叉积!

```
inline db dist(P p , L l){return fabs((p - l.p) ^ l.v) / mod(l.v);}
```



我会点积!

```
inline db dist(P p , L l){return sqrt(mod2(p - l.p) - sqr(shade(p - l.p) , l.v));}
```

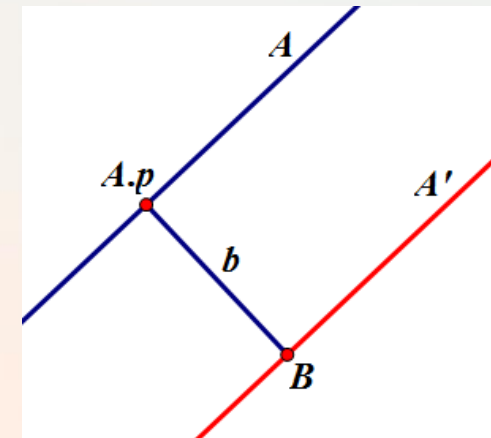
Perpendicular

- 选取右手方向作垂直向量。

```
inline P perpendicular(P p){return P(p.y , -p.x);}
```

Translation

- 将向量沿其右手方向平移。



Rotation

- 将向量逆时针旋转一定的角度。

```
inline P rotate(P p , db alpha)
{
    db cos_ = cos(alpha) , sin_ = sin(alpha);
    return P(p.x * cos_ - p.y * sin_ , p.y * cos_ + p.x * sin_);
}
```

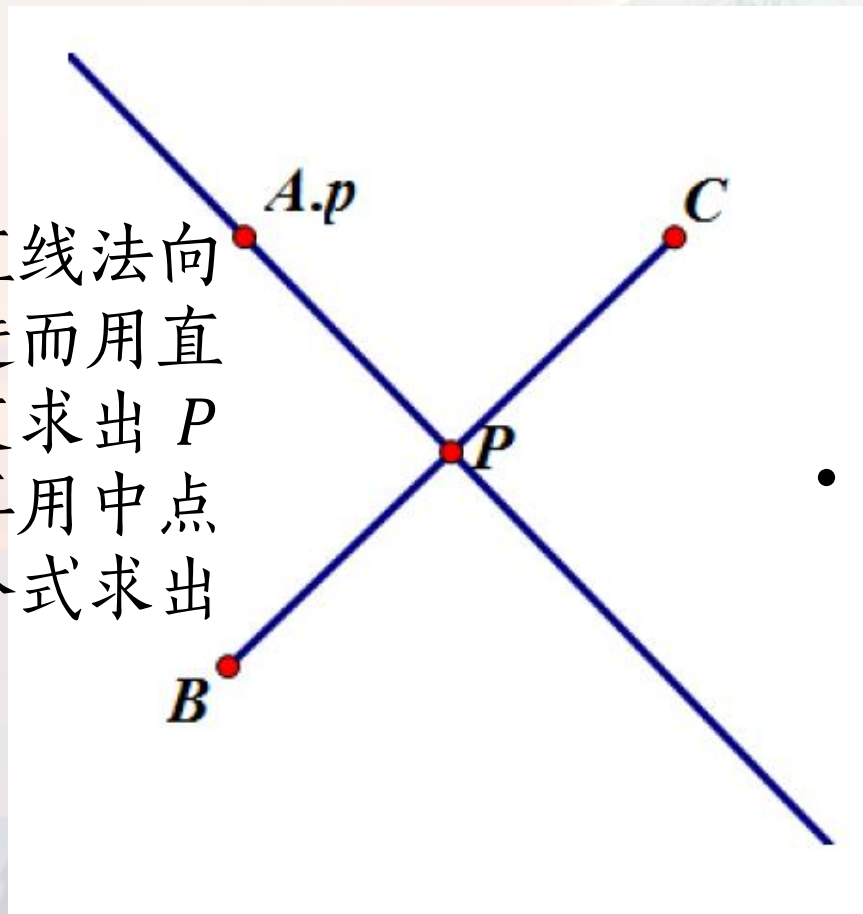
Adjustment

- 调整向量的模长。

```
inline P adjust(P p , db l){return p * (1 / mod(p));}
```

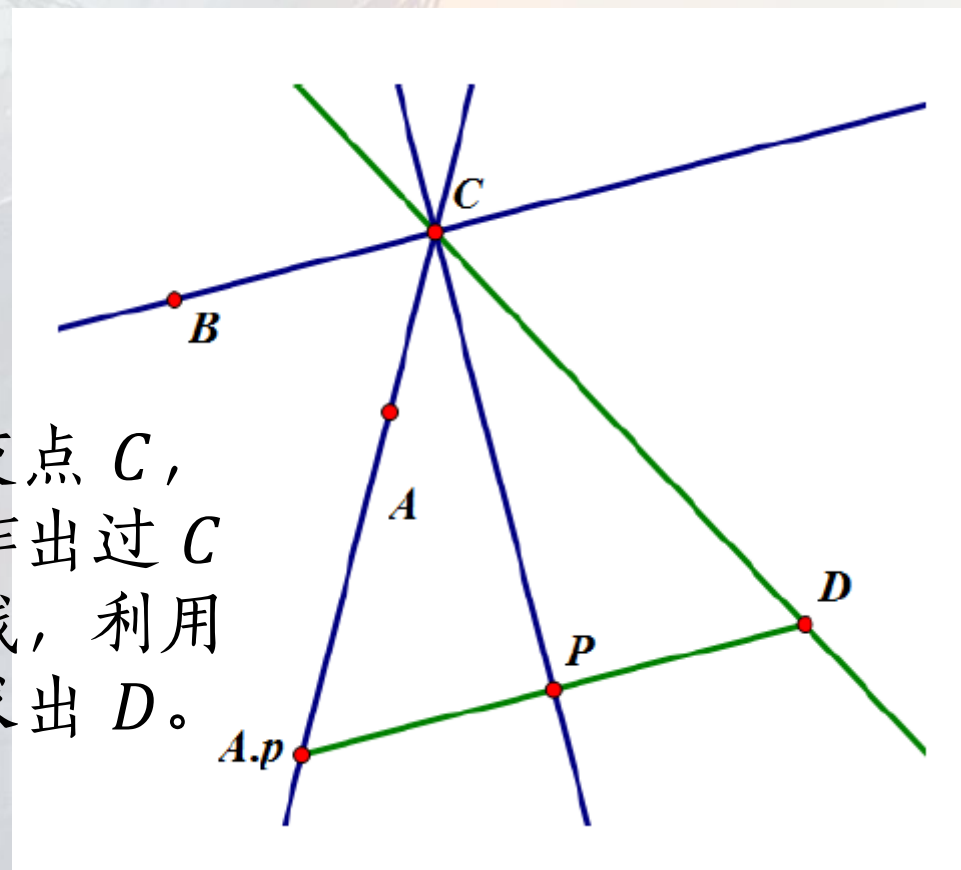
Mirror

- 作出直线法向量，进而用直线求交求出 P 点，再用中点坐标公式求出 C 点。



Reflection

- 计算交点 C ，进而作出过 C 的法线，利用镜像求出 D 。



So What?

- 什么？你说我讲了这么多，开始那道题还是不会？



- *Hint:* 爆精度模拟。



Halfplane

- 考虑利用直线的向量表达，直接强制规定向量右手方向一侧为该半平面即可。

Polygon

- 按照特定时针方向将凸包上的点排列出来即可。
- 凸多边形的面积：任选一个参照点对所有相邻点作叉积求代数和。
- 点在凸多边形内：射线法。可以对退化的相交情况作详细规定来避免共线导致的错误。

Circle

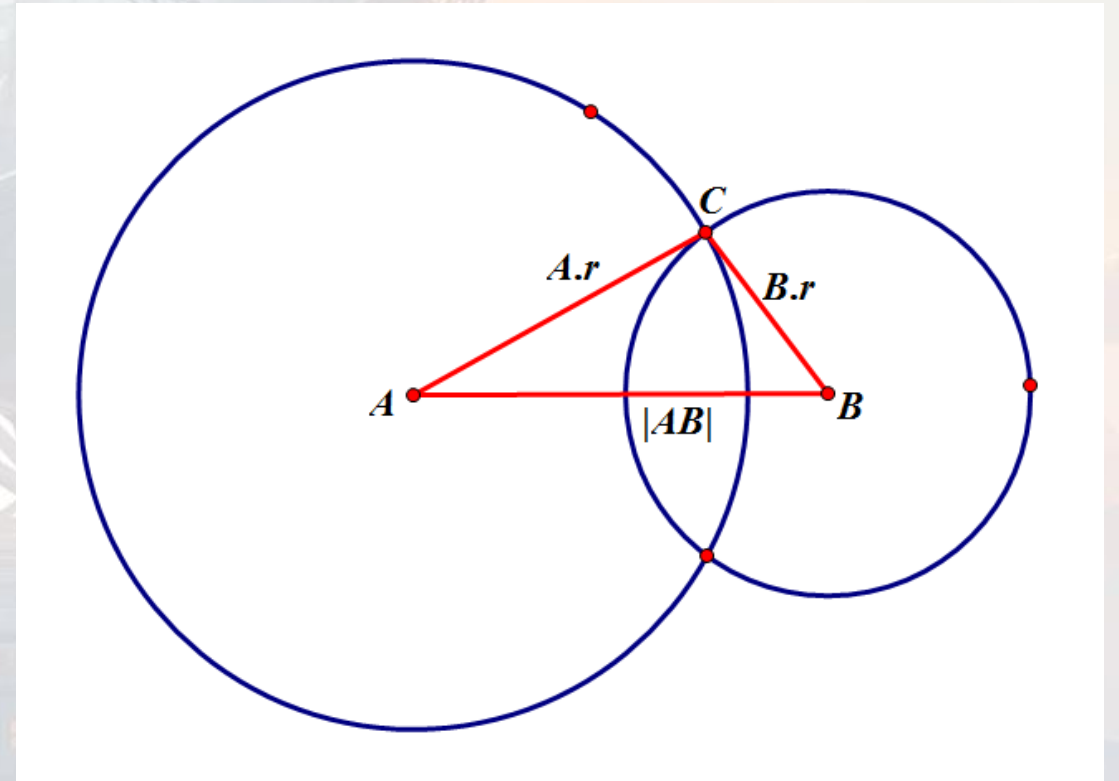
- 定义：圆心和半径即可。
- 直线与圆的关系：利用圆心与直线距离和圆半径判断。
- 过点作圆切线：勾股定理、向量旋转和向量调整。

```
struct C
{
    P O;
    db r;

    inline C (){}
    inline C (P O_ , db r_ = 0){O = O_ , r = r_;}
};
```


Intersection Of Circles

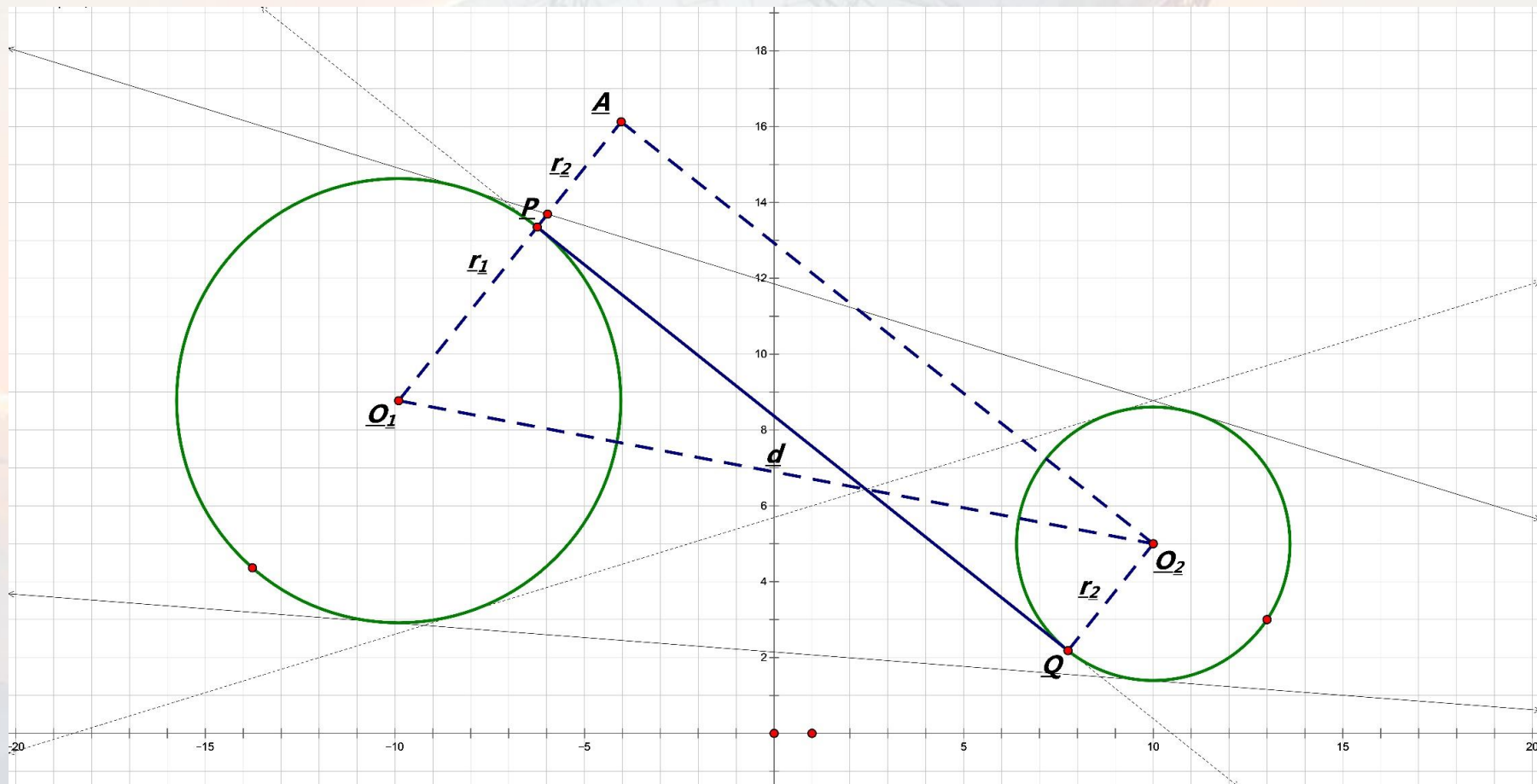
- 先判断圆与圆的位置关系。
- 然后就是余弦定理、向量旋转和向量调整解决的问题了。



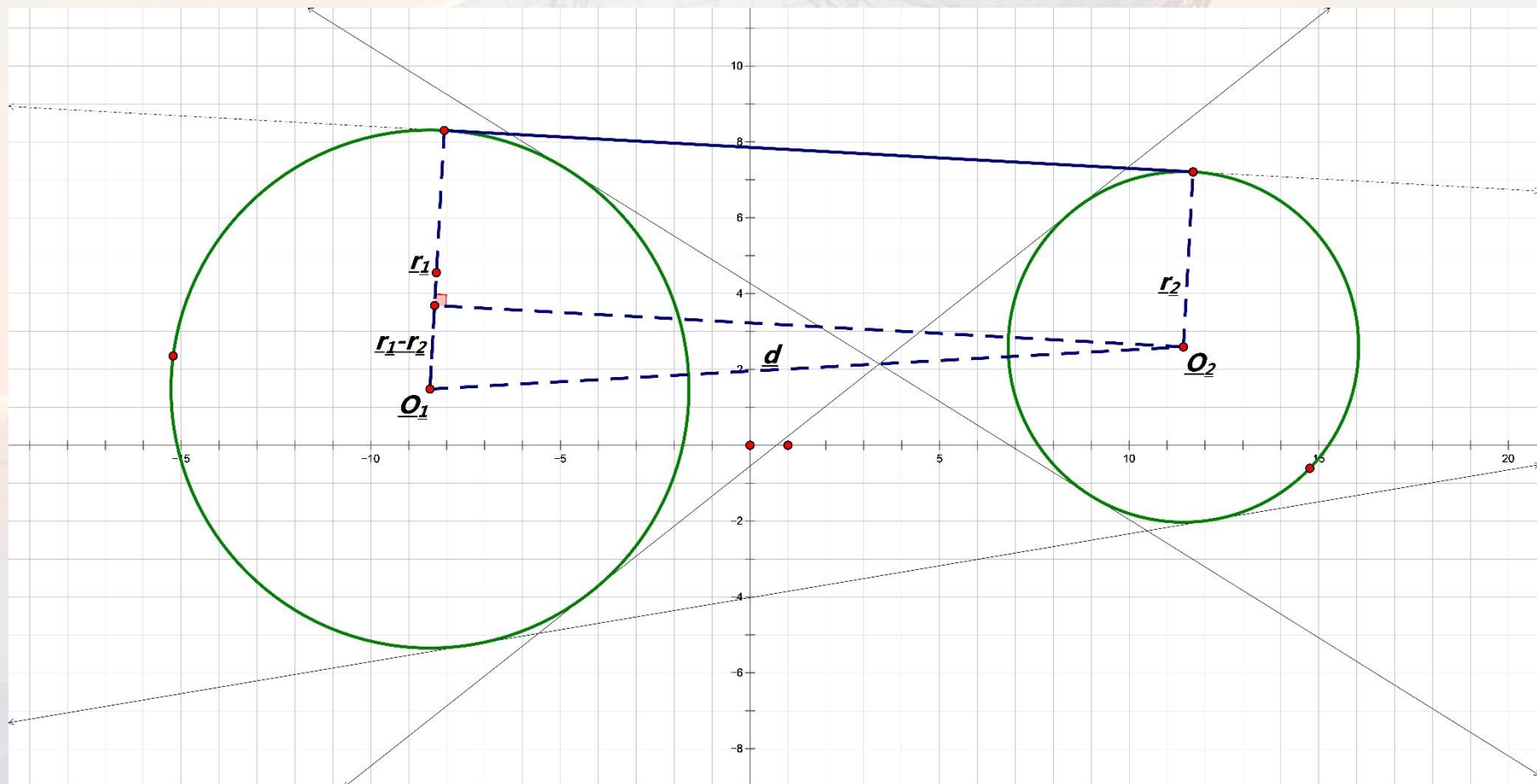
Common Tangents Of Circles

- 两圆完全重合，无数条公切线
- 两圆内含，没有公切点。没有公切线。
- 两圆内切。有 1 条外公切线。
- 两圆相交。有 2 条外公切线。
- 两圆外切。有 1 条内公切线，2 条外公切线。
- 两圆相离。有 2 条内公切线，2 条外公切线。

Common Tangents Of Circles



Common Tangents Of Circles



Stochastic Disturbance

- 当你发现你的算法不能很好解决共线之类的细节问题时。
- 你可以试试简单粗暴地将所有点绕着某定点旋转同一个随机的角度，这样就不用考虑什么花里胡哨的东西了。
- 但是这样可能会让原本是整点的问题变为实数点问题，带来精度误差。
- 而且有时候你对坐标系操作了会导致答案不对。
- 所以看着办吧。

A person stands on a rocky outcrop in the foreground, looking out at a massive satellite dish antenna. The dish is mounted on a complex mechanical base and is tilted towards the sky. The scene is set against a dramatic sunset or sunrise sky, with warm orange and yellow light. Several birds are flying in the sky, and a helicopter is visible on the left. In the background, there are dark, silhouetted mountains and a small building with a chimney. The overall mood is contemplative and futuristic.

Basic Algorithm

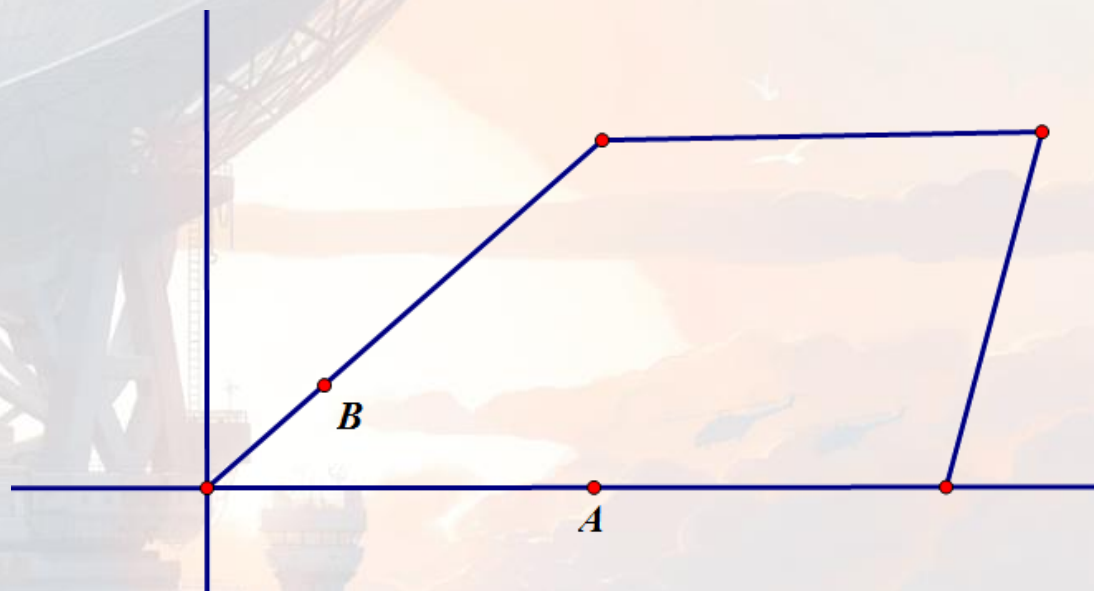
Convex Hull

- *Graham Scan*

- 极角序凸包
- 缺点：无法找出凸包上所有点

- *Andrew Algorithm*

- 水平序凸包+上下凸壳合并



Convex Hull

Andrew Algorithm

```
/*If you want to get all the points on the boundary of the convex hull , please change "<= 0" to "< 0".*/
/*ch.p[1] is equal to ch.p[ch.tot] , actually there are only ch.tot - 1 points.*/
void Convex_Hull()
{
    ch.tot = 0;
    for (int i = 1; i <= n; ++ i) kth[i] = i;
    sort(kth + 1 , kth + 1 + n , cmp1);
    top = 0;
    for (int i = 1; i <= n; ++ i)
    {
        P p = pts[kth[i]];
        for (; top > 1 && ((pts[stk[top]] - pts[stk[top - 1]]) ^ (p - pts[stk[top - 1]])) <= 0; -- top);
        stk[++ top] = kth[i];
    }
    for (int i = 1; i <= top; ++ i) ch.p[++ ch.tot] = pts[stk[i]];
    for (int i = 1; i <= n; ++ i) kth[i] = i;
    sort(kth + 1 , kth + 1 + n , cmp2);
    top = 0;
    for (int i = 1; i <= n; ++ i)
    {
        P p = pts[kth[i]];
        for (; top > 1 && ((pts[stk[top]] - pts[stk[top - 1]]) ^ (p - pts[stk[top - 1]])) <= 0; -- top);
        stk[++ top] = kth[i];
    }
    for (int i = 2; i <= top; ++ i) ch.p[++ ch.tot] = pts[stk[i]];
}
```

Scan Line Algorithm

- 扫描线与事件点是计算几何的关键思想方法。
- 将所有关键的点/坐标提取出来排序，相邻两个元素之间的信息通常会比较好维护。



Scan Line × Angel Order

- 极角排序 × 扫描线可以说是计算几何最常见套路。
- 其对题目的要求就是在一个极角区间内需要维护的信息要么不变，要么是一个很容易维护的函数或者其他形式的信息。
- 如果你是使用 `atan2` 函数来判断极角的话，在第二象限与第三象限的交界位置，函数值是从 π 到 $-\pi$ 的突变，要小心因此造成的精度误差或者是 bug。
- 同时，你也要注意你的程序能够处理相邻两个关键的极角跨度较大（如超过 π ）的情况，有时这个也会让你程序出现一些莫名错误。

Red and blue points

Codechef December Challenge 2017, REDBLUE

- 给定平面上的 n 个红点和 m 个蓝点。
- 你需要删去尽可能少的点（两种颜色都可以删），使得删去后存在一条直线可以将两种颜色的点分开，即直线一侧只有红色的点，另一侧只有蓝色的点。
- $1 \leq n, m \leq 10^3$

Red and blue points

Codechef December Challenge 2017, REDBLUE

- 假设我们已经删除了要删的点，剩下了一些红点和蓝点。
- 那么这条直线一定能够调整到与红凸包或者蓝凸包相切的位置上。
- 考虑需要枚举切点以及直线。枚举一个点做切点，将其它所有点按照它极角排序，然后扫描线，我们规定红蓝点分别在直线的哪个方向，在扫描过程中维护红蓝点个数就好了。
- 时间复杂度 $O(n^2 \log n)$ 。

Pionek

Polish XXV OI(POI2018), etap I.

- 在无限大的二维平面的原点 $(0,0)$ 放置着一个棋子。你有 n 条可用的移动指令，每条指令可以用一个二维整数向量表示。
 - 每条指令最多只能执行一次，但你可以随意更改它们的执行顺序。棋子可以重复经过同一点，两条指令的方向向量也可能相同。
 - 你的目标是让棋子最终离原点的欧几里得距离最远，请问这个最远距离是多少？
- $1 \leq n \leq 2 \times 10^5$

<https://www.lydsy.com/JudgeOnline/problem.php?id=5099>

Pionek

Polish XXV OI(POI2018), etap I.

- 假设我确定了向量的方向，那么我选择所有在该方向上投影为正的向量显然最优。
- 考虑极角排序 × 扫描线。
- 一个极角区间内没有任何给定向量。这个区间内的方向向量方向虽然不完全一致，但是其它所有给定向量在这个区间内的方向投影正负性都是确定的。
- 我们只需要使用双指针来实时更新投影为正的向量区间，然后通过预处理的向量前缀和来计算区间内向量的矢量和，这样就能更新答案。
- 时间复杂度 $O(n \log n)$ 。

凸包

- 给定二维平面上的 n 个点，第 i 个点的坐标是 (x_i, y_i) 。
- 对于第 i 个点，以 $1/2$ 的概率将其染为黑色， $1/2$ 概率染为白色。
- 设 X 是黑色点组成的凸包上点的个数，求 X 的期望值 $E[X]$ 。
- 答案乘上 2^n ，对 $10^9 + 7$ 取模。
- 注意，这里的凸包一定是非退化的！即任意两点不重合，任意三点不共线！
- 题目不保证输入的点互不重叠。
- $1 \leq n \leq 2 \times 10^3$

凸包

- 由于乘上了 2^n ，答案相当于所有情况凸包点数的和。
- 先不考虑共线和重点情况。
- 从点入手，统计每个点对答案的贡献似乎很难。因为一个点在凸包内的充分必要条件是写不出来的。
- 但是我们可以发现，由于凸包是闭合图形，因此边数与点数相等，因此，统计边数之和与统计点数之和等价（当然，单个点凸包特殊处理，由两个点构成的凸包恰好有两条边）。

凸包

- 我们枚举边的起点，然后对其余所有点极角排序。
- 依次枚举边的终点，一条边（注意我们规定它是有向线段）在凸包内当且仅当该线段右手方向（你也可以规定左手方向）没有任何点，并且与它共线的点不能在这条线段外。
- 其余的点取和不取，对凸包形态都没有影响。记这些没有影响的点个数为 tot ，这条边对答案贡献显然是 2^{tot} 。

凸包

- 怎么统计这种点的个数呢？
- 当我们枚举的有向线段在一二象限上时，我们可以使用一个指针来得到第一个不在这条线段右手方向的点。否则使用一个指针来得到第一个在这条线段右手方向的点。
- 然后就是简单的加加减减。注意共线情况，线内的点可以任选，要在指针扫描同时记录共线的线内点个数。
- 如果要考虑重点呢？将所有重点用一个代替，并且记录出现次数。
- 一条边 (i, j) 的贡献乘上 $(2^{cnt_i} - 1)(2^{cnt_j} - 1)$ 。并且统计自由点个数的時候使用前缀和即可。
- 时间复杂度 $O(n^2 \log n)$ 。具体实现细节还挺多的。

Scan Line × Convex Polygon



Crossfire

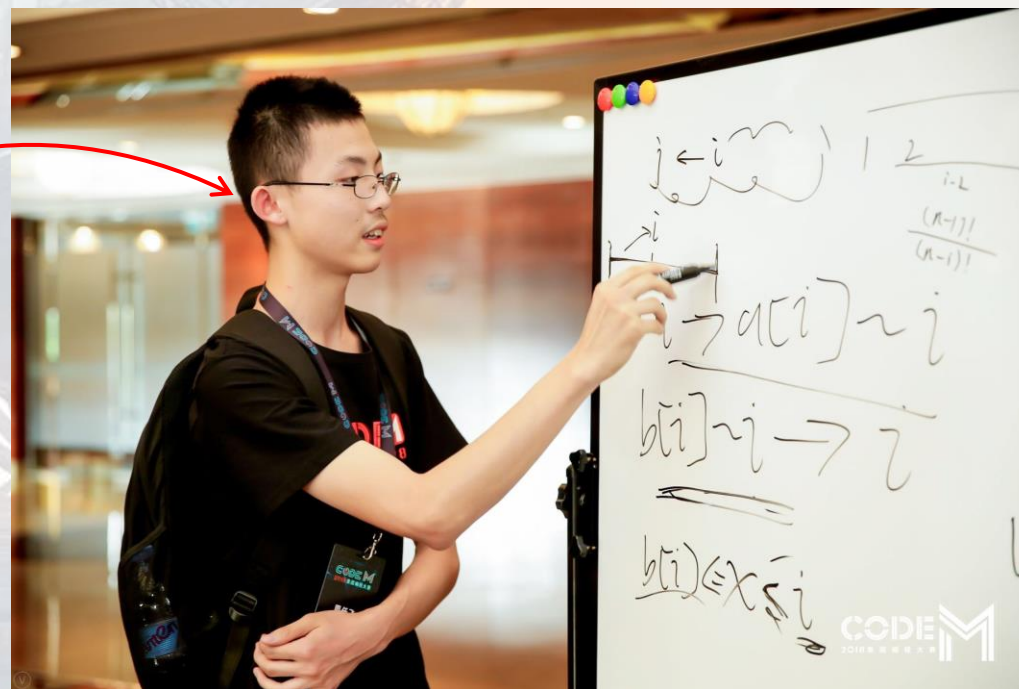
CodeM2018 Final, F

- 二维平面上一共有 n 个防御塔，第 i 个塔位于 (x_i, y_i) 的位置，这 n 个防御塔形成了一个严格的凸多边形。
- 有 m 道防御网，你可以把第 i 道防御网看做连接第 u_i 个防御塔和第 v_i 个防御塔的一条线段，穿越它就需要消耗 c_i 的血量。
- 请计算从 (sx, sy) 位置走到 (tx, ty) 位置，最少要消耗的血量。
- $1 \leq n, m \leq 10^5$

Crossfire

CodeM2018 Final, F

- 这是一道很简单的扫描线题，可惜现场时没有多少人过。
- 估计是看到计算几何都被劝退了。
- 一血是学军中学的 **orbitingflea** 拿下的（嘤嘤嘤本来我要抢的）。



CodeM2018 Final, F

Crossfire

- 同时插播一条美团爸爸的硬广。
- 这个比赛真的办得非常用心。
- 这么好的比赛可惜今年不办了。



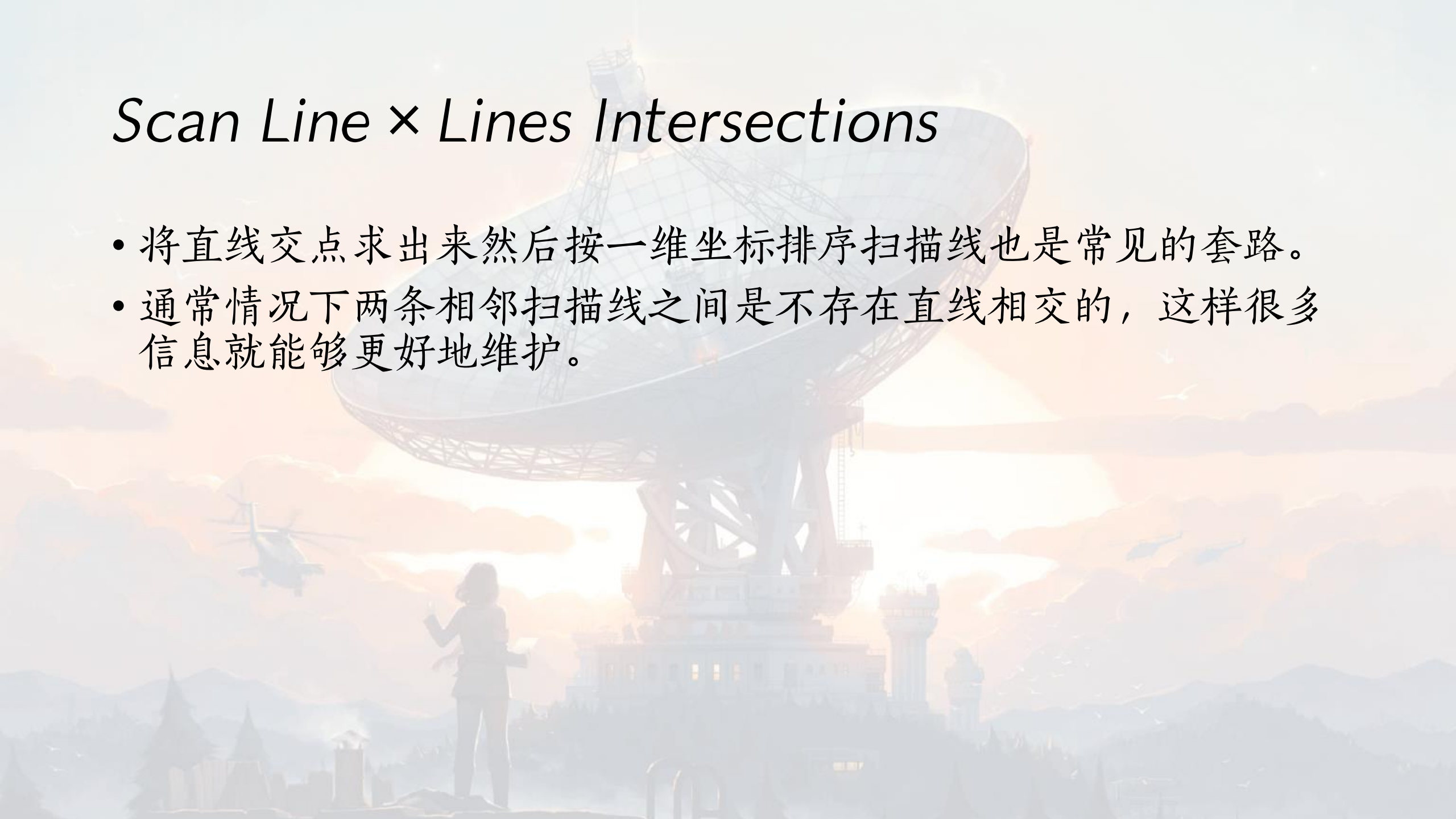
Crossfire

CodeM2018 Final, F

- 分两种情况讨论：
 - 直接在多边形内部从起点走到终点。
 - 从起点走出多边形再走到终点。
- 如果不走出多边形，最小代价显然是已经固定了的。
- 如果走出多边形，可以看成两个点走出多边形代价之和。
- 将线段看成区间。因为是凸多边形，所以一个点走出多边形的代价等于包含了出多边形时那条边界的区间代价之和。
- 扫描线求最少代价即可。
- 时间复杂度 $O((n + m) \log n)$ 。

Scan Line × Lines Intersections

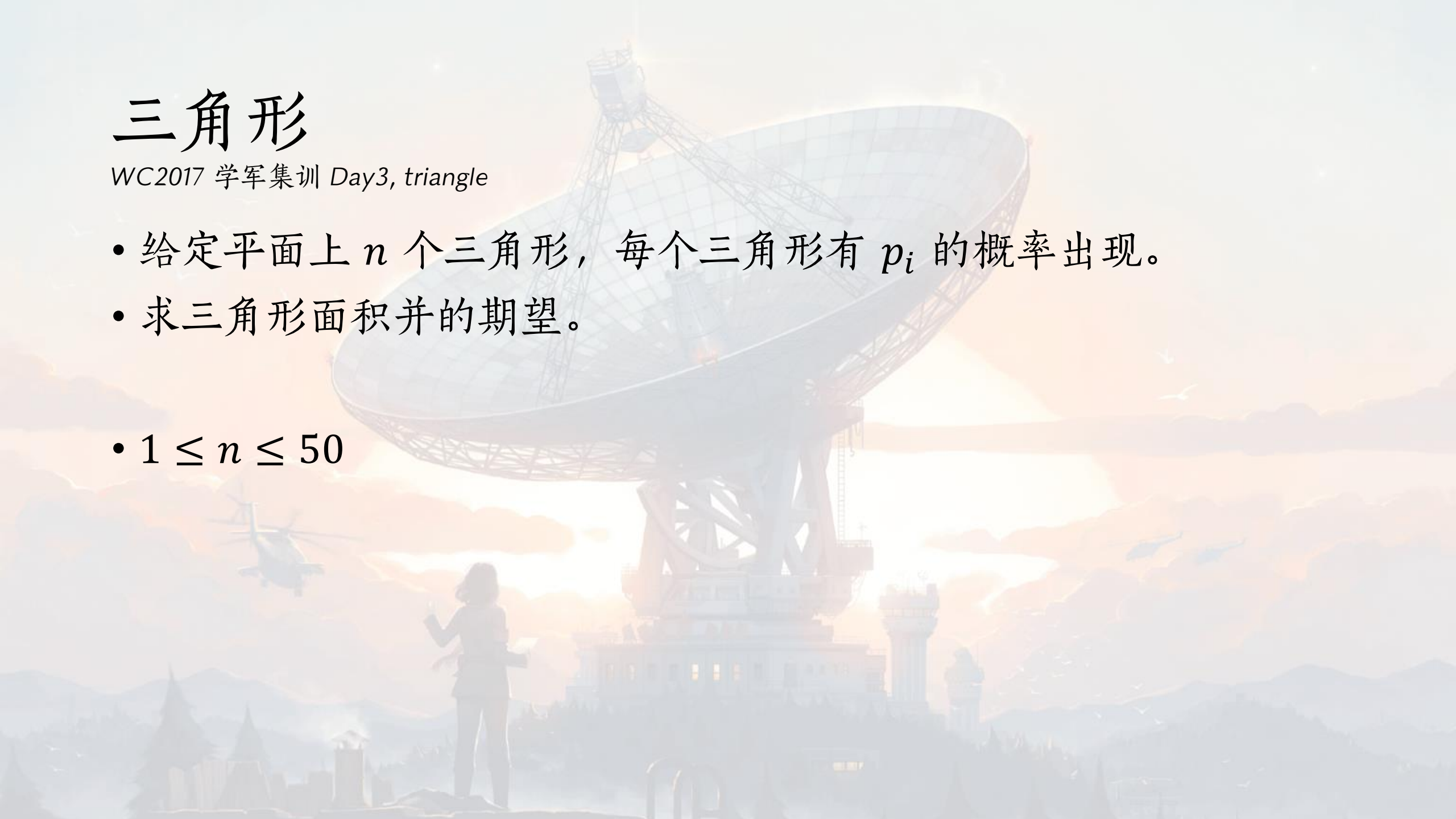
- 将直线交点求出来然后按一维坐标排序扫描线也是常见的套路。
- 通常情况下两条相邻扫描线之间是不存在直线相交的，这样很多信息就能够更好地维护。



三角形

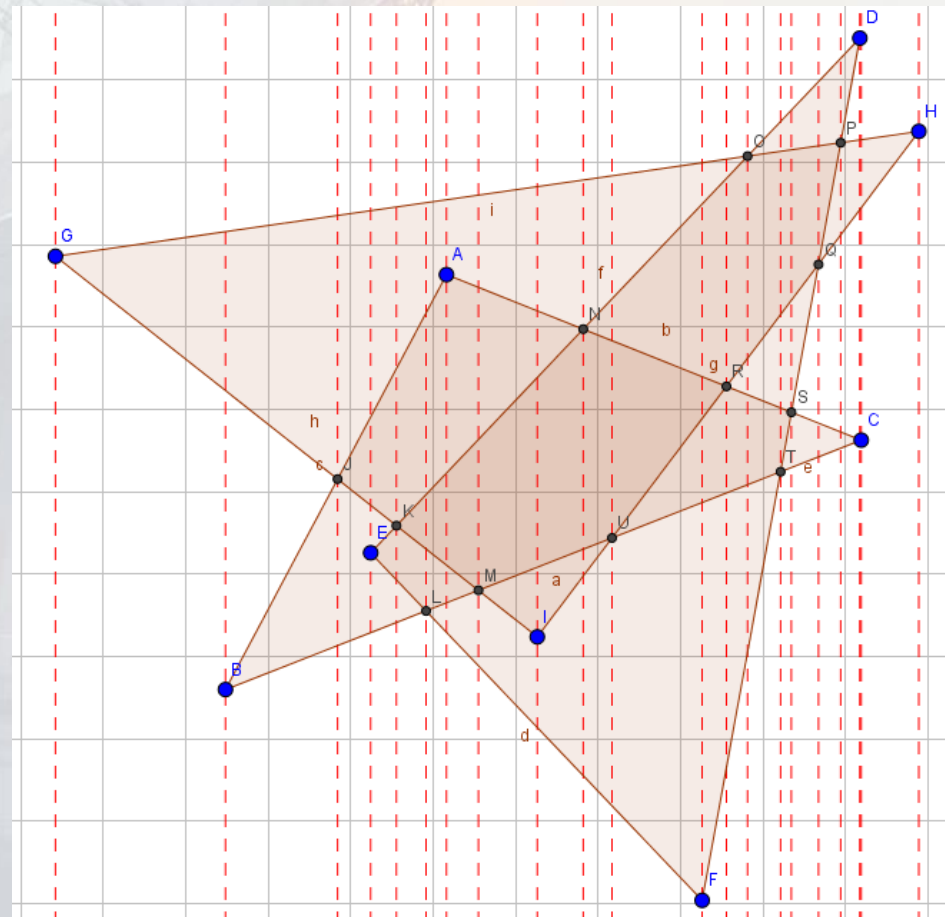
WC2017 学军集训 Day3, triangle

- 给定平面上 n 个三角形，每个三角形有 p_i 的概率出现。
- 求三角形面积并的期望。
- $1 \leq n \leq 50$



三角形

- 扫描线，关键的 x 坐标包括三角形的三个顶点的 x 坐标以及所有三角形交点的 x 坐标。
- 一个扫描区间内有若干个梯形的块，显然它们互相独立，可以分开统计。
- 枚举扫描区间内的每一个梯形区域，然后再枚举每一个三角形，得出这个区域出现的概率，乘上面积累加到面积并期望那里。
- 时间复杂度 $O(n^4)$ 。



Scan Line × Arcs

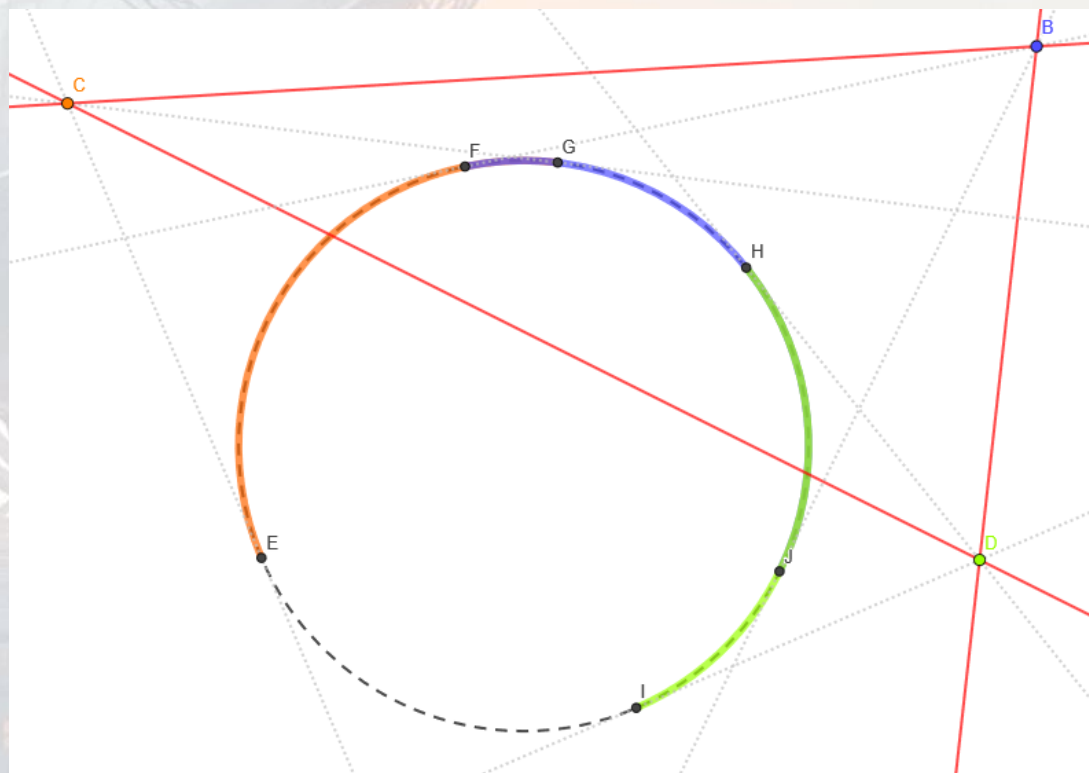


气球

- 给定 n 个互不重合的整点。
- 原点处有一个半径为 d 的圆。
- 你要从这 n 个点中选出两个不同的点，然后将产生一条经过这两个点的直线。
- 求使得直线与圆相交（不包括相切）的方案数。
- $2 \leq n \leq 2 \times 10^5$

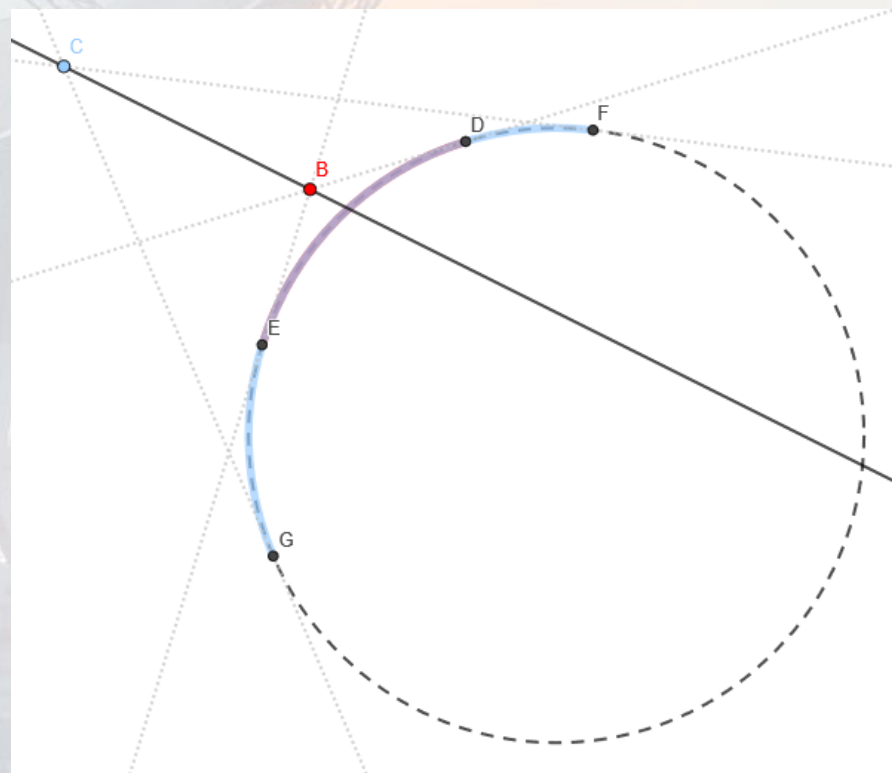
气球

- 过圆内点直线肯定会过圆，单独计算。
- 考虑从圆外点向圆作切线，转化成若干个弧区间。
- 两点连线过圆似乎等价于两点对应圆弧无交？



气球

- 似乎有点问题？冷静分析：将相交定义为严格相交（即不包含）即可。
- 那么统计不穿过圆的直线就等价于统计严格相交的区间。
- 接下来和普通的序列扫描线就差不多了，极角排序之后用树状数组处理即可。跨象限处略有细节。
- 时间复杂度 $O(n \log n)$ 。



Scan Line × Circles

- 圆虽然是曲线图形，却也可以通过扫描线化曲为直。



Moonmist

2009 Asia Wuhan Regional Contest Online, G

- 给定平面上 n 个两两相离的圆，请求出距离最近的一对圆。
- 相离的两个圆距离定义为圆心距离减去两圆半径和。
- $1 \leq n \leq 5 \times 10^4$

Moonmist

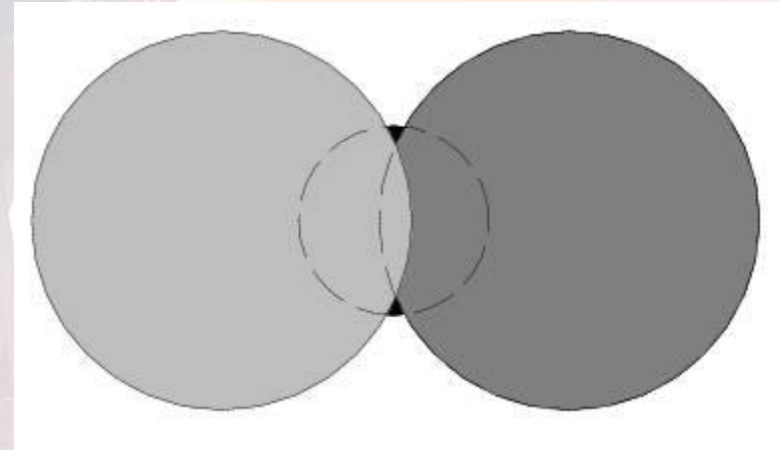
2009 Asia Wuhan Regional Contest Online, G

- 二分答案，转化为判断是否有圆相交/内含。
- 怎么判定？考虑将圆左边界和右边界列为扫描线。
- 对于每对相邻扫描线，考察跨过这个区间的所有圆的相交情况？超时怎么破？
- 我们只在一个圆被加入/删除的时候检查。
- 那么如果是加入，我们只需要找到这个圆左边界的右边最近的右边界，考察跨过这个区间的所有圆与加入圆的相交情况；如果是删除，我们就找到这个圆右边界的左边最近的左边界，考察跨过这个区间的所有圆与删除圆的相交情况。
- 显然，维护这些边界只需要简单的 *two pointer* 即可。
- 怎么判定呢？可以发现圆心向扫描线作垂线，是覆盖区域对称中心。
- 假定按照圆心的 y 坐标排序，如果第 i 个圆和第 j 个圆相交，那么这个区间内的圆一定都存在， i 到 j 之间的每一个圆与 i 和 j 至少一个有交。因此只需要检查圆心高度相邻圆。
- 用一个 *set* 维护就好了。时间复杂度 $O(n \log X \log n)$ 。

Viva Confetti

Japan 2002 Kanazawa

- 平面上顺次放入 n 个圆，问有多少个圆最终可见。
- $1 \leq n \leq 100$



Viva Confetti

Japan 2002 Kanazawa

- 将所有圆的左右边界和交点所在垂线列为扫描线。
- 通过求交将圆映射成直线上的一个区间。
- 圆的可见问题等价于区间的可见问题。
- 离散化加并查集合并即可。
- 时间复杂度 $O(n^3(\log n + \alpha(n)))$ 。

Scan Line × Binary Search

- 扫描线和二分算法结合起来，可以解决一类寻找平面上满足特定条件的点的问题。



Judging the Trick

2017–2018 ACM–ICPC, NEERC, Moscow Subregional Contest, J

- 有一个 $w \times h$ 的矩形，内部覆盖有 n 个三角形。
- 这些三角形不会超过矩形区域，两两之间可能会重叠。但是保证矩形内部存在点 **严格** 没有被三角形覆盖，所有三角形的面积之 **和** 也 **严格** 小于 $w \cdot h$ 。
- 你需要输出矩形内部任意一个没有被任何三角形覆盖的点的坐标。
- $1 \leq n \leq 10^5, 1 \leq w, h \leq 10^4$

<https://codeforces.com/gym/101611/problem/J>

Judging the Trick

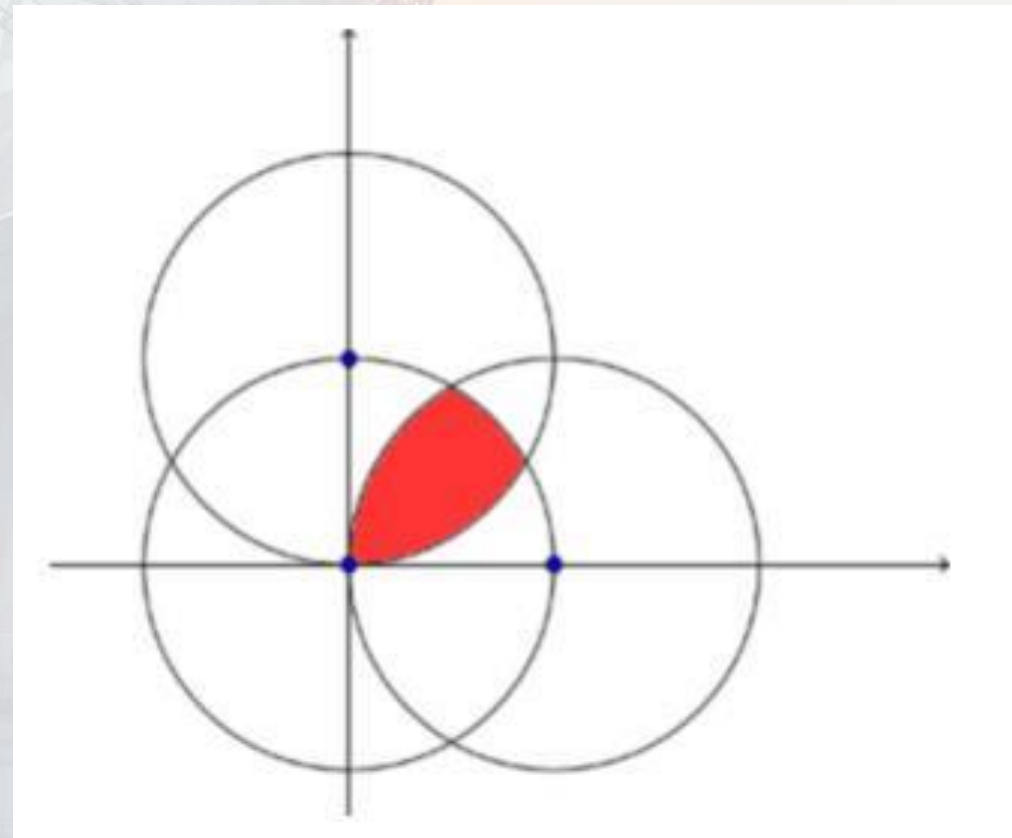
2017–2018 ACM–ICPC, NEERC, Moscow Subregional Contest, J

- 假如我确定了一条垂直于 x 轴的直线上存在答案，那么可以通过与三角形求交+扫描线来求答案。
- 怎么确定一条直线呢？注意到三角形面积之和严格小于 $w \cdot h$ ，那么直线两边至少存在一边三角形面积之和严格小于 $\frac{w \cdot h}{2}$ ，满足这个条件代表着那一边存在点没有被三角形覆盖。
- 所以采用二分的思路，每次取最中间那条直线验证，如果不行就将区间缩小到三角形面积之和小于 $\frac{w \cdot h}{2}$ 的一侧，注意穿过中线的三角形要截一下。
- 时间复杂度 $O(n \log X)$ 。

圆，圆，圆

POJ Challenge – 2011.04.10, C

- 给定平面上 n 个圆，判断它们是否有面积非零的交集。
- $1 \leq n \leq 10^5$



圆，圆，圆

POJ Challenge – 2011.04.10, C

- 假设我们已经知道了这些圆有交，如何找出一个交点？
- 考虑作一条垂线，通过求交将所有圆映射成这个垂线上的区间。
- 如果垂线上存在一个交点，那么这些区间一定有交。否则找出任意两个不相交的区间，求出它们对应圆的交点，如果这个交点在垂线左边，那么垂线要向左调整，否则向右调整。
- 我们把这个过程写成一个二分即可。刚开始的时候二分范围是由圆左右边界定义的区间的公共交集。
- 那么如果这些圆没交，无非是三种情况：
 - 由圆左右边界构成的区间公共交集为空。
 - 二分时发现两个相离/相切的圆
 - 二分足够次数仍没有答案（实测 20 次即可）。

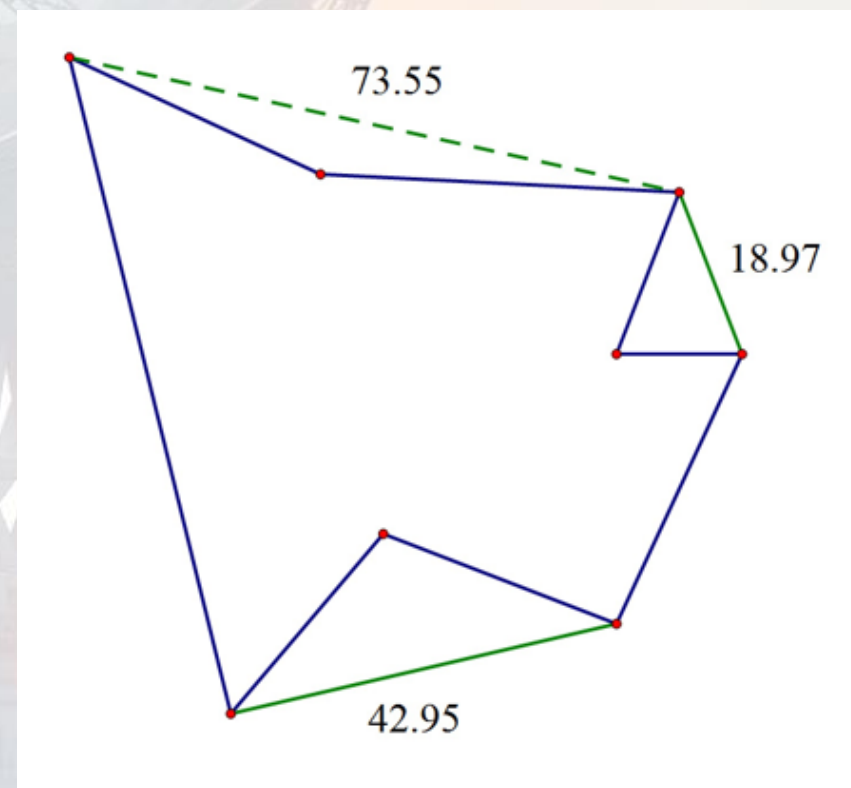
Pick's Theorem

- *Pick* 定理给出了一个计算点阵中顶点在格点上的简单多边形面积公式：
- 令 i 为多边形内部的点数， b 表示多边形边界上的点数。
- 则 $S = i + \frac{b}{2} - 1$
- 证明思路：多边形三角剖分
- *Step 1*: 证明如果 T 和 P 为两个满足 *Pick* 定理的简单多边形，那么多边形 TP 也满足 *Pick* 定理。
- *Step 2*: 证明任意三角形都满足 *Pick* 定理。
 - *Step 2.1*: 证明任意平行于轴线的矩形满足 *Pick* 定理。
 - *Step 2.2*: 证明任意直角边平行于轴线的直角三角形都满足 *Pick* 定理。
 - *Step 2.3*: 证明任意三角形都满足 *Pick* 定理（将三角形看做矩形减去 3 个 *Step 2.2* 中的直角三角形）。

Castle Wall

ACM-ICPC Japan Alumni Group Practice Contest, for World Finals, Tokyo, Japan, 2008-02-23

- 给定一个简单多边形的墙，你可以选择两个点相连建造一些新的墙。新的墙不能和原来的墙严格相交，总长不能超过 L 。
- 求方案使得增加的面积最大。
- 点数 $n \leq 64$ ，坐标都是 0 至 100 之间的整数。



Castle Wall

ACM-ICPC Japan Alumni Group Practice Contest, for World Finals, Tokyo, Japan, 2008-02-23

- 直接做好像很难做，考虑题目的特殊条件？所有点都是整点？
- 根据 *Pick* 定理（其实应该可以不用），整点多边形的面积的两倍一定是整数。
- 考虑 *dp*，令 $f_{u,area}$ 表示当前枚举到多边形的点 u ，增加的面积为 $area$ 的最小新墙总长。
- $O(n^3)$ 预处理多边形的两个点是否能连墙以及连起来之后增加的面积。
- 那么总的时间复杂度就是 $O(n^3 + n^2S)$ 的。
- 至于边界条件？显然 y 坐标最小的点（如果有多个，取 x 坐标最小的）一定不会被某个墙包括在内，把这个点作为起始点就好了。

残破的二维平面

51nod 算法马拉松31, D

- 给定一个的网格图（横坐标 $[1, n]$ ，纵坐标 $[1, m]$ ），上面有 $n \times m$ 个格点。
- 你要在这 $n \times m$ 个格点中选出四个点 p_1, p_2, p_3, p_4 。
- 然后你会用线段分别连接 p_1, p_2 以及 p_3, p_4 。
- 计算有多少种选择方法能使两条线段相交（端点处也算）。
- 答案对 $10^9 + 7$ 取模。
- $1 \leq n, m \leq 70$

残破的二维平面

51nod 算法马拉松31, D

- 这题分成四个子问题.
 1. 四个点四点共线
 2. 三个点三点共线另外一个点不在直线上
 3. 四个点任意三点不共线且四个点的凸包是三角形
 4. 四个点任意三点不共线且四个点的凸包是四边形
- 在不考虑选择顺序的情况下, 容易发现子问题 1 中的每种选择方法会对答案产生 16 的贡献, 子问题 2/4 对答案产生 8 的贡献, 子问题 3 对答案不产生贡献。
- 子问题 1/2 都可以枚举在线上的两个点在 $O((nm)^2)$ 的时间内计算出来。
- 考虑子问题 3, 我们就要求在网格图上任意三角形的内部整点数的和。
- 直接枚举是 $O((nm)^3)$ 的, 无法接受。

残破的二维平面

51nod 算法马拉松31, D

- 注意到暴力枚举其实重复计算了许多形态相同的三角形。
- 我们考虑枚举三角形的外接长方形。
- 用三个点去卡四个边界，于是只有两种可能：
 - 两个点卡住长方形的两个对角顶点，剩下一个点随便选
 - 一个点卡住长方形的一个顶点，剩下两个点在边界上。
- 使用Pick定理就可以 $O((nm)^2)$ 算了。
- 总时间复杂度 $O((nm)^2)$ 。

Advanced Algorithm

很好，恭喜你学会了基础内容。
坐稳了，现在我们才刚刚开始！

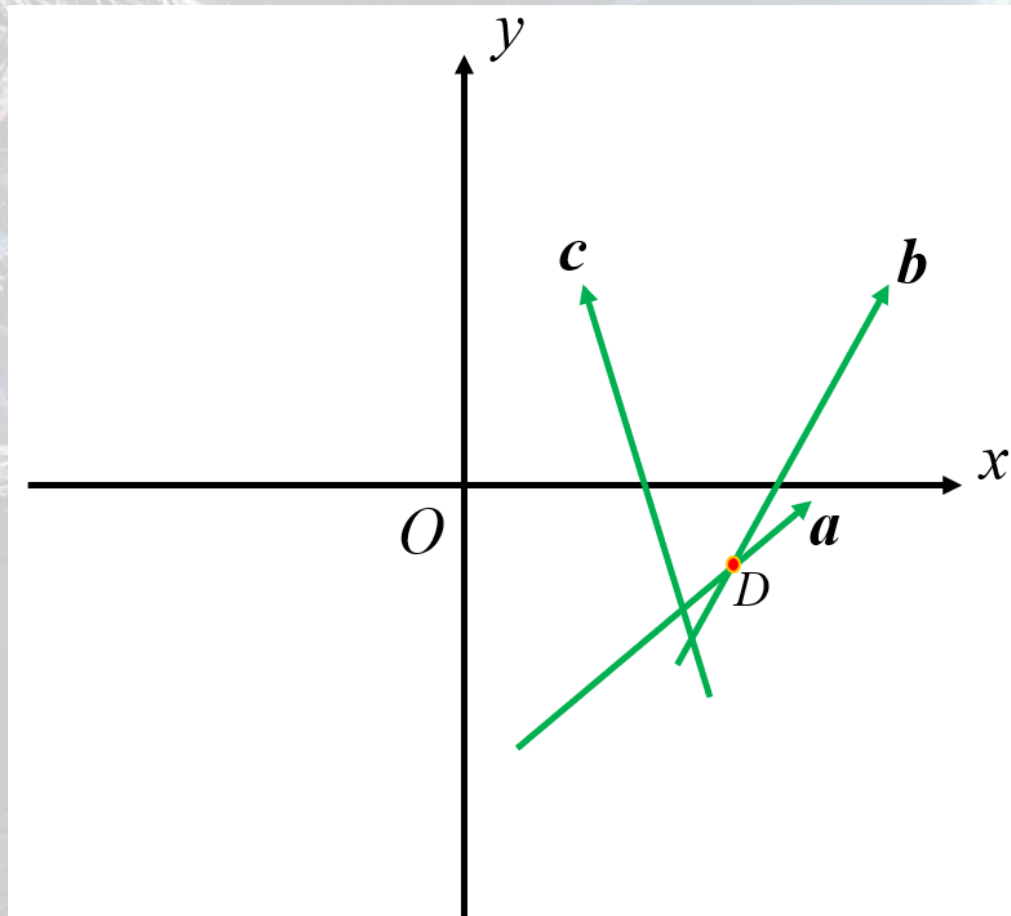


Halfplane Intersection

- 理论上讲，半平面交和凸包互为对偶问题，因此存在相似的解法。
- 半平面交可以使用排序增量法来计算，不过与凸包的计算要利用栈不同，半平面交的计算要利用的是双端队列，因为半平面可能会“绕了一圈”对最开始加入的半平面产生影响。

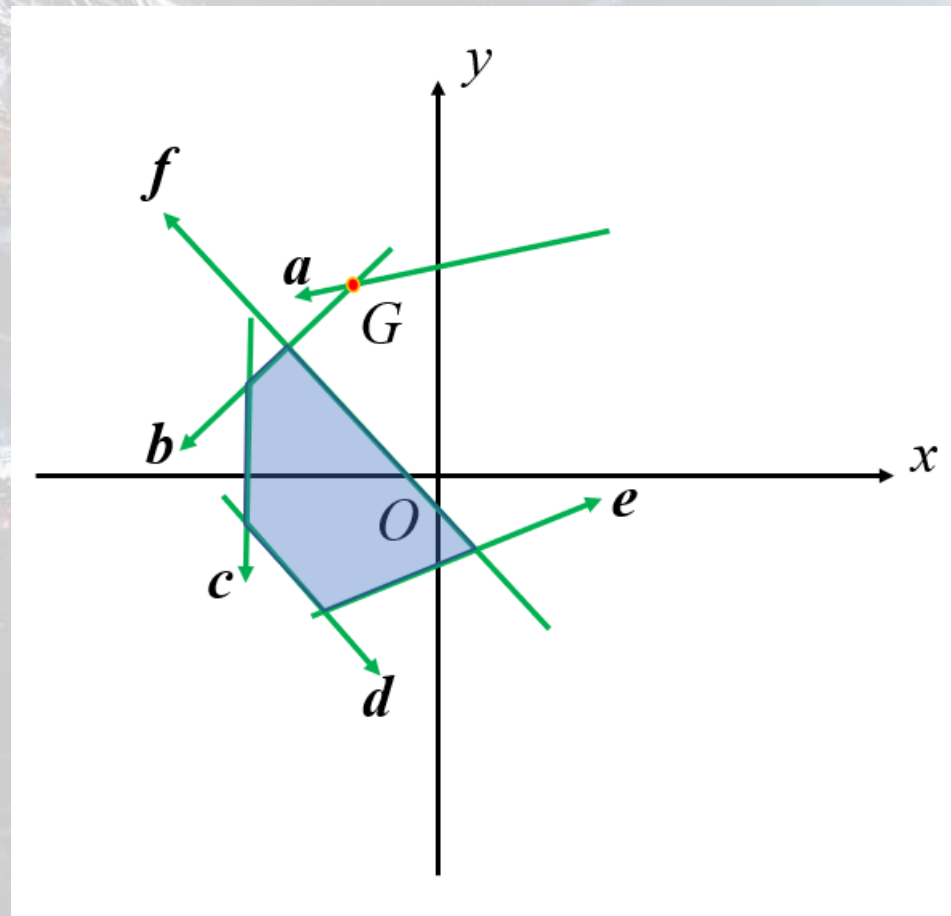
Incremental Method

- 这是最普通的情况：和凸包一样，新加入的半平面导致了上一个加入的半平面的失效。
- 判断标准是前两个半平面的交点在新加入半平面之外。
- 直接弹出队尾即可。



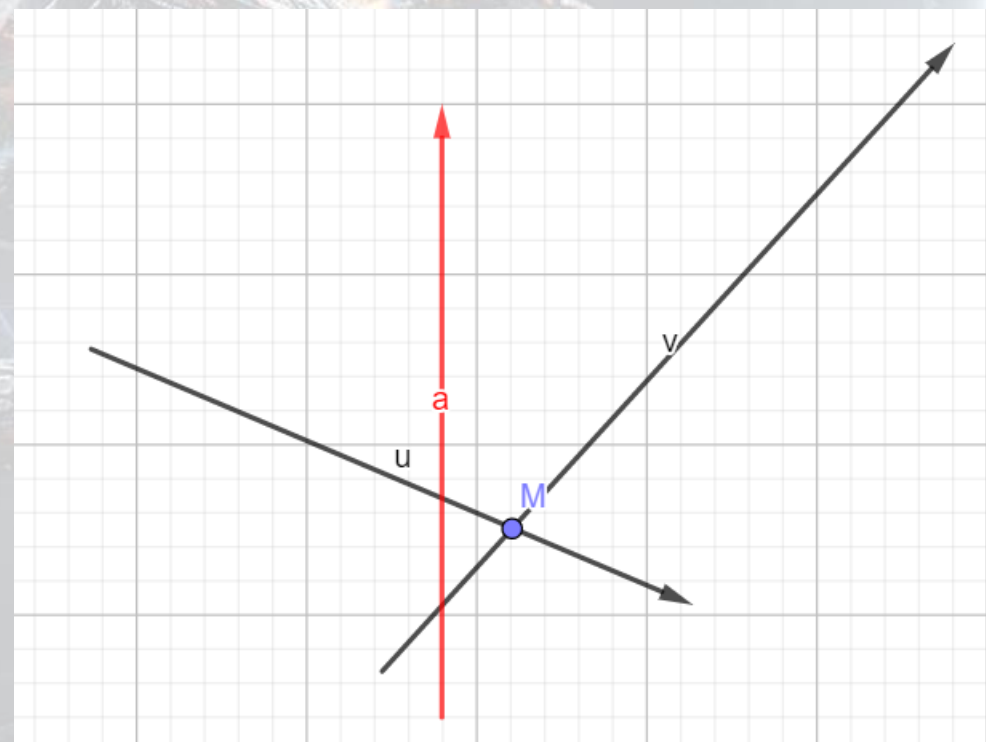
Incremental Method

- 这是半平面交特有的情况：新加入的半平面 f 导致了最先加入的半平面 a 的失效。
- 判断标准是最先加入的两个半平面交点在新加入的半平面之外。
- 直接弹出队头即可。
- 既然到新加入的半平面会对最先加入的半平面产生约束，那么最先加入的半平面肯定也会对新加入的半平面产生约束。但是我们目前只考虑了前者。
- 因此在算法执行到最后时，我们要用队头对队尾执行类似的检测及弹出操作。



Incremental Method

- 半平面交的实现细节比凸包要多。
- 例如最重要的一点：先弹出不合法队尾，再弹出不合法队头！
- 原因如右图，根据判断法则，新加入的半平面 a 会使半平面 u 和 v 都不合法。
- 而删除 u 显然是错误的。
- 此外，为了排除一些无界的情况带来的错误，增强代码的鲁棒性，建议在执行算法之前先加上四个无穷远的矩形边界。



Incremental Method

```
inline db HPI()
{
    pl[++ n] = L(P(0 , 0) , P(0 , 1));
    pl[++ n] = L(P(0 , F) , P(1 , 0));
    pl[++ n] = L(P(F , F) , P(0 , -1));
    pl[++ n] = L(P(F , 0) , P(-1 , 0));
    sort(pl + 1 , pl + 1 + n , cmp);
    int n0 = 0;
    for (int i = 1; i <= n; ++ i)
    {
        if (!n0) pl[++ n0] = pl[i];
        else if (!equ(arg(pl[n0].v) , arg(pl[i].v))) pl[++ n0] = pl[i];
        else if (in(pl[i].p , pl[n0])) pl[n0] = pl[i];
    }
    n = n0 , hd = 1 , tl = 2;
    hpi[1] = pl[1] , hpi[2] = pl[2];
    for (int i = 3; i <= n; ++ i)
    {
        for (; hd != tl && !in(ict(hpi[tl] , hpi[tl - 1]) , pl[i]); -- tl);
        for (; hd != tl && !in(ict(hpi[hd] , hpi[hd + 1]) , pl[i]); ++ hd);
        if (hd == tl && sgn(hpi[hd].v ^ pl[i].v) <= 0) return 0;
        hpi[++ tl] = pl[i];
    }
    for (; hd != tl && !in(ict(hpi[tl] , hpi[tl - 1]) , hpi[hd]); -- tl);
    int m = tl - hd + 1;
    for (int i = 1 ; i < m; ++ i) pts[i] = ict(hpi[hd + i] , hpi[hd + i - 1]);
    pts[0] = pts[m] = ict(hpi[tl] , hpi[hd]);
    db ret = .0;
    for (int i = 0; i < m; ++ i) ret += pts[i] ^ pts[i + 1];
    return ret * .5;
}
```


小凸想跑步

SCOI2015 Day1, convex

- 小凸晚上喜欢到操场跑步，今天他跑完两圈之后，他玩起了这样一个游戏。
- 操场是个凸 n 边形， n 个顶点按照逆时针从 $0 \dots n-1$ 编号。现在小凸随机站在操场中的某个位置，标记为 P 点。将 P 点与 n 个顶点各连一条边，形成 n 个三角形。如果这时 P 点， 0 号点， 1 号点形成的三角形的面积是 n 个三角形中最小的一个，小凸则认为这是一次正确站位。
- 现在小凸想知道他一次站位正确的概率是多少。
- $3 \leq n \leq 10^5$

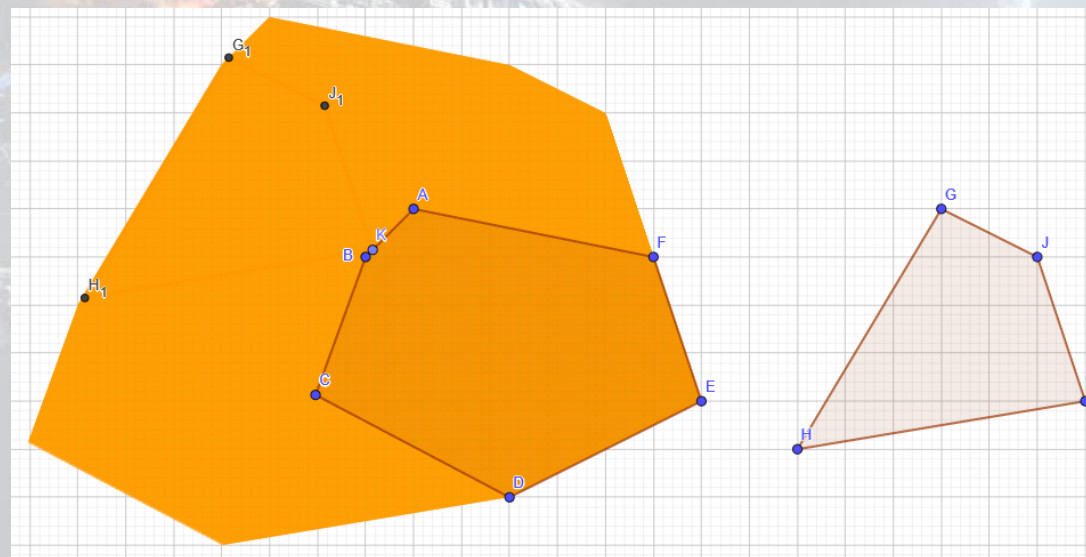
小凸想跑步

SCOI2015 Day1, convex

- 用叉积表达一下面积，发现对 P 的坐标就是一堆半平面的限制。
- 直接半平面交然后求出面积就能算概率了。
- 时间复杂度 $O(n \log n)$ 。

Minkowski Sum

- $A + B = \{a + b | a \in A, b \in B\}$
- $A - B = \{a - b | a \in A, b \in B\}$
- 如何快速计算两个凸多边形的 Minkowski 和?
- 将两个凸多边形所有边放在一起按照极角排序, 然后顺次选取作为 Minkowski 和的轮廓。
- 但是这样我们只知道 Minkowski 和的形状, 具体的位置还不知道。
- 可以发现 Minkowski 的最低点就是两个凸多边形的最低点的和。利用最低点来恢复整个多边形就好了。



部落战争

JSTSC2018 Round2 Day2, war

- 二维平面上有 n 个黑点和 m 个白点。
- 对于平面上每一个点，如果它被三个同色点围成的三角形（可以退化成一条线段）所包含（包括边界），那么这个点就被这种颜色支配。
- 现在有 q 次询问，每次给定一个向量 (dx, dy) 并将所有白点位移这个向量，请判断平面上是否存在一个点同时被黑色和白色支配。
- $0 \leq n, m, q \leq 10^5$

部落战争

JSTSC2018 Round2 Day2, war

- 支配的区域显然是一个凸包。
- 将白点凸包与黑点凸包作 *Minkowski* 差，得到的凸多边形就是合法的位移向量区域。
- 现在问题变为判断点是否在凸多边形内。在线二分或者离线扫描线即可。
- 时间复杂度 $O(n \log n)$ 。

distance

- 二维平面上有 n 个点，每个点位于 (x_i, y_i) ，颜色为 c_i 。
- 请计算最远的异色点对距离。
- $2 \leq n \leq 2.5 \times 10^5$

distance

- 如果只有两种颜色怎么做?
- 距离等价于向量差模长。
- 这启示我们只需要对两种颜色凸包作 *Minkowski* 差然后求模长最大的点即可。
- 多种颜色? 对颜色分治即可。
- 时间复杂度 $O(n \log^2 n)$ 。

Nearest Neighbor

- 平面最远点对问题的一般做法为旋转卡壳，这里不再累赘。
- 如何做平面最近点对？
- 我会 *KD-Tree*!



- 下面介绍一个时间复杂度为 $O(n \log n)$ 的分治算法。

Nearest Neighbor

- 和一般的分治算法类似，我们将所有点以 x 坐标为第一关键字， y 坐标为第二关键字排序，然后从中间将点集 S ，分成两个大小相当的部分 S_1, S_2 。
- 递归求出 S_1, S_2 的最近点对距离，设为 h_1, h_2 ，令较小值为 h 。
- 下面考虑如何计算跨过两个点集的最近点对距离。

Nearest Neighbor

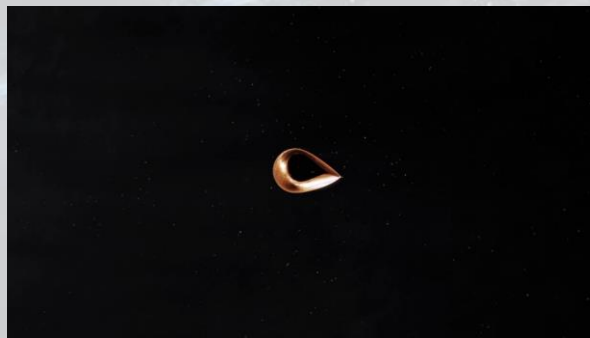
- 令分界线的 x 坐标为 x_{mid} 。
- 定义点集 $A = \{p_i | p_i \in S, |x_i - x_{mid}| < h\}$
- 对于这个点集中每一个点 p_i ，我们同样希望找到与它距离可能小于 h 的点集 $B(p_i) = \{p_j | p_j \in A, y_i - h < y_j \leq y_i\}$
- 如果我们将 A 中的点按照 y 坐标排序，那么 p_i 相邻的若干点即为 $B(p_i)$ 。
- 于是我们只需要对每个 p_i 都在 $B(p_i)$ 中找到所属集合不同的最近点。
- 等等这不还是暴力吗？？？

Time Complexity

- 冷静分析: $|B(p_i)|$ 的规模是多少?
- $B(p_i)$ 中的点 y 坐标都在 $(y_i - h, y_i]$ 内, 且 $B(p_i)$ 内所有点横坐标都在 $(x_{mid} - h, x_{mid} + h)$ 内, 构成了一个 $2h \times h$ 的矩形。
- 将这个矩形拆成两个 $h \times h$ 的正方形, 由已知, 每个正方形内点的距离都不超过 h 。
- 将正方形再拆成四个 $\frac{h}{2} \times \frac{h}{2}$ 的小正方形, 每个小正方形内最大距离为对角线长度 $\frac{h}{\sqrt{2}} < h$, 因此一个小正方形内最多只能有一个点。
- 于是矩形内最多就只剩下 8 个点了, 去掉 p_i 本身, 也就 7 个, 即常数规模。
- 至于算法中的排序可以用归并优化, 总时间复杂度就是 $O(n \log n)$ 。
- 对算法略加改造还能做平面最小周长三角形。

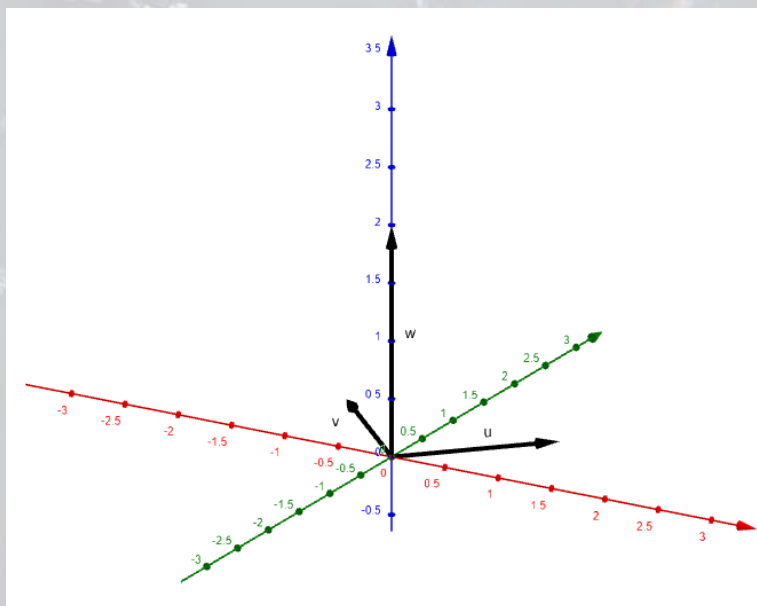
Basic 3D Geometry

- 我的建议是：
- “傻孩子们，快跑啊！”（《三体：黑暗森林》－丁仪）



Point & Vector

- 三维点/向量 (x, y, z)
 - 点积、叉积



```
struct P3
{
    db x , y , z;

    inline P3 (db x_ = 0 , db y_ = 0 , db z_ = 0){x = x_ , y = y_ , z = z_;}

    inline P3 operator + (P3 const p)const{return P3(x + p.x , y + p.y , z + p.z);}
    inline P3 operator - (P3 const p)const{return P3(x - p.x , y - p.y , z - p.z);}
    inline P3 operator * (db const k)const{return P3(x * k , y * k , z * k);}

    inline P3 operator ^ (P3 const p)const
    {
        return P3(y * p.z - z * p.y , z * p.x - x * p.z , x * p.y - y * p.x);
    }

    inline db operator * (P3 const p)const{return x * p.x + y * p.y + z * p.z;}
};
```

Line & Plane

- 线类型的表达依然可沿用二维的方法，记录位置向量和方向向量。
- 一般使用点法式 (p, \mathbf{n}) ，为了方便 \mathbf{n} 一般为单位向量。
- 如果是给定三个点，要转化为点法式直接用叉积得到法向量再调整为单位向量就好了。

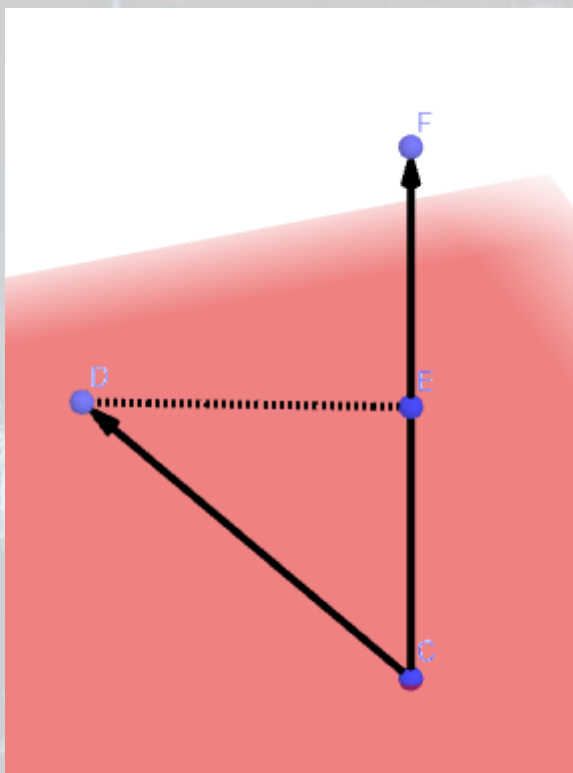
Line & Plane

- 线类型的表达依然可沿用二维的方法，记录位置向量和方向向量。
- 一般使用点法式 (p, \mathbf{n}) ，为了方便 \mathbf{n} 一般为单位向量。
- 如果是给定三个点，要转化为点法式直接用叉积得到法向量再调整为单位向量就好了。

Distance

Point vs Plane

- 由于法向量被规定为单位向量，直接点乘即可得到距离。

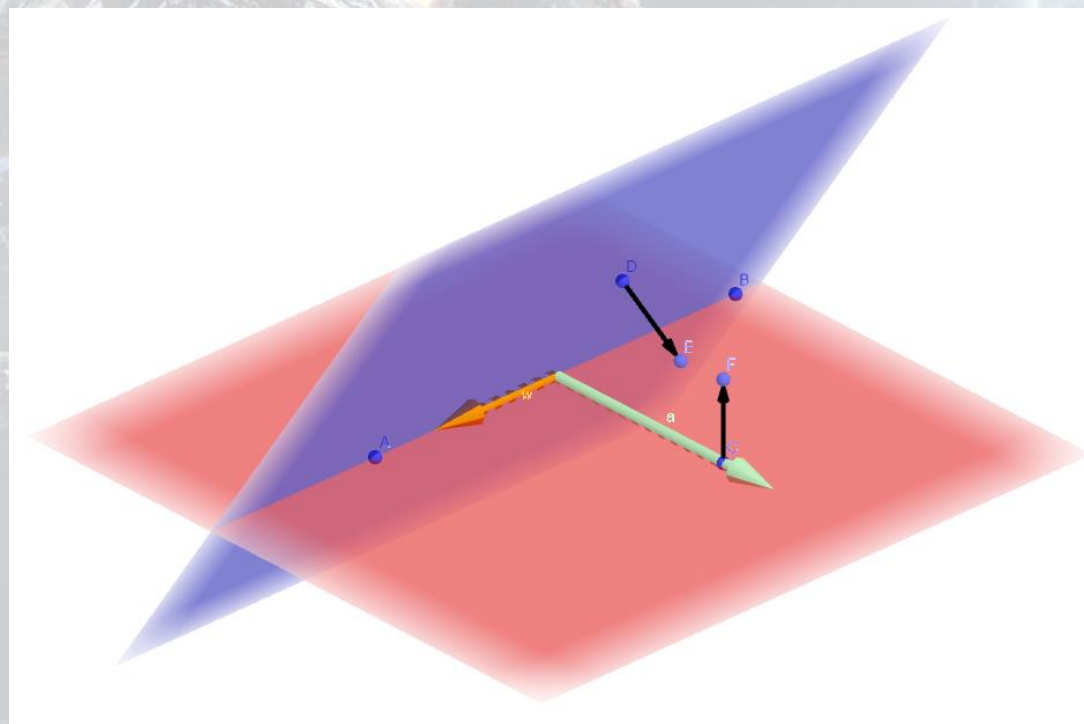


Other Operations


- 四点共面：任意三点作平面再与第四点求距离。
- 直线求交：先判共面，然后直接用二维的方法。
- 直线和平面求交：比例关系。
- 四面体（平行六面体）体积及半空间位置判定：三阶行列式（混合积）。
- 点是否在四面体内：分割求体积和。

Intersection Of Planes

- 法向量叉乘得到方向向量。
- 方向向量叉任一法向量得到其平面内垂直于交线的贴面向量。
- 利用贴面向量和该平面上一点作平面内垂直于交线的直线，与另一平面求交，作为交线的位置向量。

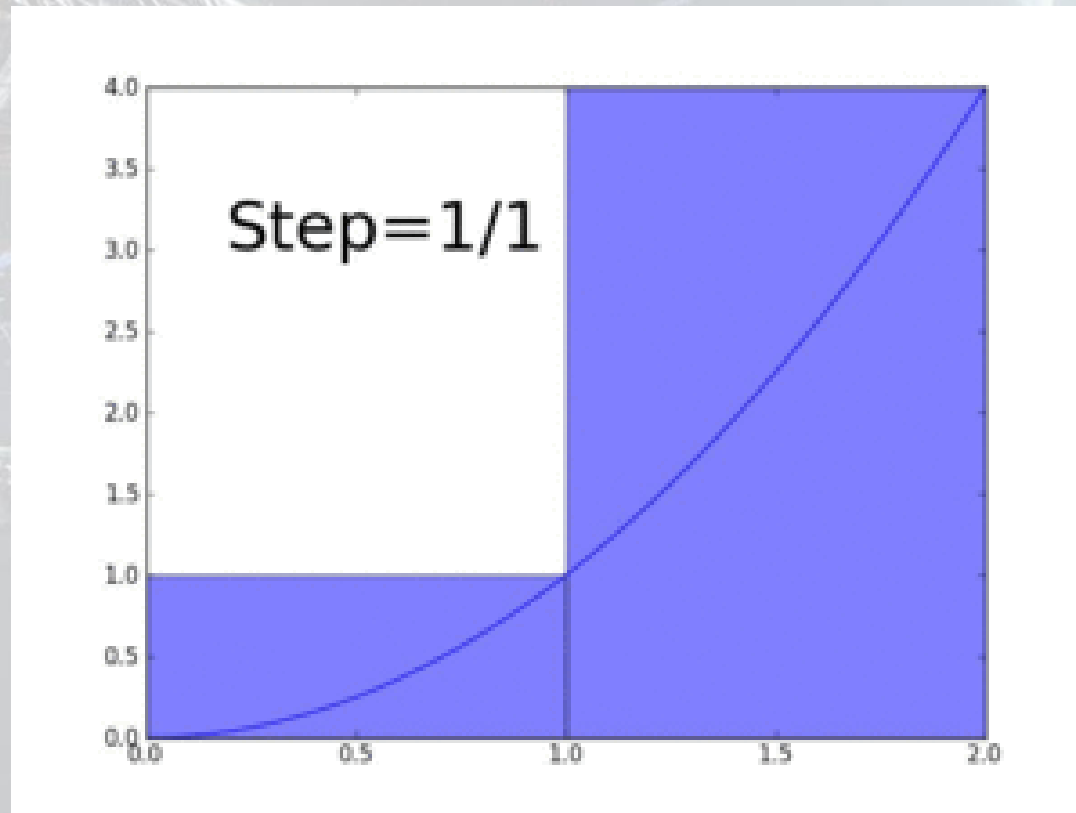


Numerical Algorithm

- 什么是数值算法?
- 一个一个 EPS 地模拟去爆精度?
- 
- 很好，你已经掌握了最简单的数值算法。
- 但是这样太逊了，很难平衡时间效率和答案精度。
- 我们需要一种能够在更低时间复杂度内拟合答案的算法。

Simpson's Rule

- 很多时候，我们要算定积分时都要面对一个不规则的函数，你发现你并不能求它的原函数，这时候我们往往要使用一些数值算法。
- 在高中数学课本里面我们学习了矩形积分法，这就是一个典型的数值算法。
- 但是在实测中，这种积分法效率不怎么好。为了在有限的时间内计算更高精度的定积分，我们得使用一种更加高明的数值算法。

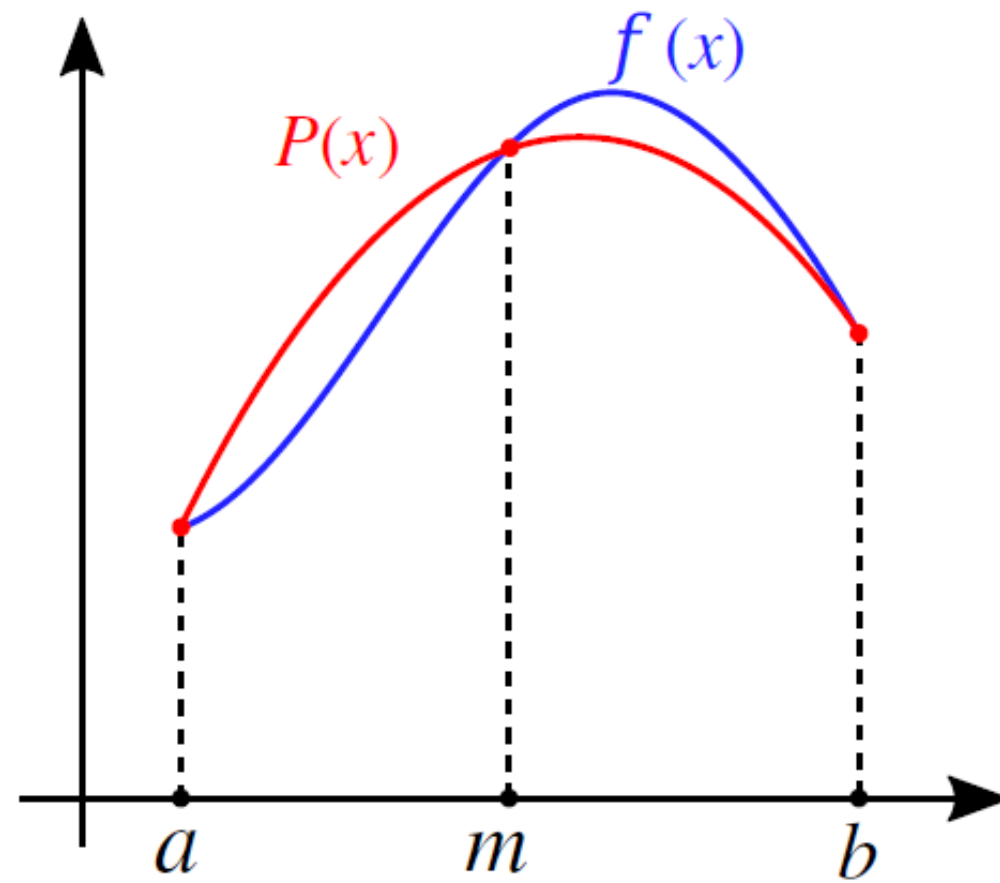


Simpson's Rule

- 在矩形积分法中，我们采用了化曲为直的思想，但是这不可避免地造成了很多的“空洞”。
- 辛普森积分采用一种全新的思想：“化曲为曲”。将复杂的曲线近似地看成若干条简单的抛物线，即所谓“曲线积分法”。
- 具体思路是，假设我们要求

$$\int_a^b f(x) dx$$

- 把这段曲线看成是一条过 $f(a), f\left(\frac{a+b}{2}\right), f(b)$ 三点的抛物线 $P(x)$ 。
- 令抛物线 $P(x) = ax^2 + bx + c$ 。
- 计出原函数 $P'(x) = \frac{1}{3}ax^3 + \frac{1}{2}bx^2 + cx$ 求出这一段的近似定积分值。



Simpson's Rule

- 具体就不推导了，最后式子是：

$$\int_a^b f(x)dx \approx \frac{b-a}{6} \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

- 记忆即可。
- ~~怎么记？除了 2 剩下两个数字是什么心里没点数？~~

Adaptive Simpson's Rule

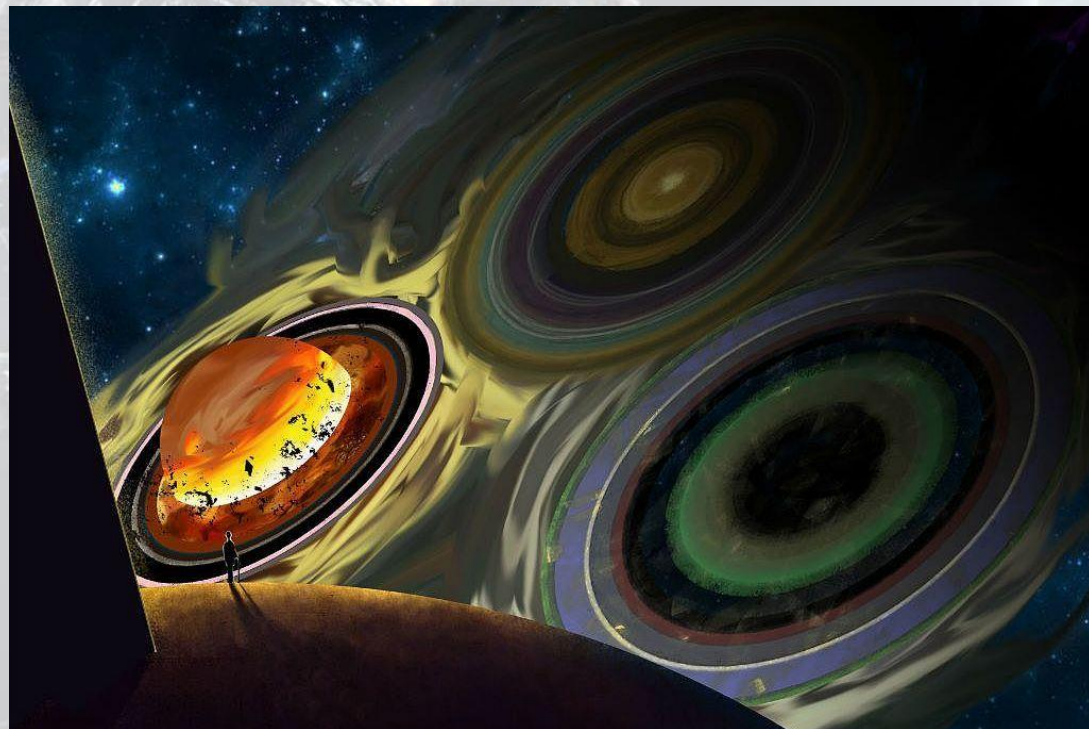
- 但是只作一次拟合精度显然是不够的。
- 怎么办？考虑递归拟合直到相差值在允许范围内。
- 具体来说，我们实现一个 $AdaptiveSimpson(l, r)$ 如下
 - 计算 $mid = \frac{l+r}{2}$
 - 计算 $f = Simpson(l, r), f_l = Simpson(l, mid), f_r = Simpson(mid, r)$
 - if $|f - f_l - f_r| \leq EPS$ return f
 - else return $AdaptiveSimpson(l, mid) + AdaptiveSimpson(mid, r)$
- 一般来说面积题都能用自适应辛普森积分来做。
- 像什么圆的面积并已经沦落成它的模板题了。
- 值得一提的是：这种算法对所有由不超过三次的函数组合而成的函数积分求值都是精确值。

Application In 3D Geometry

- 众所周知，三维的几何问题一般处理起来要比二维棘手。
- 可以考虑利用自适应辛普森积分将三维问题降为二维问题。

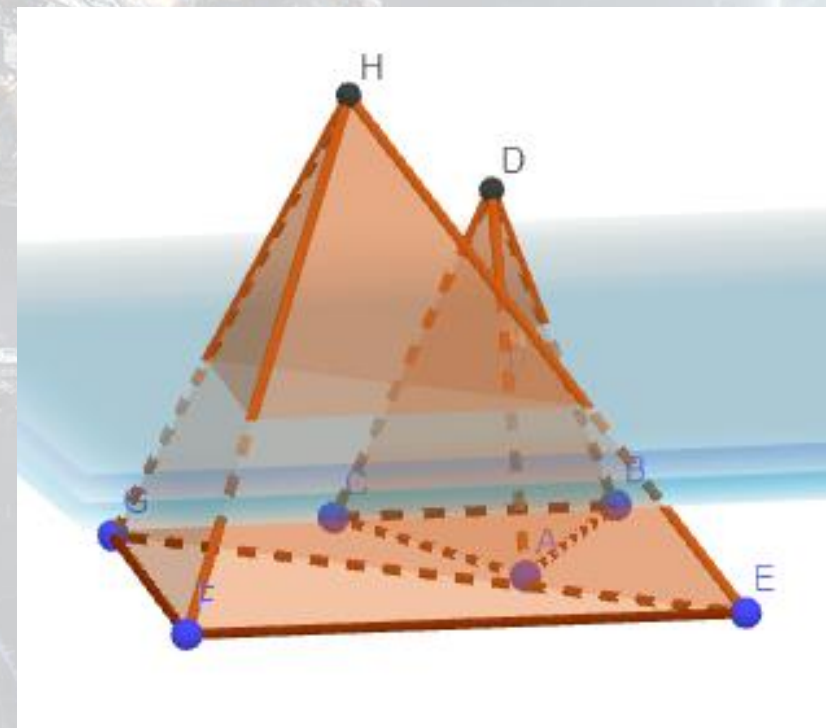
Dimensional Reduction Strike

- 就像右图那样：
- 不好意思好像走错片场了。



Transform To 2D Geometry

- 是这样：
- 令 $f(x)$ 表示截面 $z = x$ 上的答案，对 $f(x)$ 用自适应辛普森积分拟合。



Green Formula

- 自适应辛普森积分在处理封闭图形面积并问题上并不是最优秀的算法。
- 还要更强的?
- 格林公式了解一下:

$$\iint_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \oint_L P dx + Q dy$$

Randomized Algorithm

- 某一天我坐在讲台上，听到台下如下的对话：
 - A 君：“给你平面上 n 个点，求最小的圆将其全部覆盖怎么做啊？”
 - B 君：“我查一下，诶有个 $O(n)$ 的算法诶！”
 - A 君：“这么厉害？！”（马上跑过去）
 - B 君：“好像是随机算法。”
 - A 君：“切！”（十分不屑，扫兴离去）
 - 我：“……”
-
- 虚假的随机算法：纯粹看脸的比赛水法。
 - 真实的随机算法：有理有据的神仙操作。



Randomized Halfplane Intersection

- 很多时候我只是需要判断某半平面交是否有解，亦或是得到一个可行解，也可能是沿着某一个方向最远的解（即二维的线性规划问题）。
- 难道我需要大动干戈搞一发排序增量法？
- 有没有更加简单而且快速的算法？

Randomized Halfplane Intersection

- 现在我们将问题形象化描述：
- 给定若干个半平面，要找出半平面交区域中与目标函数向量 $\vec{c} = (c_x, c_y)$ 点积最大的点的坐标。如果半平面交为空也要判断出来。
- 当然了，无界的半平面交区域可能在目标函数的方向上延伸到无穷远处。我们人为地添加两个位于无穷远处的互相垂直的半平面来约束就好了。

Incremental Method

- 考虑使用增量法求出这个点。假设我们已经加入了一些半平面，然后求出了当前的最优点 v 。
- 那么我们加入一个新的半平面 h 会发生什么呢？显然这个最优点要么不变（最优点在新加入的半平面内），要么在新加入的半平面的边界上（最优点在新加入的半平面外）。这个很容易直观地感受，为了不影响学习体验，严谨的证明放在本节的最后面。
- 有这个结论怎么做呢？我们只需要考虑第二种情况，既然最优点一定在这条边界直线上，那么问题就转化成在一个一维的线性规划问题了。我们线性地扫一遍之前加入的半平面就能够算出来了。

Time Complexity

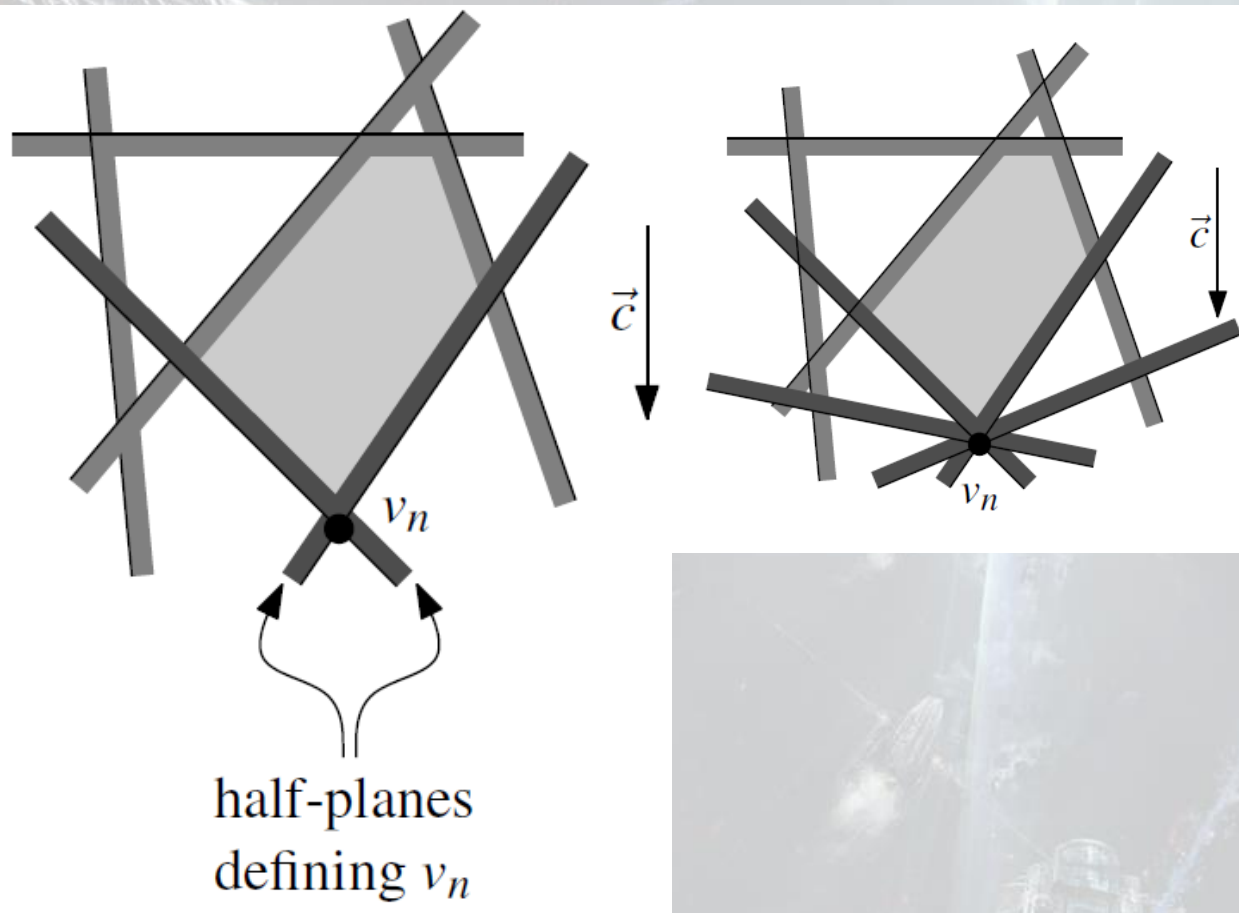
- 可是这样最坏情况下可能被卡成 $O(n^2)$ 的。
- 怎么办？我们随机增量！即将所有的半平面随机洗牌之后依次加入。
- 时间复杂度怎么分析呢？利用期望的线性性，我们设 P_i 表示执行第 i 步之后最优点发生变化的概率。那么就有时间复杂度为

$$\sum_{i=1}^n O(i)P_i$$

- 这个看起来很难正向分析，考虑反向分析，既然加入第 i 个半平面最优点会变动，那么抽出第 i 个半平面最优点也会变动。既然如此，第 i 个半平面一定是参与确定最优点位置的半平面之一。

Time Complexity

- 可以看到，确定点 v 的半平面有以下几种情况：
- 两个半平面确定了 v ，此时抽掉任一个都会使得最优解点变化。此时概率至多为 $\frac{2}{i}$ 。
- 很多个半平面确定了 v ，此时抽掉任一个都无法使最优解点变化。概率为 0。
- 综上，对于所有情况， $P_i \leq \frac{2}{i}$ 。
- 结合上一页，时间复杂度就是 $O(n)$ 的。



Minimum Covering Circle Problem

- 给定平面上 n 个点所组成的一个集合 P , 找出 P 的最小包围圆。

Incremental Method

- 设计增量法的关键就是要考察一步步引入条件对答案的影响。
- 这里我们需要用到一个性质：令 $P_i = \{p_1, p_2, \dots, p_i\}$, D_i 为相对于 P_i 的最小包围圆。
- 设 $2 < i < n$, 一定有
 - 若 $p_i \in D_{i-1}$, 则 $D_i = D_{i-1}$ 。
 - 若 $p_i \notin D_{i-1}$, 则 p_i 必然落在 D_i 的边界上。
- 和上一节类似, 为了学习体验我们将证明放在最后面。
- 考虑当前的最小包围圆为 C , 我们需要引入 p_i 。
 - 如果 $p_i \in C$, 那么我们什么都不需要做。
 - 如果 $p_i \notin C$, 那么我们就需要对已经加入的所有圆重新计算最小包围圆。

Incremental Method

- 考虑当前的最小包围圆为 C ，我们需要引入 p_i 。
 - 如果 $p_i \in C$ ，那么我们什么都不需要做。
 - 如果 $p_i \notin C$ ，那么我们就需要对已经加入的所有圆重新计算最小包围圆。
- 先记 P_1 表示这个 p_i 。
- 已知信息： P_1 一定在这个圆的边界上。怎么重新计算这个最小包围圆呢？

Incremental Method

- 故技重施，重头开始引入所有的点。
- 注意现在我們要求的是严格经过 P_1 的最小包围圆，可以证明，在加了强制经过若干点的限制下，之前的性质依然成立（同样见文末证明部分）。
- 令 C 为当前最小包围圆（初始时圆心为 P_1 ，半径为 0）。
- 然后我们一直加入点直到我们遇到了一个 $p_j \notin C$ ，又要对已经加入的所有圆重新计算最小包围圆了。
- 令 P_2 表示 p_j ，我们已经能够确定两个点 P_1 和 P_2 了。同理，我们再执行第三层，初始是一个以 P_1 和 P_2 为直径的圆，出现矛盾的时候就能够确定 3 个点，这已经足够我们确定一个圆了。

Time Complexity

- 和半平面交一样，如果我们将所有圆随机洗牌再依次加入，对每一个阶段都倒着分析，可以发现每一个阶段的期望时间复杂度都是线性的。
- 一个看起来是 $O(n^3)$ 的大暴力在随机增量下变成了期望 $O(n)$ 的。

王叔球场

- 二维平面上有 n 个关键点 (x_i, y_i) 。
- 现在你要确定一个圆的半径 r 和位置。这个圆必须平行于二维平面，其所在平面的高度可以为 $0 \dots H$ 中的一个整数。
- 当圆的高度为 i 时，它会带来 $r \times p_i$ 的代价。同时，所有关键到圆的最短距离都不能超过 D ， r 不能小于给定的半径 R 。
- 求满足条件的最小代价。
- $1 \leq n, H \leq 5 \times 10^5$

王权球场

- 高度为 h 时，距离 (x_i, y_i) 小于等于 D 的点组成一个半径为 $\sqrt{D^2 - h^2}$ 的圆，我们称这个值为 **合法半径**。
- 考虑二分答案，那么判定可以转化为将二分值加在半径上，判断圆是否有公共交点。
- 这样无法通过此题。反过来考虑，将 **合法半径** 以及答案圆半径加起来当成替代圆的半径，最小化答案圆半径等价于最小化替代圆的半径，替代圆必须覆盖所有点。
- 那么这个替代圆不管高度是多少都是一样的，就是点集的最小覆盖圆。
- 求出覆盖圆后枚举高度，判断半径是否合法并更新答案即可。
- 时间复杂度 $O(n)$ 。

Relevant Proofs

Randomized Halfplane Intersection

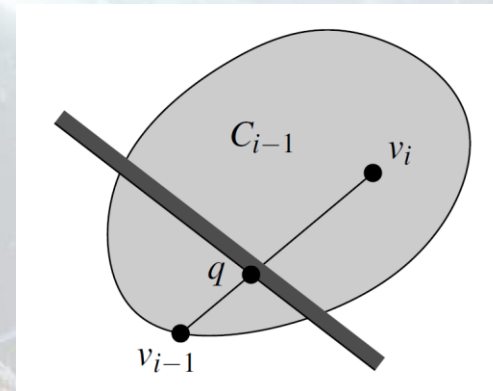
- 设 $1 \leq i \leq n$, v_i 为加入 i 张半平面时的最优点, 一定有
 - 若 $v_{i-1} \in h_i$, 则 $v_i = v_{i-1}$
 - 若 $v_{i-1} \notin h_i$, 要么 $C_i = \emptyset$, 要么 $v_i \in l_i$ (其中 l_i 就是 h_i 的边界线)

Relevant Proofs

Randomized Halfplane Intersection

- 证明:

- 设 $v_{i-1} \in h_i$, 因为 $C_i = C_{i-1} \cap h_i$, 且 $v_{i-1} \in C_{i-1}$, 所以必定有 $v_{i-1} \in C_i$, 此外, 既然 $C_i \subseteq C_{i-1}$, 故与原先 C_{i-1} 的最优解顶点相比, C_i 的最优解顶点不可能更优。因此, v_{i-1} 必然还是 C_i 的最优解顶点
- 设 $v_{i-1} \notin h_i$, 假设上面的定理不成立, 也就是说 C_i 不为空, 同时 v_i 也不落在 l_i 上
- 如右上图所示, 考察线段 $\overline{v_i v_{i-1}}$, 首先, 已知 $v_{i-1} \in C_{i-1}$; 既然 $C_i \subseteq C_{i-1}$, 故必有 $v_i \in C_{i-1}$ 。考虑到 C_{i-1} 的凸性, 线段 $\overline{v_i v_{i-1}}$ 必然整体包含于 C_{i-1} 之中。因为 v_{i-1} 是 C_{i-1} 的最优解点, 并且目标函数 $f_{\vec{c}}$ 是线性的, 所以在沿着 $\overline{v_i v_{i-1}}$ 从 v_i 到 v_{i-1} 的运动过程中, $f_{\vec{c}}(p)$ 必然是单调递增的。
- 再考虑 $\overline{v_i v_{i-1}}$ 与 l_i 的交点 q , 既然 $v_{i-1} \notin h_i$ 而 $v_i \in C_i$, 故这个交点一定存在。根据前面的分析, 线段 $\overline{v_i v_{i-1}}$ 包含于 C_{i-1} 中, 因此点 q 必然属于 C_i , 由于目标函数沿着 $\overline{v_i v_{i-1}}$ 是单调递增的, 所以 $f_{\vec{c}}(q) > f_{\vec{c}}(v_i)$, 这与 v_i 定义不符。



Relevant Proofs

Minimum Covering Circle Problem

- 设 $2 < i < n$, 一定有:
 - 若 $p_i \in D_{i-1}$, 则 $D_i = D_{i-1}$
 - 若 $p_i \notin D_{i-1}$, 则 p_i 必然落在 D_i 的边界上。
- 我们来看下面一个显然能够推导出上面结论的引理:
- 设 P 为平面上的一个点集: 设 R 为另一个 (允许为空的) 点集, 而且 $P \cap R = \emptyset$; 设点 $p \in P$. 则有下列命题成立:
 - 如果存在某个圆覆盖了 P , 而且其边界穿过 R 中的所有点, 那么这样的圆中必然存在唯一的最小者, 记作 $md(P, R)$ 。
 - 如果 $p \in md(P \setminus \{p\}, R)$, 那么 $md(P, R) = md(P \setminus \{p\}, R)$ 。
 - 如果 $p \notin md(P \setminus \{p\}, R)$, 那么 $md(P, R) = md(P \setminus \{p\}, R \cup \{p\})$ 。

Relevant Proofs

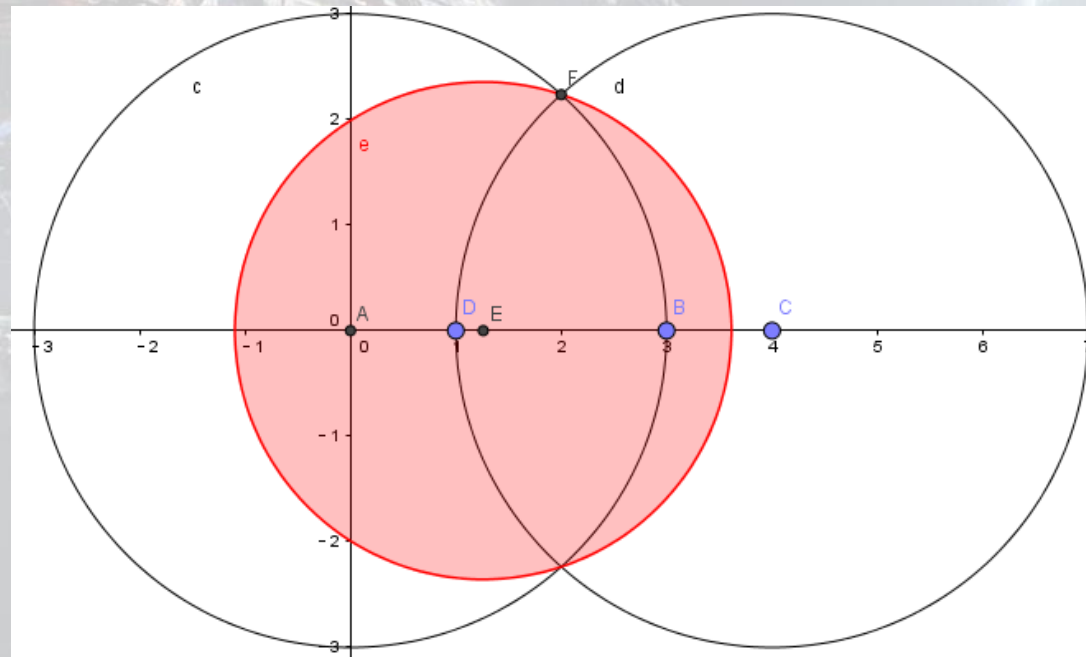
Minimum Covering Circle Problem

- 引理一:
- 假设存在半径相同的两个不同的包围圆 D_0 和 D_1 , 其圆心分别为 O_0 和 O_1 。显然, P 中所有的点必然落在交集 $D_0 \cap D_1$ 中, 且 $R \in \partial D_0 \cap \partial D_1$ 。
- 考虑使用以下方法来构造一系列的圆 $D_\lambda (0 \leq \lambda \leq 1)$:
- 取 D_0 和 D_1 的边界 (后面我会写成 ∂D_0 和 ∂D_1) 的一个交点 Q 。
- D_λ 的圆心是 $O(\lambda) = (1 - \lambda)O_0 + \lambda O_1$, 半径是 $r(\lambda) = \text{dist}(O(\lambda), Q)$ 。

Relevant Proofs

Minimum Covering Circle Problem

- $\forall \lambda \in [0,1]$, 都有 $D_0 \cap D_1 \subset D(\lambda)$
- 因此作为一个特例, 显然 $\lambda = \frac{1}{2}$ 的时候肯定成立。
- 并且可以发现 ∂D_λ 一定经过 ∂D_0 和 ∂D_1 的两个交点。
- 那么由上面种种性质可得 $D\left(\frac{1}{2}\right)$ 不仅是 P 的一个包围圆, 而且边界经过了 R 中的所有点。
- 并且 $r(\lambda)$ 显然小于 $r(1)$, 因此 D_0 和 D_1 肯定不是 $md(P,R)$, 出现矛盾。



Relevant Proofs

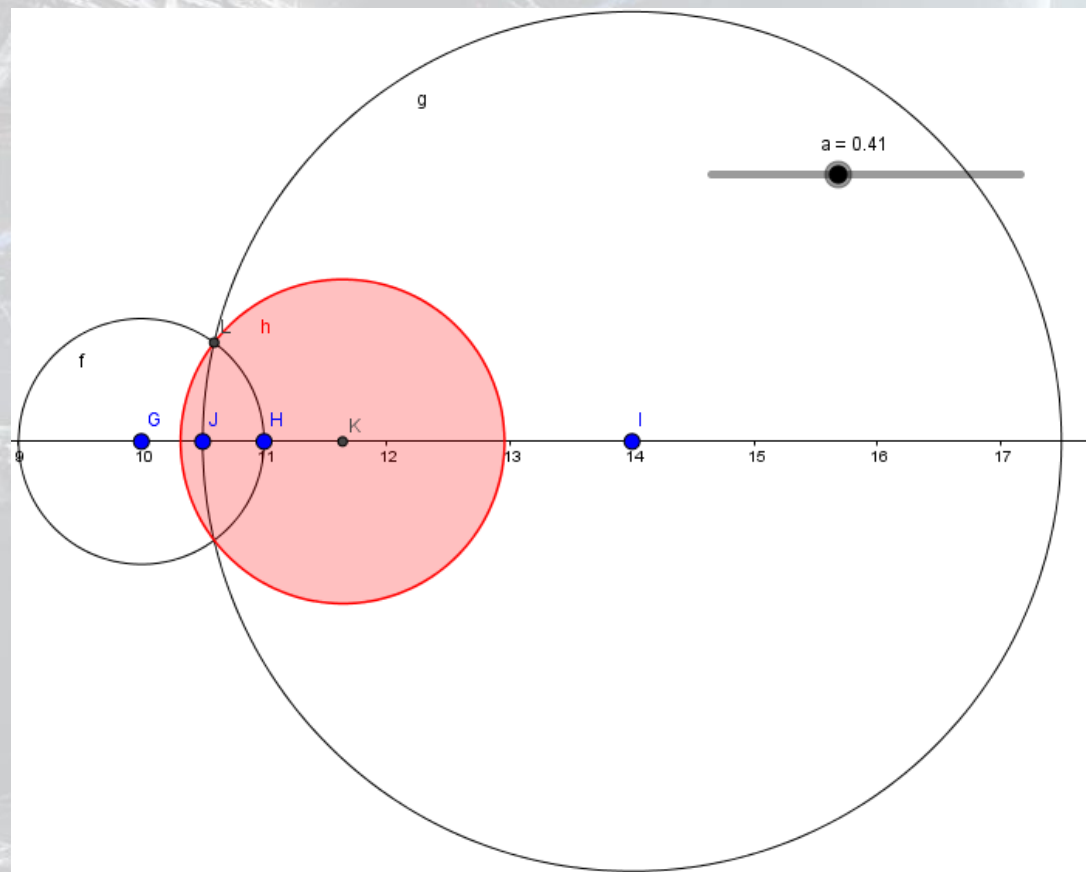
Minimum Covering Circle Problem

- 引理二:
- 令 $D = md(P \setminus \{p\}, R)$, 若 $p \in D$, 则 D 必然包括 P , 而且其边界穿过 R 中的所有点。不可能有更小的圆可以覆盖 P , 否则这个圆一定能够覆盖 $P \setminus \{p\}$ 并经过 R , 这与 D 的定义矛盾。

Relevant Proofs

Minimum Covering Circle Problem

- 引理三:
- 令 $D_0 = md(P \setminus \{p\}, R)$, $D_1 = md(P, R)$, 再次考虑引理一证明中构造一系列圆 D_λ ($0 \leq \lambda \leq 1$) 的方法。可以发现, 这种方法实际上定义了一个由 D_0 到 D_1 的连续变换过程。
- 根据假设, 有 $p \notin D_0$, 且 $p \in D_1$, 因此一定存在 $\lambda \in (0, 1]$ 满足 $p \in \partial D_\lambda$ 。
- 并且和引理一证明同理, 有 $P \subset D_\lambda$ 且 $R \subset \partial D_\lambda$, 且 $r(\lambda) \leq r(1)$ 。
- 根据定义, $D_1 = md(P, R)$, 因此 λ 只有一种可能, 即 $\lambda = 1$, 因此 P 在 D_1 边界上。



Bear and Shuffled Points

Codechef October Challenge 2016, GEOCHEAT

- 给定平面上的 n 个点 (x_i, y_i) 。
- 对于每个 $1 \leq i \leq n$, 求出只保留前 i 个点, 平面上点对的最远距离。
- $n \leq 750000, |x_i|, |y_i| \leq 10^9$
- 数据的生成方式是: 以某种方式把所有点的坐标生成后, 把点随机排列。

<https://www.codechef.com/OCT16/problems/GEOCHEAT>

Bear and Shuffled Points

- 求一个点集的最远点对距离，是经典的旋转卡壳。
- 这题我们显然不用对于每一个 i 都去旋转卡壳：假设整个点集的最远点对是 (a, b) ，显然下一步我们只需要求 1 到 $\max(a, b)$ 的最远点对，中间的答案都是一样的。求完之后我们用同样的过程迭代下去。
- 时间复杂度？注意数据随机，那么我们将任意一个点对成为最远点对的概率视作均等。

Bear and Shuffled Points

- 先来考虑这样一个问题：在 1 至 n 的序列中随机选择两个不同的数，这两个数最大值的期望是多少？

$$\begin{aligned} E[\max(a, b)] &= \frac{\sum_{x=1}^n \sum_{y=x+1}^n y}{\binom{n}{2}} \\ &= \frac{\sum_{x=1}^n (n-x)(x+1+n)/2}{\binom{n}{2}} \\ &= \frac{2(n+1)}{3} \end{aligned}$$

Bear and Shuffled Points

- 时间复杂度:

$$\begin{aligned} T(n) &= T\left(\frac{2}{3}n\right) + O(n \log n) \\ &= O\left(\left(1 + \frac{2}{3} + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3 + \dots\right)n \log n\right) \\ &= O(n \log n) \end{aligned}$$

- 是不是很棒棒?

Dual

- 计算几何中对偶虽然出现的比较少，但是往往能极大地简化问题。

Point \times Line

- 点/线对偶是最简单的几何对偶。
- 平面上的任何一点，都具有两个参数， x 坐标和 y 坐标。平面上任何一条（非垂直的）直线，也拥有两个参数斜率 k 和截距 b 。
- 因此，可以通过某种一一对应的方式，将一组点映射为一组直线，反之亦然。
- 如果做得巧妙的话，甚至可以将原先点集所具有的某些性质，转化为直线集所具有的某些性质。

Point \times Line

- 我们来考察一种简单的对偶形式:

- $p \stackrel{\text{def}}{=} (p_x, p_y) \xrightarrow{\text{dual}} p^*: y = p_x x - p_y$

- $\ell: y = kx + b \xrightarrow{\text{dual}} \ell^* \stackrel{\text{def}}{=} (k, -b)$

- 观察可得, 这一种对偶变换满足下列性质:

- 关联性的保持: $p \in \ell$ 当且仅当 $\ell^* \in p^*$

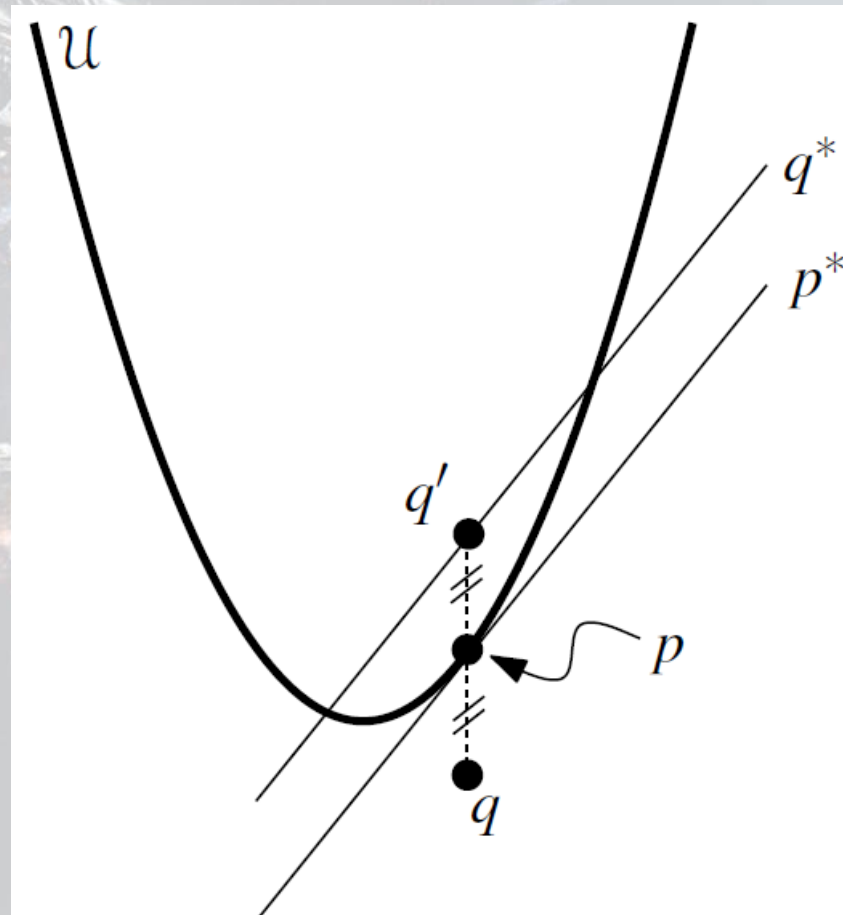
- (位置) 次序的保持: p 位于 ℓ 上方, 当且仅当 ℓ^* 位于 p^* 的上方

Point × Line

- 性质的证明:
- 现在我们有 点 (x, y) , 直线 $y = kx + b$
- $\text{sgn}(kx + b - y) = \begin{cases} -1, & p \text{ is above } \ell \\ 0, & p \text{ is on } \ell \\ 1, & p \text{ is below } \ell \end{cases}$
- 对偶之后 $\text{sgn}(xk - y + b) = \text{sgn}(kx + b - y)$

A Brilliant Geometric Explanation

- 这种对偶形式有一个漂亮的几何解释
- 现在有一个抛物线 $\mathcal{U}: y = \frac{1}{2}x^2$
- 首先来考察 \mathcal{U} 上任意一点的对偶
- 在 p 处, \mathcal{U} 的微分是 p_x , 也就是说 p^* 的斜率和在 p 点处的切线斜率相同。而事实上, 任意 $p \in \mathcal{U}$ 的对偶, 正是 \mathcal{U} 在 p 处的切线。
- 现在再考察没有落在 \mathcal{U} 上的点 q 的对偶。位于同一条垂线上的任意两点, 其对偶的两条直线必然斜率相等。也就是说, q^* 必然与 p^* 平行。其中的 p 就是落在 \mathcal{U} 上、与 q 的 x 坐标相同的点。
- 除此之外可以发现, q^* 一定穿 q 关于 p 的对称点 q' 。因此 q^* 是过 q' 且与 p^* 平行的直线。



Lines

2015–2016 Petrozavodsk Winter Training Camp, SPb SU + SPb AU Contest, B

- 平面上有 n 个不同的点 a_1, \dots, a_n 。
- 对于每一对 $(i, j) (i < j)$, 考虑连接 a_i 和 a_j 的直线 $L_{i,j}$ 。令 $A_{i,j}$ 表示水平直线顺时针旋转到 $L_{i,j}$ 的弧度, 由定义得 $0 \leq A_{i,j} < \pi$ 。
- 请求出所有的 $A_{i,j} (i < j)$ 组成的序列的中位数。
- $2 \leq n \leq 10^5$

Lines

2015–2016 Petrozavodsk Winter Training Camp, SPb SU + SPb AU Contest, B

- 直接做好像无从下手，考虑对偶。
- 点 (a, b) 变为直线 $y = ax - b$ ，那么两直线交点的 x 坐标就是原本两点连线的斜率。
- 考虑二分答案 mid 。
- 我们要求 $(-\infty, mid]$ 的交点个数，这个逆序对搞一搞就好了。
- 时间复杂度 $O(n \log^2 n)$ 。

Convex Hull × Halfplane Intersection

- 注意到凸包和半平面交之间的关系与点和直线的关系类似。
- 具体如何在两者之间反演留作课后思考。

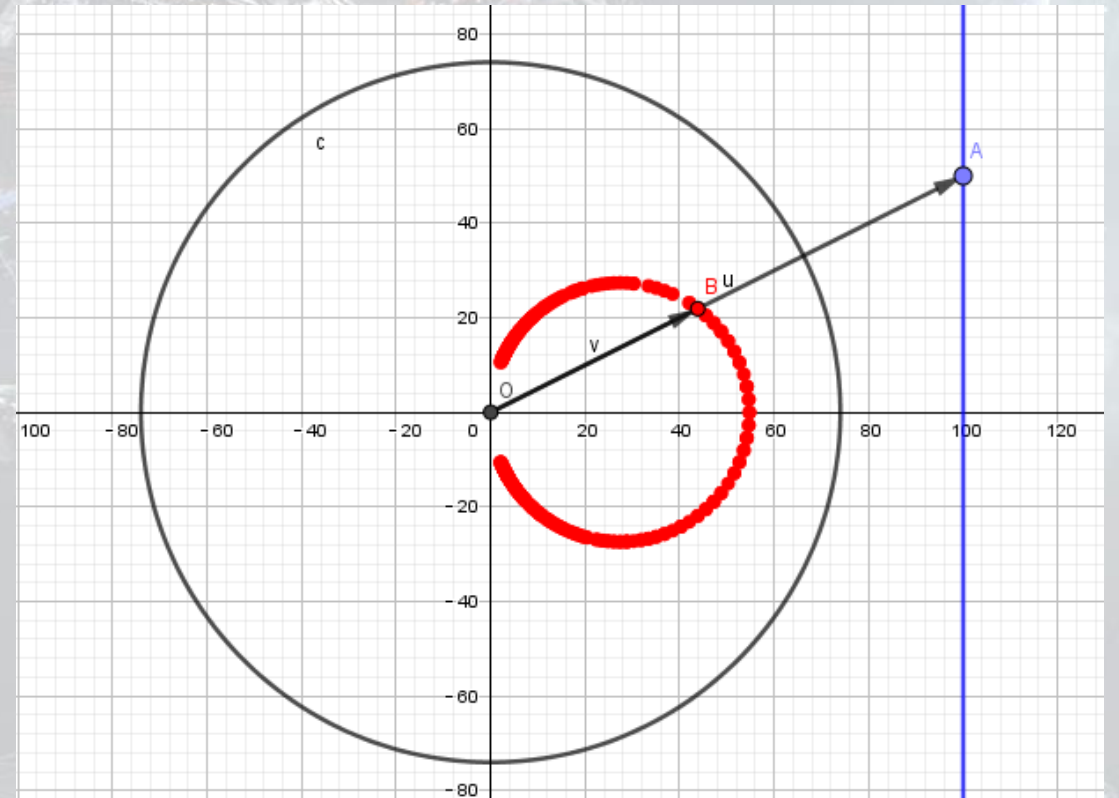
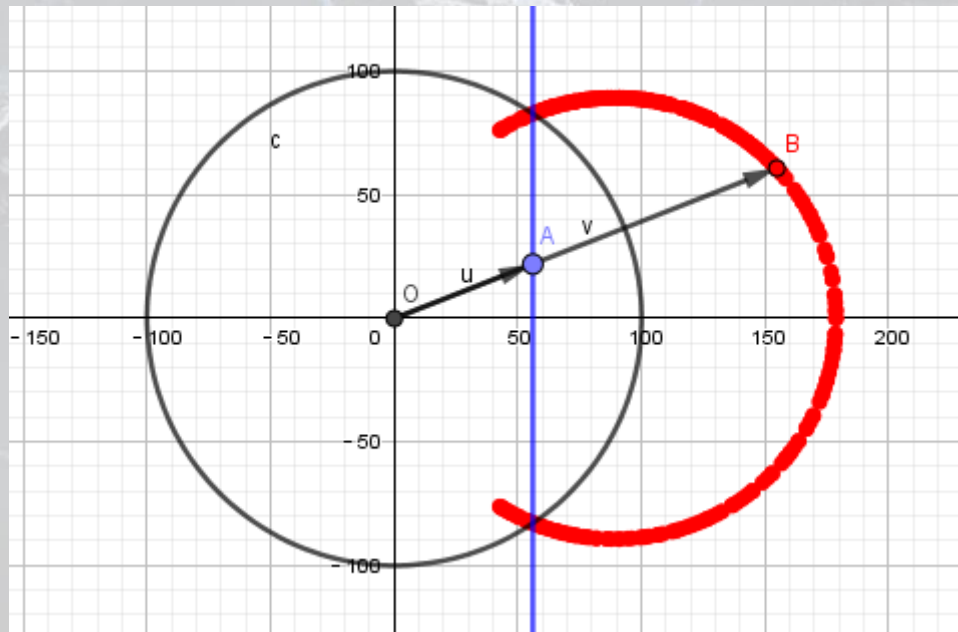
Circle Inversion

- 圆的反演是处理圆与直线/圆相切问题的利器。
- 已知圆 C ，圆心为 O ，半径为 r 。如果 P 与 P' 在过圆心 O 的直线上，且 $|OP| \times |OP'| = r^2$ ，则称 P 与关于 O 互为反形。

Properties Of Circle Inversion

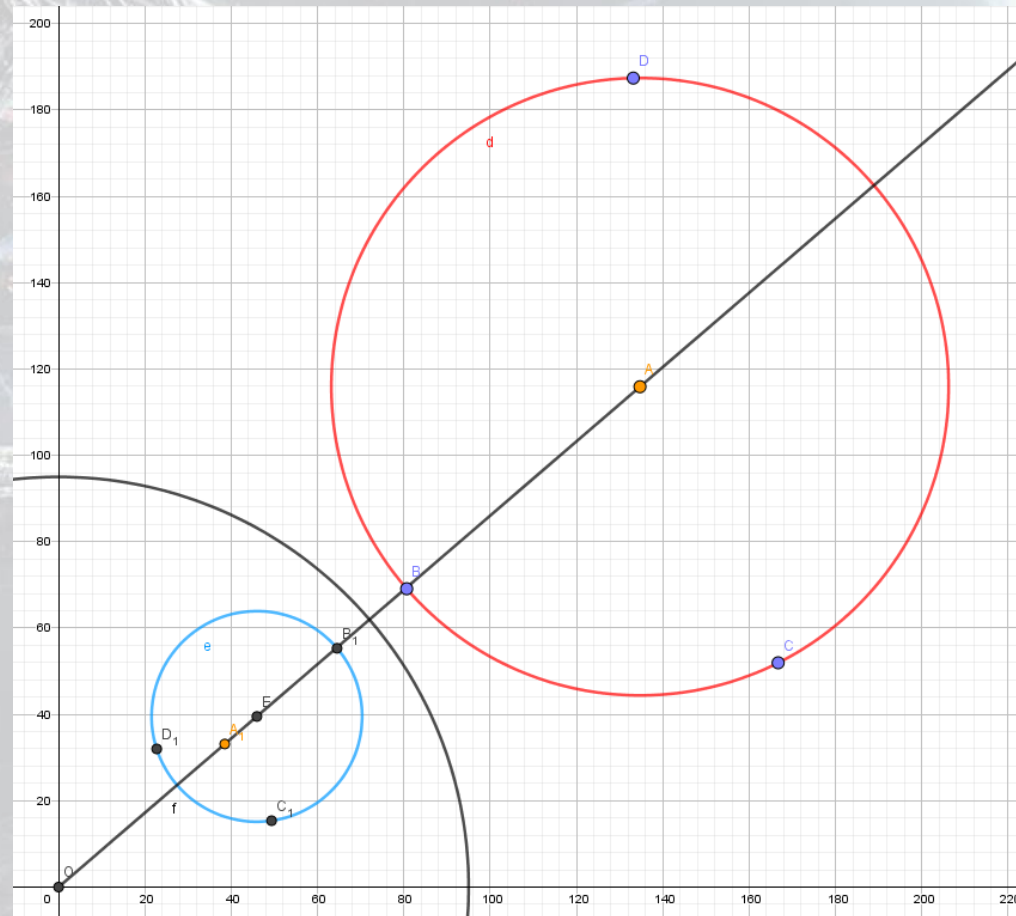
- 除了反演中心以外，平面上的每一个点都只有唯一的反演点，且这种关系是对称的。位于反演圆上的点，保持在原处，位于反演圆外部的点，变为圆内部的点，反之亦然。
- 保圆性：即原坐标系的任意圆，反演后依然是圆。
- 任意一条不过反演中心的直线，它的反形是经过反演中心的圆。反之亦然。过反演中心的直线的反形是它自己（半径为无穷大的过反演中心的圆）。

Properties Of Circle Inversion



Properties Of Circle Inversion

- 不过反演中心的圆，它的反形是一个圆，反演中心是这两个互为反形的圆的一个位似中心，任意一对反演点是逆对应点（如右图的 B 与 B_1 、 C 与 C_1 、 D 与 D_1 ）。
- 需要注意的是，两个反演圆的圆心虽然的确是在过反演中心的一条直线上，但是它们并不互为反形。



Calculation Of Circle Inversion

- 我们不能直接通过反演圆心来求反演圆圆心，那么有什么正确而又简洁有效的方法求反演圆呢？
- 设圆 C_1 圆心为 O_1 半径为 r_1 ，反演圆 C_2 同理，反演半径为 r 。
- 我们依然考察两个圆圆心的射线，假定这条射线过 C_1 上的 A 和 B 点，两点反形（显然在这条射线上）为 A' 和 B' 。
- 根据定义：
 - $|OA||OA'| = (|OO_1| + r_1)(|OO_2 - r_2|) = r^2$
 - $|OB||OB'| = (|OO_1| - r_1)(|OO_2 + r_2|) = r^2$
- 变形之后，得到 $r_2 = \frac{1}{2} \left(\frac{1}{|OO_1| - r_1} - \frac{1}{|OO_1| + r_1} \right) r^2$

Calculation Of Circle Inversion

- 知道了反演圆半径，如何求反演圆圆心？
- 设 $O(x_0, y_0), O_1(x_1, y_1), O_2(x_2, y_2)$
- 显然
 - $x_2 = x_0 + \frac{|OO_2|}{|OO_1|}(x_1 - x_0)$
 - $y_2 = y_0 + \frac{|OO_2|}{|OO_1|}(y_1 - y_0)$
- 这个 $|OO_2|$ 咋算呢？之前一页的方程里面代入 r_2 就好了。

Problem of Apollonius

2013 Asia Hangzhou Regional Contest, D

- 给定两个相离的圆，在这两个圆外给定一个点 P 。
- 求过点 P 且与已知的两个圆外切的所有圆的总数，它们的圆心坐标，半径。

Problem of Apollonius

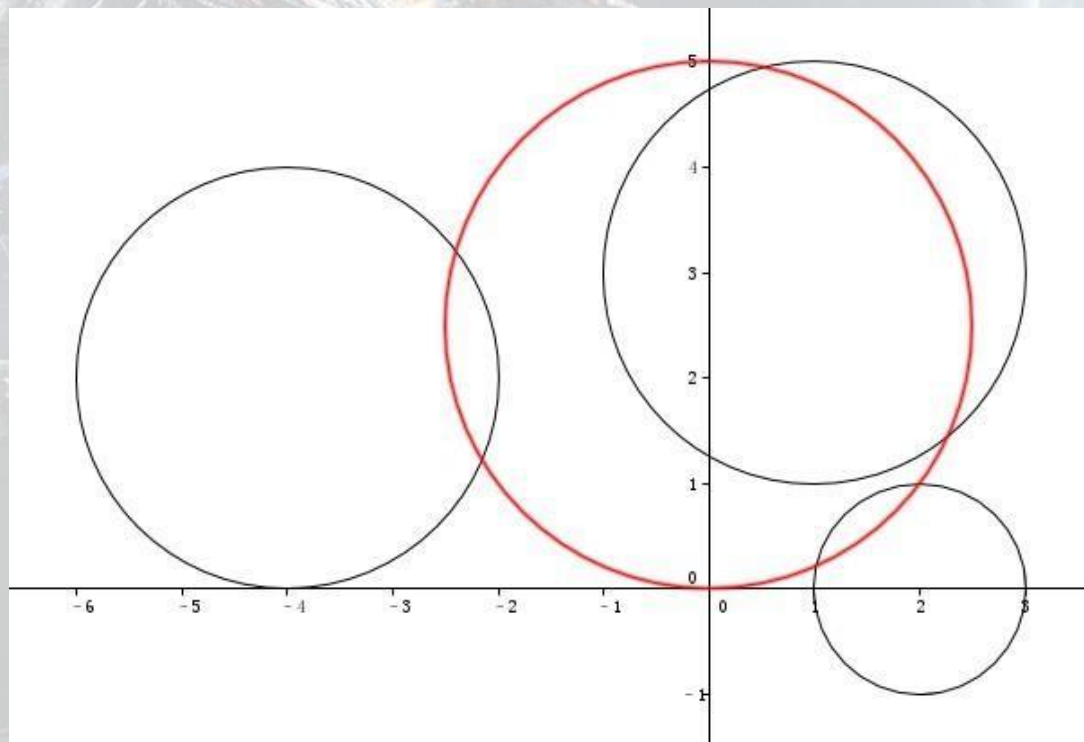
2013 Asia Hangzhou Regional Contest, D

- 由于本题的做法是这样的，以点 P 为反演中心，反演半径随便设置都可以，为了计算方便就设为 1，把圆 C_1 和圆 C_2 反演后再求这两个圆的公切线，再把这个公切线反演回去，那么就是一个过点 P 的圆，且与原来的 C_1 和 C_2 相切。
- 那么接下来就是如何计算两个圆的公切线了。这里只需要考虑公切线在同一侧的情况（因为是外切）。相信认真学习过计算几何基础的同学都会求圆的公切线。

环日加速器

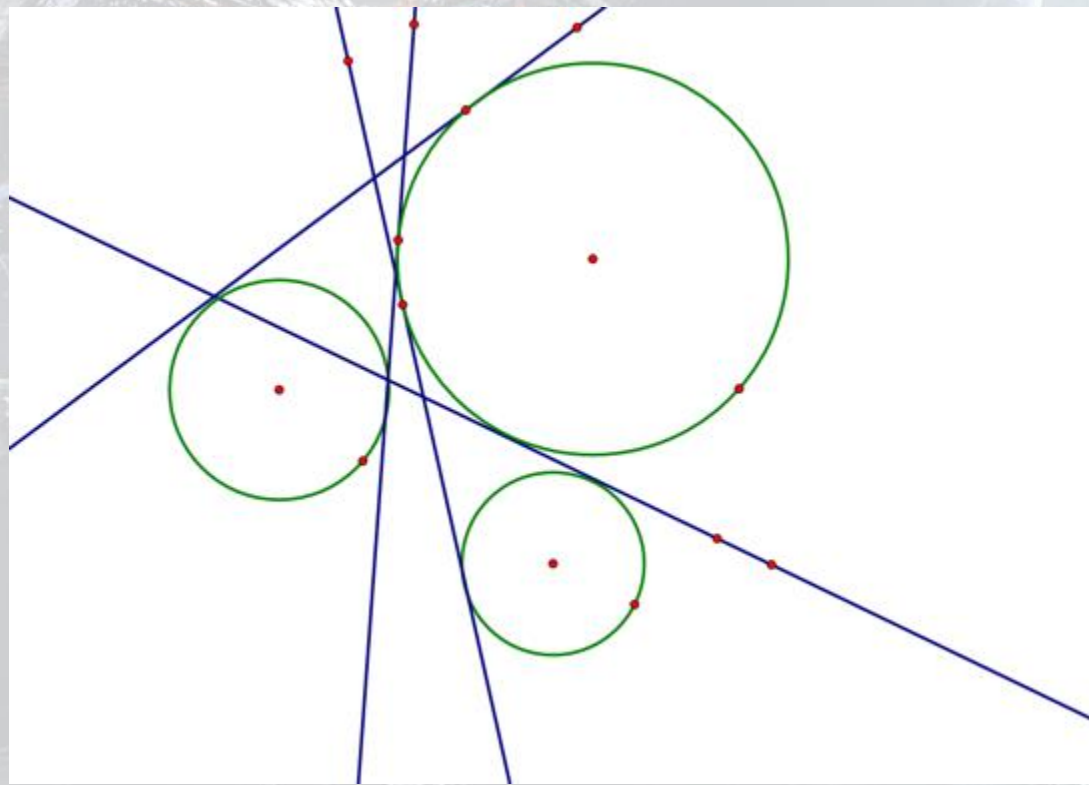
- 给定平面上 n 个不相交且不覆盖原点的圆，问过原点的圆最多经过几个圆。

- $1 \leq n \leq 10^3$



环日加速器

- 既然答案圆经过原点，那么对其反演之后就是任意一条直线。
- 问题变成求一条直线穿过尽量多的圆。显然这条直线一定可以调整成两个圆的公切线。
- 考虑枚举其中一个圆。可以发现，对于一个中心圆上的若干条关键切线，按照它在圆上的分布顺序排序，可以得出当普通切线切点在两个点之间的时候这条切线才会与另一个圆相交（切）。这样，我们可以做一些类似关键点的东西，然后按顺序扫一遍就行了。
- 时间复杂度 $O(n^2 \log n)$ 。



Drawing Circles is Fun

Codeforces Round #219 (Div. 1), E

- 平面上有 n 个点，第 i 个点的坐标是 (x_i, y_i) ，没有点在原点上。
 - 你要找点对集 $(P_1, P_2), (P_3, P_4), \dots, (P_{2k-1}, P_{2k})$ 。要求满足：
 - $k \geq 2$ 。
 - P_i 互不相同，并且都是输入的。
 - 对于任意 $(P_{2i-1}, P_{2i}), (P_{2j-1}, P_{2j})$ ， $\triangle OP_{2i-1}P_{2j-1}$ 的外接圆与 $\triangle OP_{2i}P_{2j}$ 的外接圆只有一个交点。
 $\triangle OP_{2i-1}P_{2j}$ 的外接圆与 $\triangle OP_{2i}P_{2j-1}$ 的外接圆只有一个交点。
 - 求这样的点对集的个数。
-
- $1 \leq n \leq 10^3$
 - 不存在两点重合。

<https://codeforces.com/contest/372/problem/E>

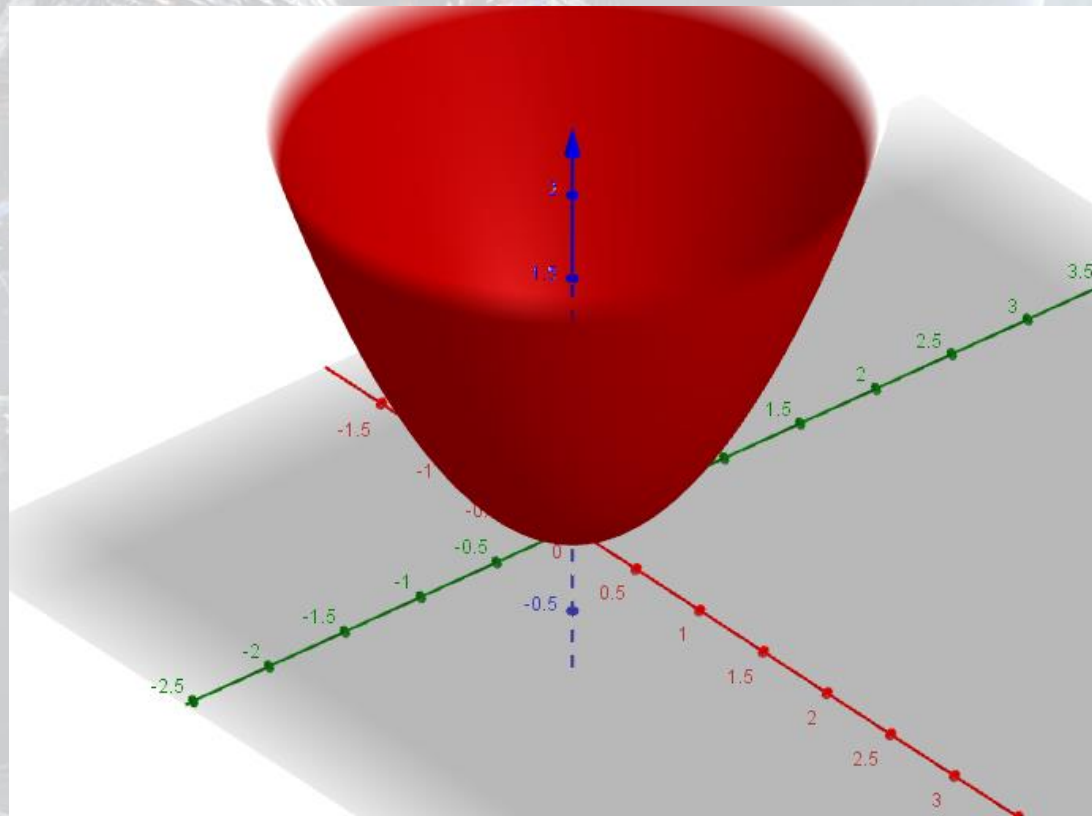
Drawing Circles is Fun

Codeforces Round #219 (Div. 1), E

- 首先我们肯定要把所有点先以原点为中心反演，两个过原点的圆只有一个交点，即反演之后的两条直线平行（在无穷远处有一个交点）。
- 然后那个点对之间外接圆的条件就直接变成了要求四个点围成一个平行四边形。
- 考虑枚举 n^2 条线段计算出中点，显然只有中点相同，**而且斜率不同**的线段能够围成平行四边形。
- 而且每一个点对集一定是一堆中点相同的线段，考虑对中点相同的线段按照斜率分组，然后每组最多只能选择一个点对。简单计数一下就好了。
- 时间复杂度 $O(n^2 \log n)$ 。

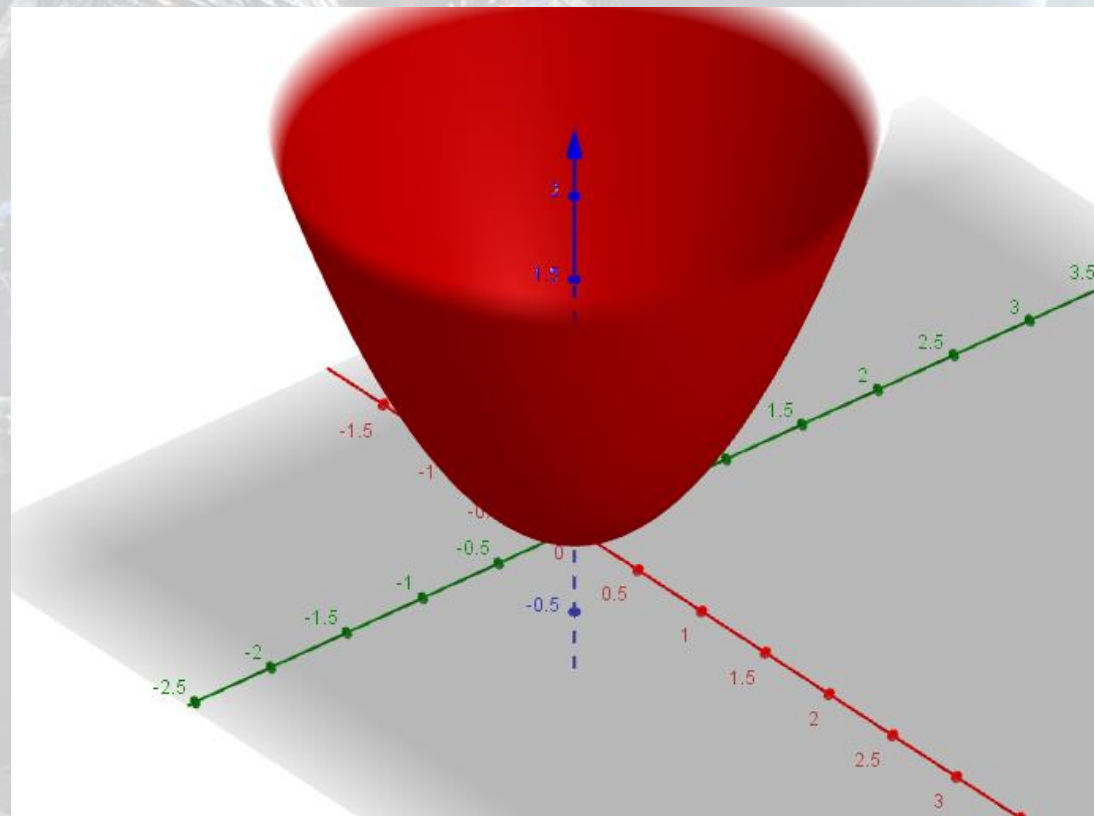
Circle \times Paraboloid

- 考虑二维平面上的一个点 (x, y) 。
- 我们把它投影到一个三维的单位抛物面 $x^2 + y^2 = z$ (大概就是右图这玩意儿) 上。
- $(x, y) \xrightarrow{\text{dual}} (x, y, x^2 + y^2)$



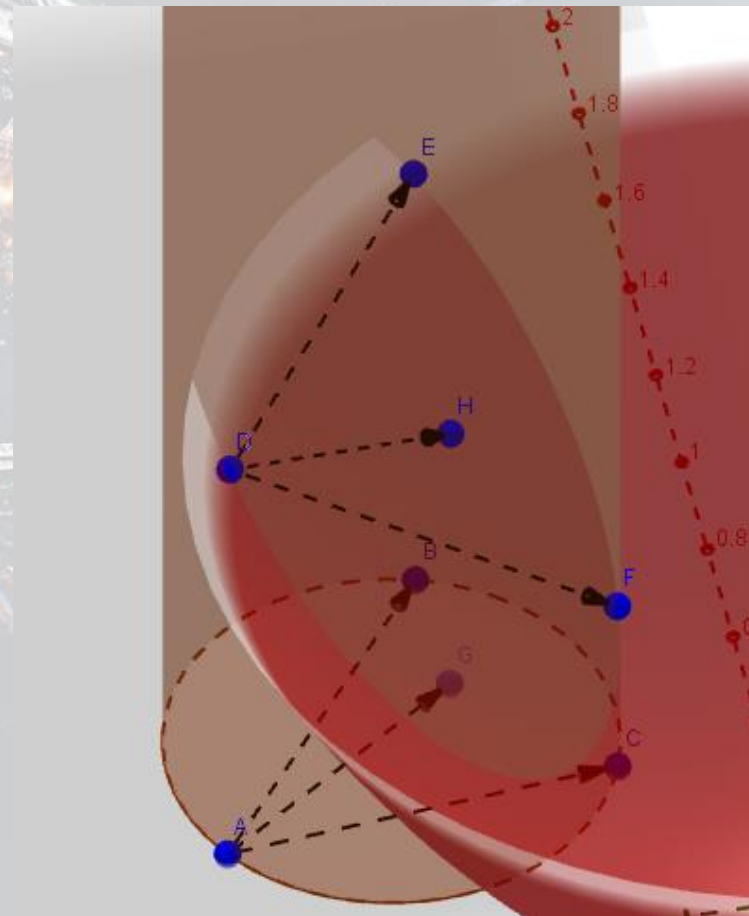
Circle \times Paraboloid

- 我们来看看一个圆投影到上面变成什么。
- 可以看见，圆投影到上面是一堆连续的点
- 并且这些点看起来在同一个平面上
- 我们来试着证明一下：
 - $(x - a)^2 + (y - b)^2 = r^2$
 - $x^2 + y^2 - 2ax - 2by + a^2 + b^2 = r^2$
 - $z = x^2 + y^2 = 2ax + 2by + r^2 - a^2 - b^2$
- 显然是一个平面的方程。



Circle \times Paraboloid

- 根据证明处的式子化得，二维平面上一个点在圆内当且仅当其对偶点在对偶平面下方（点在圆上、外的情况类似）。
- 具体怎么判断呢？以点在圆内为例，使用三维向量的方法。
- 首先我们要保证在二维平面上， $\overrightarrow{p_1 p_2}$ 在 $\overrightarrow{p_1 p_3}$ 的逆时针方向
- 那么在三维空间内， $\overrightarrow{p_1^* p_2^*}$ 、 $\overrightarrow{p_1^* p_3^*}$ 、 $\overrightarrow{p_1^* q^*}$ 成右手系。
- 使用行列式判断一下就好了。



Circle \times Paraboloid

- 使用这个方法可以简化点在三点圆内的判断。
- 同时也有部分毒瘤题可能会用到这一种反演。
- 例如?
- [yosuga no sora](#) from GMOJ

Complex Algorithm

什么？你居然看到这里了！
还没被劝退？



3D Convex Hull

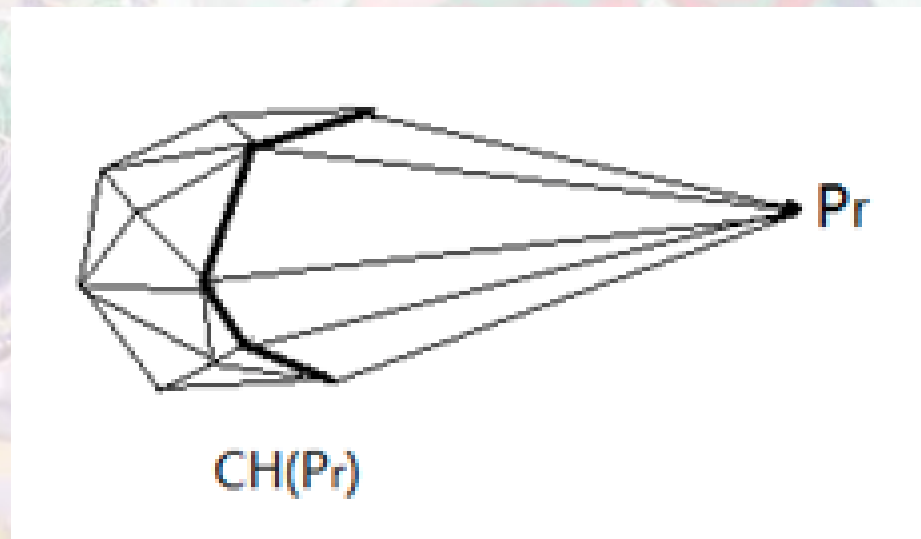
- 如果你真的遇到了，那大概是活久见吧……
- 三维凸包的规模： $O(n)$
- 如何证明？欧拉公式！
- 对于多面体/平面图： $|V| - |E| + |F| = 2$ 。
- 每个面由至少三条边组成，每条边都和两个面相连： $2|E| \geq 3|F|$ 。
- 两式组合化得 $|F| \leq 2|V| - 4, |E| \leq 3|V| - 6$ 。

Brute Force

- 枚举所有平面三角形，判断是否所有点都在其某一侧。
- 时间复杂度 $O(n^4)$ 。

Incremental Method

- 这是一个 $O(n^2)$ 的优秀算法，且实现简单，C++ 代码在 60 行以内，可谓短小精悍。
- 考虑先选出三个点构成一个空间三角形，作为初始凸包。
- 然后一个个点加入凸包中，如果其在凸包内最直接无视；否则将所有外侧处于这个点可见范围的面全部删除，然后遍历所有剩余的边，如果其相邻两个面分别为可见和不可见，就将其两个端点与新加入点组成的空间三角形作为一个面加入到凸包中。
- 注意到我们没有考虑四点共面，因此要在算法的最开始加上随机微小扰动。



Randomized Incremental Method

- 有一种利用二分图等结构储存面的牛逼实现。
- 如果再加上随机化的话，时间复杂度可以达到惊人的 $O(n \log n)$ 。
- 这种算法在算法竞赛中大概是一辈子都用不上的了。
- 不过这一节的算法大概都是这样的吧（小声）。

Voronoi Diagram

- 点集 P 对应的 Voronoi 图，就是平面的一个子区域划分。
- 整个平面被划分为 n 个单元，且具有如下性质：
- 对于位于点 p_i 所对应的单元中的任意一点 q ， $\forall p_j \in P, j \neq i$ ，都有 $\text{dist}(q, p_i) < \text{dist}(q, p_j)$ 。

Conventions

- $Vor(P)$: P 所对应的 Voronoi 图。
- $V(p_i)$: p_i 相对应的 Voronoi 单元。
- $h(p, q)$: 点 p 和点 q 的垂直平分线确定的两张半平面中, 点 p 所在的那张。

Some Evident Conclusions

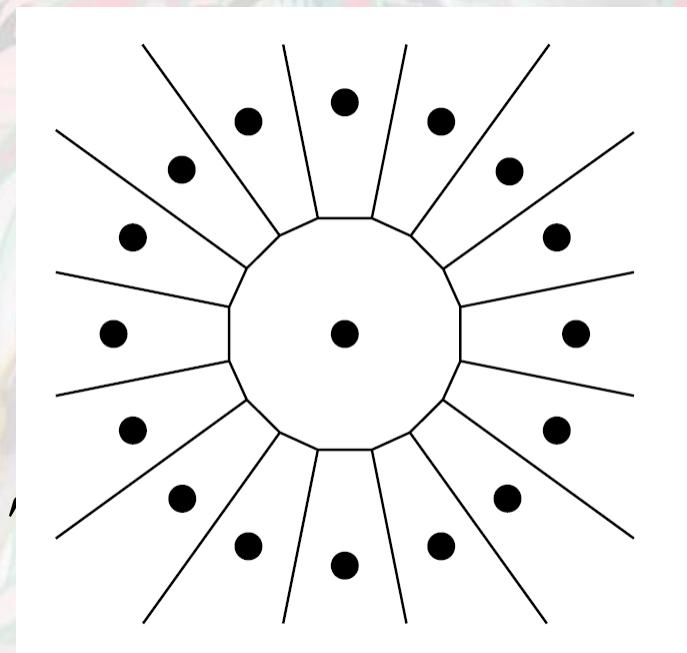
- 结论 1:

$$V(p_i) = \bigcap_{1 \leq j \leq n, j \neq i} h(p_i, p_j)$$

- 结论 2:
- 给定平面上任意 n 个点构成的集合 P 。
- 若所有点都共线，则 $Vor(P)$ 由 $n - 1$ 条平行直线构成；
- 否则， $Vor(P)$ 将是连通的，而且其中的边不是线段就是半直线。

Complexity Of Voronoi Diagram

- 显然 $Vor(P)$ 中单个单元的复杂度可能达到 $O(n)$ 。
- 喵喵喵???
- 不虚不虚，它的总规模依然是 $O(n)$ 的！
- 定理：
- 若 $n \geq 3$ ，则在与平面上任意 n 个点的 Voronoi 图中，顶点的数目不会超过 $2n - 5$ ，而且边的数目不会超过 $3n - 6$ 。
- 证明？欧拉公式。



A Significant Theorem

- 对于任一点集 P 所对应的 Voronoi 图 $Vor(P)$, 下列命题成立:
 - 点 q 是 $Vor(P)$ 的一个顶点, 当且仅当在其最大空圆 $C_P(q)$ 的边界上, 至少有三个点。
 - p_i 和 p_j 之间的垂直平分线确定了 $Vor(P)$ 的一条边, 当且仅当在这条线上存在一个点 q , $C_P(q)$ 的边界经过 p_i 和 p_j , 但不经过其它点。

A Significant Theorem

- 证明一:
 - 假设存在某个点 q , $C_P(q)$ 的边界经过至少三个点。从中选出三个点 p_i 、 p_j 和 p_k 。既然 $C_P(q)$ 的内部是空的, 故 q 必然同时落在 $V(p_i)$ 、 $V(p_j)$ 和 $V(p_k)$ 的边界上, q 肯定是 $Vor(P)$ 的一个顶点。
 - 反之, $Vor(P)$ 中的每个顶点都与至少三条边相关联, 因此也至少与三个 Voronoi 单元相关联, 不妨设这三个单元分别为 $V(p_i)$ 、 $V(p_j)$ 和 $V(p_k)$ 。顶点 q 到 p_i 、 p_j 和 p_k 的距离相等, 而且到其他各点的距离都不可能更近, 否则这三个单元的边界就不可能相交于 q 。于是, 边界由 p_i 、 p_j 和 p_k 确定的这个圆, 内部不可能包含任何点。

A Significant Theorem

- 证明二:
 - 假设存在某个点 q 具有本定理的性质, 既然 $C_P(q)$ 的内部不含任何点, 且 p_i 和 p_j 落在其边界上, 故 $\forall 1 \leq k \leq n$, 都应该有 $\text{dist}(q, p_i) = \text{dist}(q, p_j) \leq \text{dist}(q, p_k)$ 。于是, q 必然落在 $\text{Vor}(P)$ 的某条边或某顶点上。而根据定理前半部分, q 不可能是 $\text{Vor}(P)$ 的一个顶点。因此, q 只能落在 $\text{Vor}(P)$ 的某条边上, 而且这条边是 p_i 和 p_j 之间的垂直平分线上的一段。
 - 反过来, 假设 p_i 和 p_j 的垂直平分线确定了一条 Voronoi 边。来自这条边内部的任何一个点 q , 其对应的最大空圆边界必然经过 p_i 和 p_j , 但不经过其它点。

Construction Of Voronoi Diagram

- *Halfplane Intersection* – $O(n^2 \log n)$
- *Incremental Method* – $O(n^2)$
- *Divide-and-conquer Algorithm* – $O(n \log n)$
 - Code length: 8.0k in Pascal
- *Fortune Algorithm* – $O(n \log n)$
 - Code length: 8.4k in C++

Fortune Algorithm

- 这是一个十分漂亮的算法。
- 按照扫描线的套路：用一条水平直线 ℓ 自上而下来扫描平面，并且尝试同时维护 Voronoi 图与当前扫描线的相交部分。
- 这个想法很好，但是实际上哪怕是 $Vor(P)$ 位于扫描线 ℓ 上方的部分，它都不止取决于 ℓ 上方的点，下面的点也会对这个图形造成影响。这对 $Vor(P)$ 形态的维护造成了很大的麻烦。
- 想想扫描线 ℓ 的上方有什么东西是已经确定的。

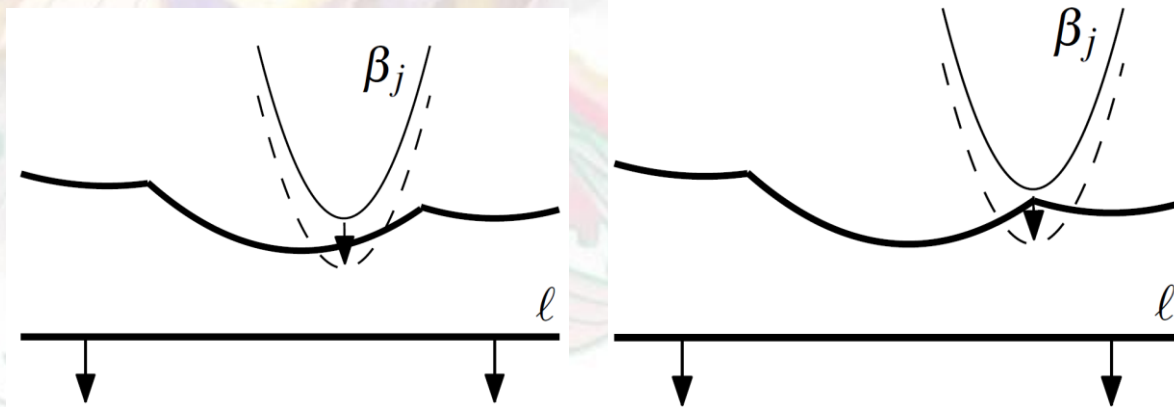
Fortune Algorithm

- ℓ 上方的若干点和 ℓ 一起定义了若干条抛物线，定义这些抛物线的下边界为海岸线。显然，海岸线在 x 方向是单调的，即它与任一垂线相交而且仅相交于一个点。
- 海岸线显然是由若干段抛物线的弧首尾依次连接而成。
- 我们称那些接合点为断点。
- 在扫描线 ℓ 自上而下扫过整个平面的过程中，所有断点的轨迹合起来，恰好就是所需要构造的 Voronoi 图（考虑抛物线的意义：到达一点距离与到达直线距离相等的点的运动轨迹）。
- 显然在 ℓ 所对应的海岸线上方部分都是已经确定的。



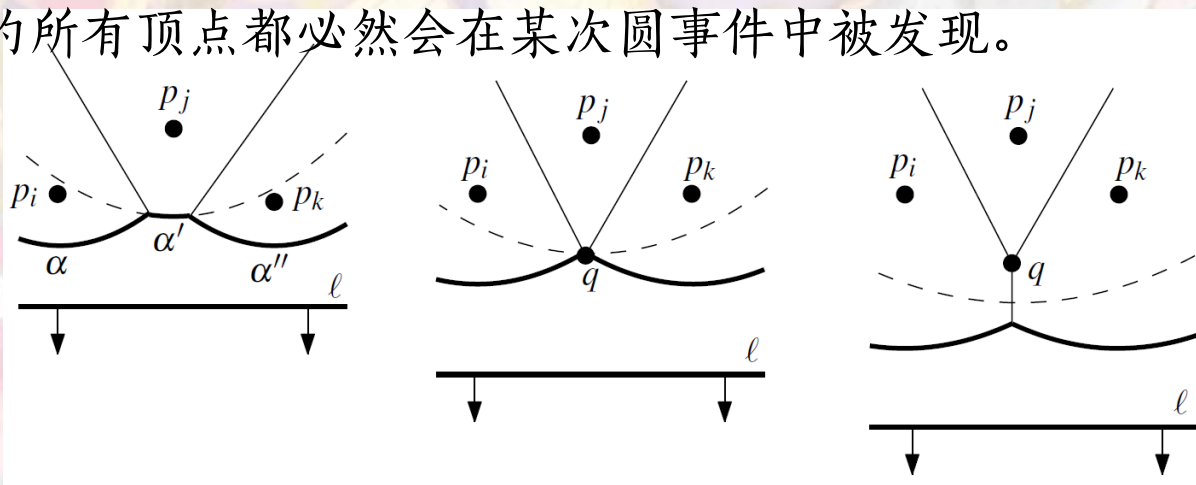
Fortune Algorithm

- 考虑在扫描中会出现什么事件？显然，当一个点出现在扫描线上时，“海岸线”的轮廓会发生变化：更确切地说，海岸线上会出现一段新的弧。
- 我们称一个点出现在扫描线上为一个“点事件”。
- 定理：只有在发生某个点事件时，海岸线上才会有新的弧出现。
- 证明：分两种情况讨论，一种是之前的某一条抛物线快速生长突破海岸线的某一条弧，一种是之前某一条抛物线刚好从海岸线的断点突破。
- 可以证明前一种不可能发生，后一种抛物线在触断点后就会缩回去（即不会新建一段弧）。



Fortune Algorithm

- 除了点事件，扫描过程中还有一种事件：原先的某段弧越缩越小变成一个点随后消失。
- 可以发现，这种事件发生时，这连续的三段弧所对应的点 p_i 、 p_j 和 p_k 三点确定的圆圆心一定是 q （中间那个退化成点的弧）。
- 我们称这样的一种事件叫做“圆事件”，这种时间必然发生在连续的三段弧中。
- 与点事件相反，其给海岸线带来的影响是其会删除海岸线上的一段弧。
- 定理：海岸线上已有的弧，只有在经过某次圆事件之后，才有可能消失。
- 更进一步：这个退化的弧 q 一定是 Voronoi 图的一个顶点！
- 定理：Voronoi 图的所有顶点都必然会在某次圆事件中被发现。

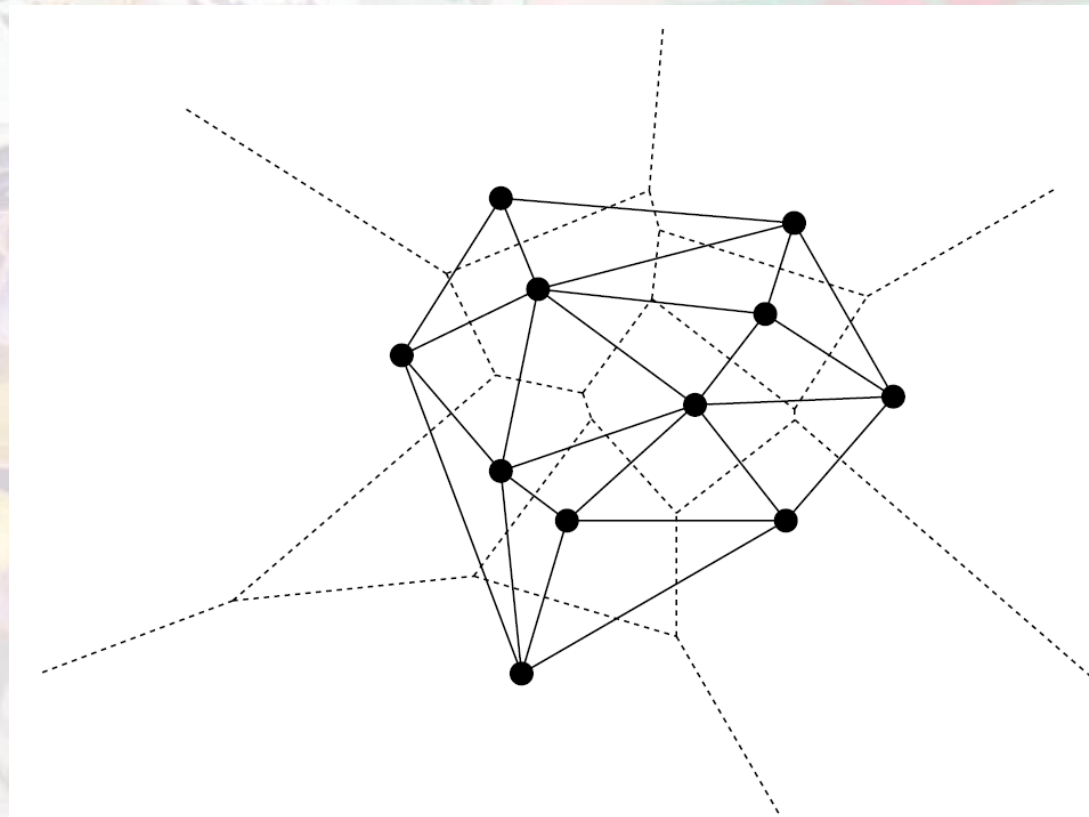


Fortune Algorithm

- 使用优先队列来储存所有事件，然后使用平衡树来维护海岸线。
- 具体过程不分析。
- 时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$ 。

Delaunay Triangulation

- Voronoi 图的对偶图:
- Voronoi 单元 $V(p)$ 所对应的的节点, 用点 p 来代替。
- 连接 $V(p)$ 和 $V(q)$ 的弧, 则用线段 \overline{pq} 来代替。



Complexity Of Delaunay Triangulation

- 定理：任何平面点集的 *Delaunay* 图都是一个平面图。
- 定理：设 P 为平面上不全部共线的任意 n 个点组成的集合，落在 P 的凸包边界上的点的个数记作 k 。则 P 的任何一个三角剖分（不一定是 *Delaunay* 三角剖分）必然由 $2n - 2 - k$ 个三角形组成，而且共有 $3n - 3 - k$ 条边。
- 证明依然使用欧拉公式。

A Significant Theorem

- 这个定理在 Voronoi 图里面已经介绍过了，在这里用 Delaunay 三角剖分的语言来描述一遍。
- 对于任一点集 P 所对应的 Delaunay 三角剖分 T ，下列命题成立：
 - $p_i, p_j, p_k \in P$ ，且同为 T 某一张面的顶点，当且仅当 p_i, p_j, p_k 外接圆的内部不包含 P 中的任何点。
 - $p_i, p_j \in P$ 同时与某条边相关联，当且仅当存在一个闭圆盘 C ，除了 p_i 和 p_j 落在其边界上，该圆盘不包含 P 中其他的任何点。
- 引理：设 P 为平面上的任一点集，而 T 为 P 的任意三角剖分。
- 则 T 是 P 的 Delaunay 三角剖分，当且仅当在 T 中的每个三角形的外接圆内部，都不包含 P 中的任何点。

Construction Of Delaunay Triangulation

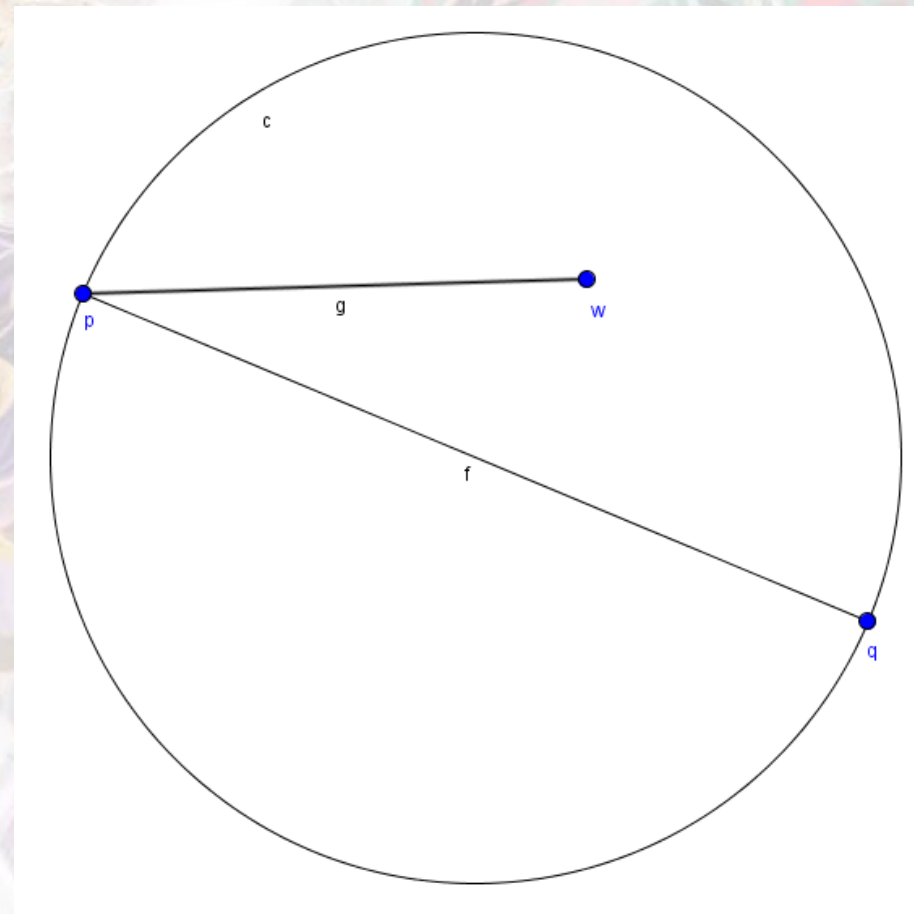
- Transform Voronoi Diagram Into Delaunay Triangulation – $O(n \log n)$
 - Hard to implement.
- Divide-and-conquer Algorithm – $O(n \log n)$
 - Code length: 5.1k in C++

Divide-and-conquer Algorithm

- 将所有点按照坐标排序，然后分治。
- 如果剩下不多于 3 个点，我们就可以直接连边。
- 否则我们先将点集上公切线连了，然后依次检查和这条线两个端点中任意一个端点相连的边，选择一个在上次连线顺时针方向的，且和原本线段端点确定的圆内没有任何点的点。然后连接点对并且删除穿过的边。
- 如果你每次都检查和线段端点相连的所有边，是可能达到 $O(n^2)$ 的。
- 要保证复杂度，需要利用单调性：对于与上一条连接线段的一个点相连的所有点（以左边为例），将其按照与基础边的极角，从小到大，若该点与线段端点形成的圆不包含下一个点，那么就得到答案，否则删除该点与线段端点的连接，考虑下一个点。
- 时间复杂度 $O(n \log n)$ 。

Euclidean Minimum Spanning Tree

- 定理：点集 P 的欧几里得距离最小生成树一定是 P 的 *Delaunay* 三角剖分的一个子图。
- 证明：
- 假设 (p, q) 是 *EMST* 上一条不属于 *Delaunay* 三角剖分的边，根据之前的定理，可得 (p, q) 为直径的圆上会有其他点。假设其中一个是 w 。
- 不失一般性，假设断开 (p, q) 之后 w 属于 q 所在的子树。
- 显然我们可以断开 (p, q) 连接 (p, w) ，然后你会得到一棵更小的生成树。
- 那么我们求 *EMST* 只需要三角剖分一发然后直接将所有边拿出来做 *Kruscal* 即可。
- 时间复杂度 $O(n \log n)$ 。





Thanks For Listening!

你居然看到这里了，真是太巨了！