計算幾何

Lecture by WiwiHo
Credit: baluteshih, yp155136, TreapKing

Sprout

開始之前

- 影片看了嗎？
  - 那個影片只有 34 分鐘
- 作業寫了嗎？

Sprout

大綱

- 前言與複習
- 向量的基本應用
- 凸包
- 總結

Sprout

前言與複習

Sprout

影片都說了些啥

- 座標與向量
  - 向量基本運算，加減乘除內外積
- 有向面積、線段相交
- 誤差分析
- 投影片後面還有一些東西但他沒有講

Sprout

前言：計算幾何與數學

- 計算幾何的名字裡有個幾何，所以很多人會認為它是數學

前言：計算幾何與數學

- 計算幾何的名字裡有個幾何，所以很多人會認為它是數學
- 事實上，大多數時候計算幾何的數學成份都沒那麼重
- 通常需要的只有**熟悉計算幾何的基本操作**，除此之外都不是什麼困難的數學
  - 剩下就是觀察力（多畫畫可以看出來的東西）＋實作

Sprout

前言：計算幾何與數學

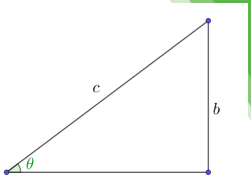
- 計算幾何的名字裡有個幾何，所以很多人會認為它是數學
- 事實上，大多數時候計算幾何的數學成份都沒那麼重
- 通常需要的只有**熟悉計算幾何的基本操作**，除此之外都不是什麼困難的數學
  - 剩下就是觀察力（多畫畫可以看出來的東西）＋實作
- 今天我們會把重點放在**二維幾何**的基本操作上

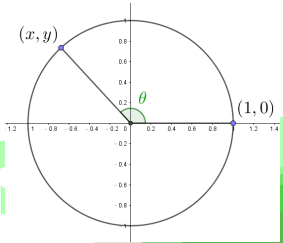
弧度

- ~~你已經是個大人了，該用弧度表示角的大小了~~
- $2\pi = 360°$
- $\pi = 180°$
- $x = \frac{x}{2\pi} \times 360°$
- 弧度沒有單位
- C/C++ 和多數程式語言裡的三角函數都是用弧度
- 弧度 x 的意思是，半徑為 1、弧度 x 的扇形的那個弧長度是 x

三角函數

- $\sin\theta = \frac{b}{c} = y$
- $\cos\theta = \frac{a}{c} = x$
- $\tan\theta = \frac{b}{a} = \frac{y}{x}$



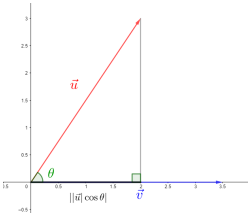


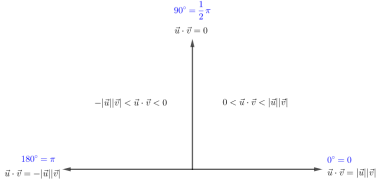
反三角函數

- arcsin、arccos、arctan
- 在 C++ 裡：`asin(v)`、`acos(v)`、`atan(v)` ／ `atan2(y, x)`
- 值域
  - `asin(v)`：$[-\frac{\pi}{2}, \frac{\pi}{2}]$
  - `acos(v)`：$[0, \pi]$
  - `atan(v)`：$[-\frac{\pi}{2}, \frac{\pi}{2}]$
  - `atan2(y, x)`：$[-\pi, \pi]$
  - 忘記的話去 cppreference 查
- `atan2(y,x)`：$+x$ 軸轉到 (x, y) 的角度

內外積

- $\vec{u} = (x_1, y_1), \quad \vec{v} = (x_2, y_2)$
- 內積：$\vec{u} \cdot \vec{v} = x_1x_2 + y_1y_2 = |\vec{u}||\vec{v}| \cos\theta$
  - θ 不管往哪邊轉都是一樣的！
  - A 向量在 B 向量上的**投影長**乘上 B 向量長度
  - 判直角

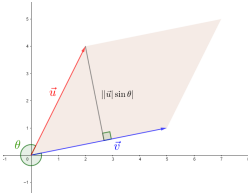


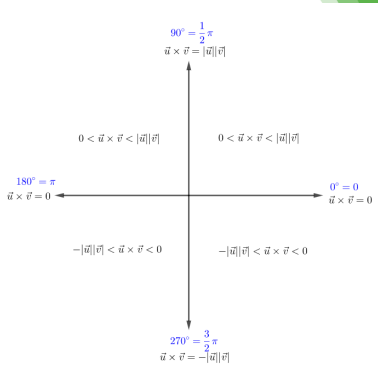


Sprout

內外積

- $\overrightarrow{u} = (x_1, y_1), \quad \overrightarrow{v} = (x_2, y_2)$
- 外積：
  $\overrightarrow{u} \times \overrightarrow{v} = x_1 y_2 + y_1 x_2 = |\overrightarrow{u}||\overrightarrow{v}| \sin\theta$
  - θ 必須是 $\overrightarrow{u}$ **逆時針**轉到 $\overrightarrow{v}$ 的那一邊
  - 兩個向量張出來的平行四邊形**有向**面積





方向？

- 三角形 ABC 的**有向面積**：$\frac{1}{2}\overrightarrow{AB} \times \overrightarrow{AC}$
  - 逆時針是正的，順時針是負的

方向？

- 三角形 ABC 的**有向面積**：$\frac{1}{2}\overrightarrow{AB} \times \overrightarrow{AC}$
  - 逆時針是正的，順時針是負的
- 多邊形的有向面積：$\frac{1}{2}\sum_{i=0}^{N-1} \overrightarrow{P_i} \times \overrightarrow{P_{i+1}}$
  - aka 測量師公式、鞋帶公式
  - 可以算凹多邊形！

方向？

- 三角形 ABC 的**有向面積**：$\frac{1}{2}\overrightarrow{AB} \times \overrightarrow{AC}$
  - 逆時針是正的，順時針是負的
- 多邊形的有向面積：$\frac{1}{2}\sum_{i=0}^{N-1} \overrightarrow{P_i} \times \overrightarrow{P_{i+1}}$
  - aka 測量師公式、鞋帶公式
  - 可以算凹多邊形！
- 我喜歡讓很多東西有方向，像是直線
  - 直線 AB 的左手邊就是 $\overrightarrow{AB} \times \overrightarrow{AP}$ 為正的那側、右手邊是為負的那側

方向函式

```
int sign(double a){
    if(abs(a) < eps) return 0;
    return a > 0 ? 1 : -1;
}
int ori(const pdd &a, const pdd &b, const pdd &c){
    return sign(cross(b - a, c - a));
}
```

線段香蕉相交

```
bool banana(const pdd &p1, const pdd &p2,
            const pdd &p3, const pdd &p4){
    if(btw(p1, p3, p4) || btw(p2, p3, p4) ||
       btw(p3, p1, p2) || btw(p4, p1, p2))
        return true;
    return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 &&
           ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}
```

- 別忘記平行線！
- 怎麼判點在線段內？

Sprout

點在線段內

```
bool btw(pdd p, pdd a, pdd b){
    return ori(p, a, b) == 0 &&
           sgn(dot(a - p, b - p)) <= 0;
}
```

誤差分析

- 能不用浮點數盡量不用浮點數
  - 有些題目是請你輸出面積到小數一位，但是面積 $\times 2$ 根據題目一定會是整數
  - 這時候就用 long long 存答案，最後輸出的時候再處理小數部份就好了
- 如果要用小數呢？
  - 可以參考影片分析誤差
  - 或者 eps 直接設 $10^{-6} \sim 10^{-9}$ 附近，WA 了再改 XD
- 技巧
  - sign function
  - 寫分數運算

Sprout

誤差從何而來

- 浮點數本身不可能絕對精確 QQ
  - 每次計算都可能造成精度損失
- 不精確的計算
  - 有一些數學函數的計算是用泰勒展開之類的方式逼進的
  - 精度有限，且可能很慢
  - **三角函數**

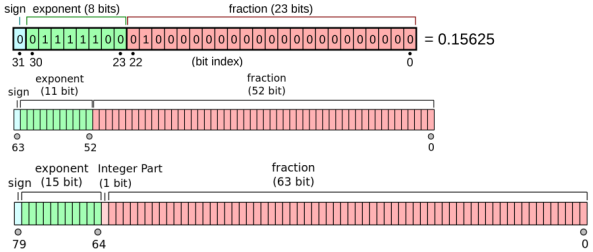
三角函數與誤差

```cpp
const double PI = acos(-1);

cout << "float      : "<< fixed << setprecision(30) << sinf(PI * 15 / 180) << "\n";
cout << "float      : "<< fixed << setprecision(30) << (sqrtf(6) - sqrtf(2)) / 4 << "\n";

cout << "double     : " << fixed << setprecision(30) << sin(PI * 15 / 180) << "\n";
cout << "double     : " << fixed << setprecision(30) << (sqrt(6) - sqrt(2)) / 4 << "\n";

cout << "long double: " << fixed << setprecision(30) << sinl(PI * 15 / 180) << "\n";
cout << "long double: " << fixed << setprecision(30) << (sqrtl(6) - sqrtl(2)) / 4 << "\n";

cout << "exact      : " << "0.258819045102520762348898837624048328349068901319930513814003207 3" << "\n";
```

float	0.258819043636322021484375000000
float	0.258819073438644409179687500000
double	0.258819045102520739476403832668
double	0.258819045102520683965252601411
long double	0.258819045102520734624599110796
long double	0.258819045102520762353069672113
exact	0.258819045102520762348898837624...

Sprout

IEEE 754



https://en.wikipedia.org/wiki/IEEE_754

https://en.wikipedia.org/wiki/Double-precision_floating-point_format

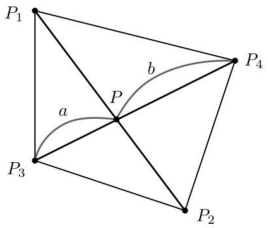
https://en.wikipedia.org/wiki/Extended_precision

（注意 long double 的實作在不同地方可能不一樣）

向量的基本應用

Sprout

線段交點



$$a : b = \triangle P_1P_2P_3 : \triangle P_1P_2P_4$$

$$\Rightarrow P = \frac{P_3 \cdot \triangle P_1P_2P_4 + P_4 \cdot \triangle P_1P_2P_3}{\triangle P_1P_2P_3 + \triangle P_1P_2P_4}$$

```
pdd intersect(pdd p1, p2, p3, p4){
  ld a123 = cross(p2 - p1, p3 - p1);
  ld a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) / (a123 - a124);
}
```

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$
  - $\overline{AC}$

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$
  - $\overline{AC}$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$
  - $\overline{AC}$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
  - $\min(\overline{AC}, \overline{BC})$

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$
  - $\overline{AC}$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
  - $\min(\overline{AC}, \overline{BC})$
- Case 3: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 內

點到線段距離

- 點 C 到線段 $\overline{AB}$ 的最短距離是？
- 可以證明，當給定的點都是格子點時，答案的平方是有理數，請輸出這個有理數
- Case 1: $A = B$
  - $\overline{AC}$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
  - $\min(\overline{AC}, \overline{BC})$
- Case 3: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 內
  - $\triangle ABC$ 以 $\overline{AB}$ 為底的高

點到線段距離

- Case 1: $A = B$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
- Case 3: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 內

點到線段距離

- Case 1: $A = B$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
- Case 3: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 內
- (?) 硬爆垂直投影出來

點到線段距離

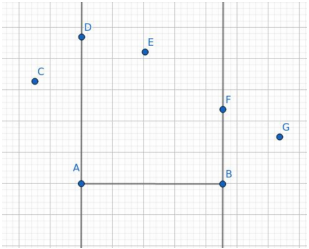
- Case 1: $A = B$
- Case 2: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 外
- Case 3: C 到 $\overline{AB}$ 的垂直投影在 $\overline{AB}$ 內
- (?) 硬爆垂直投影出來
- 這樣精度又差又差

點到線段距離：垂直投影在線段上

- 怎麼判 C 在直線 AB 的投影是不是在 $\overline{AB}$ 內？

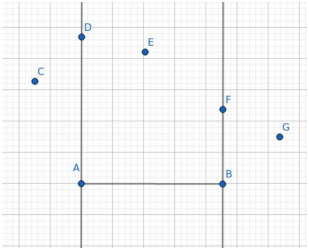
點到線段距離：垂直投影在線段上

- 怎麼判 C 在直線 AB 的投影是不是在 $\overline{AB}$ 內？



點到線段距離：垂直投影在線段上

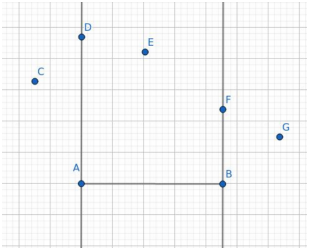
- 怎麼判 C 在直線 AB 的投影是不是在 $\overline{AB}$ 內？



- $\angle CAB$ 跟 $\angle CBA$ 都不可以是鈍角！

點到線段距離：垂直投影在線段上

- 怎麼判 C 在直線 AB 的投影是不是在 $\overline{AB}$ 內？



- $\angle CAB$ 跟 $\angle CBA$ 都不可以是鈍角！
- $\overrightarrow{AB} \cdot \overrightarrow{AC} \geq 0 \wedge \overrightarrow{BA} \cdot \overrightarrow{BC} \geq 0$

點到線段距離：點到直線距離

- $\triangle ABC$ 以 $\overline{AB}$ 為底的高？
- 硬爆投影點然後算距離
  - 當然不是這樣

點到線段距離：點到直線距離

- $\triangle ABC$ 以 $\overline{AB}$ 為底的高？
- 硬爆投影點然後算距離
  - 當然不是這樣
- (x_0, y_0) 到直線 $ax + by + c = 0$ 的距離是

$$\frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

  - 喔不，你掉進了浮點數的圈套！

點到線段距離：點到直線距離

- $\triangle ABC$ 以 $\overline{AB}$ 為底的高？
- 硬爆投影點然後算距離
  - 當然不是這樣
- (x_0, y_0) 到直線 $ax + by + c = 0$ 的距離是

$$\frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

  - 喔不，你掉進了浮點數的圈套！
- 還記得什麼東西的式子裡有高嗎？

點到線段距離：點到直線距離

- $\triangle ABC$ 以 $\overline{AB}$ 為底的高？
- 硬爆投影點然後算距離
  - 當然不是這樣
- (x_0, y_0) 到直線 $ax + by + c = 0$ 的距離是

$$\frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

  - 喔不，你掉進了浮點數的圈套！
- 還記得什麼東西的式子裡有高嗎？

$$\frac{|\overrightarrow{AB} \times \overrightarrow{AC}|}{|\overrightarrow{AB}|}$$

  - 這就是為什麼答案**的平方**肯定會是有理數

Sprout

點到線段距離：怎麼會這樣！

- 2021 全國賽 pB (TIOJ 2252)：反正就是求點到線段距離

Sprout

點到線段距離：怎麼會這樣！

- 2021 全國賽 pB (TIOJ 2252)：反正就是求點到線段距離
- https://gist.github.com/wiwiho/c9fe30e1649174721f03158ac7287f1f

Time	Score	Token	Subtask 1 (30)	Subtask 2 (30)	Subtask 3 (40)
1:14:05	30	No	30	0	0
2:17:12	30	No	30	0	0
2:20:30	30	No	30	0	0
2:28:13	60	No	30	30	0
2:30:23	60	No	30	30	0
2:35:04	100	No	30	30	40

Sprout

點到線段距離：怎麼會這樣！

- 2021 全國賽 pB (TIOJ 2252)：反正就是求點到線段距離
- https://gist.github.com/wiwiho/c9fe30e1649174721f03158ac7287f1f

Time	Score	Token	Subtask 1 (30)	Subtask 2 (30)	Subtask 3 (40)
1:14:05	30	No	30	0	0
2:17:12	30	No	30	0	0
2:20:30	30	No	30	0	0
2:28:13	60	No	30	30	0
2:30:23	60	No	30	30	0
2:35:04	100	No	30	30	40

- 聽說有很多人不會算，所以只好三分搜

Sprout

極角排序

- 把一堆點按照對著原點繞一圈的順序排序

Sprout

極角排序

- 把一堆點按照對著原點繞一圈的順序排序
- 方法一：sort by angle

```
bool cmp(pdd a, pdd b){
    return atan2(a.Y, a.X) < atan2(b.Y, b.X);
}
```

  - 優點：直觀
  - 缺點：慢、浮點數誤差、順序是從第三象限到第二象限

極角排序

- 方法二：分上下半平面、sort by cross

```
bool is_neg(pdd p){
    return p.Y == 0 ? p.X < 0 : p.Y < 0;
}
bool comp(pdd a, pdd b){
    int A = is_neg(a), B = is_neg(b);
    if(A != B) return A < B;
    return sgn(cross(a, b)) > 0;
}
```

- 你可以指出影片上的 code 的問題在哪嗎？
- 補充：角度一樣的時候，有時照長度排會很有用

一些其他的常見操作

- 點是否在多邊形裡
  - 凸多邊形有特殊版本，凹多邊形也可以做
- 圓形系列：圓圓切線、圓圓交點、點到圓切線、直線與圓交點
- 發揮想像力

Sprout

凸包

Sprout

多邊形的類型

- 簡單多邊形（simple polygon）：邊不相交的多邊形
- 凸多邊形（convex polygon）：內角都 $< 180^\circ$ 的多邊形
- 凹多邊形（concave polygon）：內角有 $> 180^\circ$ 的多邊形
- $= 180^\circ$ 我不想承認它是角

Sprout

凸包

- 有一堆點，你要用面積最小的凸多邊形把它們包起來

凸包

- 有一堆點，你要用面積最小的凸多邊形把它們包起來
- 一些性質：
  - 最邊邊的點一定在凸包裡面
  - 凸包上的點通常不會太多

凸包

- 有一堆點，你要用面積最小的凸多邊形把它們包起來
- 一些性質：
  - 最邊邊的點一定在凸包裡面
  - 凸包上的點通常不會太多
- 凸包求法：
  - Andrew's monotone chain
  - Graham's scan
  - Divide and Conquer（高維也可以這樣做）
  - ...

Monotone Chain

- 我們把凸包分成上凸包和下凸包
  - 以最左右兩個點為分界，最左邊的點屬於下凸包，最右邊的點屬於上凸包（不過有兩個的時候，一個上一個下）

Monotone Chain

- 我們把凸包分成上凸包和下凸包
  - 以最左右兩個點為分界，最左邊的點屬於下凸包，最右邊的點屬於上凸包（不過有兩個的時候，一個上一個下）
- 先把所有點照 (x, y) 排序
- 先把下凸包「圍」出來，再把上凸包「圍」出來，合起來就是凸包了

How to 「圍」

- 我們先圍下凸包

Sprout

How to 「圍」

- 我們先圍下凸包
- 開一個 stack（實作上常用 vector），記錄目前的下凸包

Sprout

How to 「圍」

- 我們先圍下凸包
- 開一個 stack（實作上常用 vector），記錄目前的下凸包
- 每加入一個新的點，它會使得有些人不可能再是凸包上的點！

How to 「圍」

- 我們先圍下凸包
- 開一個 stack（實作上常用 vector），記錄目前的下凸包
- 每加入一個新的點，它會使得有些人不可能再是凸包上的點！
- 假設新的點是 P，本來下凸包的最後兩個點 P_{-2}, P_{-1}，那我們要滿足

$$\overrightarrow{P_{-2}P_{-1}} \times \overrightarrow{P_{-1}P} \geq 0$$

不然 P_{-1} 絕對不是下凸包上的點！

How to 「圍」

- 我們先圍下凸包
- 開一個 stack（實作上常用 vector），記錄目前的下凸包
- 每加入一個新的點，它會使得有些人不可能再是凸包上的點！
- 假設新的點是 P，本來下凸包的最後兩個點 P_{-2}, P_{-1}，那我們要滿足

$$\overrightarrow{P_{-2}P_{-1}} \times \overrightarrow{P_{-1}P} \geq 0$$

  不然 P_{-1} 絕對不是下凸包上的點！
- 做完後我們就擁有整條下凸包了，不過最後一個點（最右邊點）我們預期它在上凸包，所以把它丟掉
- 把一開始排序好的順序反轉再做一次，就是上凸包

code

```
vector<pdd> hull;
for(int t = 0; t < 2; t++){
    int sz = hull.size();
    for(pdd p : pts){
        while(int(hull.size()) - sz >= 2 && ori(hull.end()
           [-2], hull.back(), p) <= 0)
            hull.pop_back();
        hull.push_back(p);
    }
    hull.pop_back();
}
```

凸包：小結

- Graham scan 就是一開始所有點對最下面的點極角排序，然後做一樣的事

Sprout

凸包：小結

- Graham scan 就是一開始所有點對最下面的點極角排序，然後做一樣的事
- 凸多邊形有很多非常好的性質，使得凸包可以做很多事

Sprout

凸包：小結

- Graham scan 就是一開始所有點對最下面的點極角排序，然後做一樣的事
- 凸多邊形有很多非常好的性質，使得凸包可以做很多事
- 有個有用且常見的應用，稱作「旋轉卡尺」，不過不在這堂課的範疇內
- 有興趣的同學可以上網自行學習

Sprout

總結

Sprout

今天講了什麼

- 我們今天講的東西很少
- 不過事實上，計算幾何不太有什麼「科技」
  - 就算有也是很 specific，沒有像什麼線段樹之類可以把一大類題目幹掉的東西
- 多數的計算幾何技巧都只是各種基本操作的應用與結合
- 因此我們今天的目標是希望大家都能熟悉基本的計算幾何操作

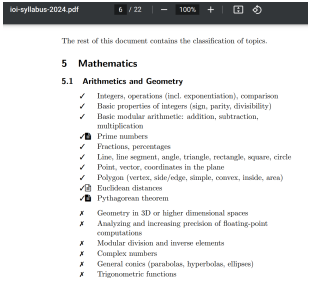
More topics

- 兩圓交點
- 圓跟多邊形的交集面積
- 旋轉卡尺
- 掃描線
- 最小包覆圓
- 半平面交
- Bentley-Ottmann algorithm （給你 N 個線段，請判斷這 N 個線段有沒有任意兩個線段相交）
- 模擬退火
- Voronoi Diagram
- Delaunay Triangulation
- 三維幾何

Sprout

計算幾何與競賽

IOI Syllabus:


ioi-syllabus-2024.pdf 6 / 22 100%

The rest of this document contains the classification of topics.

5 Mathematics

5.1 Arithmetics and Geometry

- ✓ Integers, operations (incl. exponentiation), comparison
- ✓ Basic properties of integers (sign, parity, divisibility)
- ✓ Basic modular arithmetic: addition, subtraction, multiplication
- ✓ Prime numbers
- ✓ Fractions, percentages
- ✓ Line, line segment, angle, triangle, rectangle, *square*, circle
- ✓ Point, vector, coordinates in the plane
- ✓ Polygon (vertex, side/edge, simple, convex, inside, area)
- ✓ Euclidean distances
- ✓ Pythagorean theorem

- ✗ Geometry in 3D or higher dimensional spaces
- ✗ Analyzing and increasing precision of floating-point computations
- ✗ Modular division and inverse elements
- ✗ Complex numbers
- ✗ General conics (parabolas, hyperbolas, ellipses)
- ✗ Trigonometric functions

不過在 ICPC 可以出現的類型就廣得多

Sprout

生活中的計算幾何

convexHull()

```
void cv::convexHull ( InputArray  points,
                      OutputArray hull,
                      bool        clockwise = false ,
                      bool        returnPoints = true
                    )
```

Python:

```
cv.convexHull( points[, hull[, clockwise[, returnPoints]]] ) -> hull
```

```
#include <opencv2/imgproc.hpp>
```

Finds the convex hull of a point set.

The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [245] that has *O(N logN)* complexity in the current implementation.

Parameters

points	Input 2D point set, stored in std::vector or Mat.
hull	Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves.
clockwise	Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards.
returnPoints	Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.

Note

`points` and `hull` should be different arrays, inplace processing isn't supported.

Check the corresponding tutorial for more details.