Introduction to Geometry

Mike Mirzayanov

Geometry on a Line

Given ***n*** segments on a line, the ***i***-th segment is given as a pair of numbers ***l*[*i*]**, ***r*[*i*]**. Check if there is a point belonging to each of them? I.e. is their intersection is non-empty?

Solution. Intersection (if any) is also a segment with endpoints: ***L*=max(*l*[1], *l*[2], …, *l*[*n*])** and ***R*=min(*r*[1], *r*[2], …, *r*[*n*])**.

Calculate ***L*** and ***R***. If ***L*** ≤ ***R*** then the segments intersect by the segment [L, R], otherwise there is *no intersection*.

Points and vectors

- Only about 2D, not about 3D (but many results can be generalized).
- Simplest objects are points and vectors.
- Use the same code for points and vectors: both of them are pairs (*x*, *y*).

Code snippets:

Separate Coordinates	Use std::pair	Use struct

`vector<int> x(n), y(n);`
`for (int i = 0; i < n; i++)`
`  cin >> x[i] >> y[i];`
`…`
	`#define X first`
	`#define Y second`
	`typedef pt pair<int,int>`
	`…`
	`for (int i = 0; i < n; i++)`
	`  cin >> p[i].X >> p[i].Y;`
		`struct pt {`
		`  int x, y;`
		`};`
		`…`
		`for (int i = 0; i < n; i++)`
		`  cin >> p[i].X >> p[i].Y;`

General Rules

- Prefer integer calculations if it is easy
- In case of integer calculations beware of overflows (consider “`long long`” or switch to use floating point calculations)
- In case of floating point calculations prefer “`long double`” instead of “`double`” (use typedef long double dbl)
- Use epsilon to compare float values. In most cases use:
  - **`const dbl EPS = 1E-9; // in practice 1E-12 ... 1E-6`**
- Exceptions to compare without epsilon: comparators for sortings, internals of predicate for binary search and other continuous cases.

General Rules

Be careful with precision errors. Main rule (simplified): any floating point operations can return inaccurate result.

```
dbl sina = … calculate somehow sine of angle a …
dbl a = asin(sina); // MISTAKE HERE!
```

Better

```
dbl a = asin(min(max(sina, -1.0L), 1.0L)); // crop to [-1, 1]
```

```
dbl root = sqrtl(x); // you know that x >= 0
```

Better

```
dbl root = sqrtl(max(x, 0.0L));
```

Useful Snippets

```
typedef long double dbl;
const dbl EPS = 1E-9;

inline int cmp(dbl a, dbl b) {
    return a + EPS < b ? -1
       : b + EPS < a ? +1 : 0;
}

inline bool zero(dbl x) {
    return cmp(x, 0) == 0;
}
```

```
struct pt { dbl X, Y; };

inline bool operator <(const pt& a,
        const pt& b) {
  return cmp(a.X, b.X)<0 || (cmp(a.X,
b.X) == 0 && cmp(a.Y,b.Y)<0);
}

inline bool operator ==(const pt& a,
        const pt& b) {
    return !(a < b) && !(b < a);
}
```

Useful Snippets

Reading

For “double” (scanf):

```
    double x;
    scanf(“%lf”, &x);
```

For “double” (cin):

```
    double x;
    cin >> x;
```

For “long double” (scanf):

```
    long double x;
    scanf(“%Lf”, &x);
```

For “long double” (cin):

```
    long double x;
    cin >> x;
```

Writing

For “double” (printf):

```
    printf(“%.10f”, x);
```

For “double” (cout):

```
    cout << setprecision(9) << fixed;
    cout << x;
```

For “long double” (printf):

```
    printf(“%.10Lf”, x);
```

For “long double” (cout):

```
    cout << setprecision(9) << fixed;
    cout << x;
```

Basic point/vector operations

- Squared length:

```
len2(p) {return p.X * p.X + p.Y * p.Y;}
```

- Length (avoid `hypot`):

```
len(p) {return sqrtl(p.X * p.X + p.Y * p.Y);}
```

- Normalize (scale to of length 1):

```
norm(p) {lenp = len(p); return {p.X/lenp, p.Y/lenp};
```

- rotate(p, a):

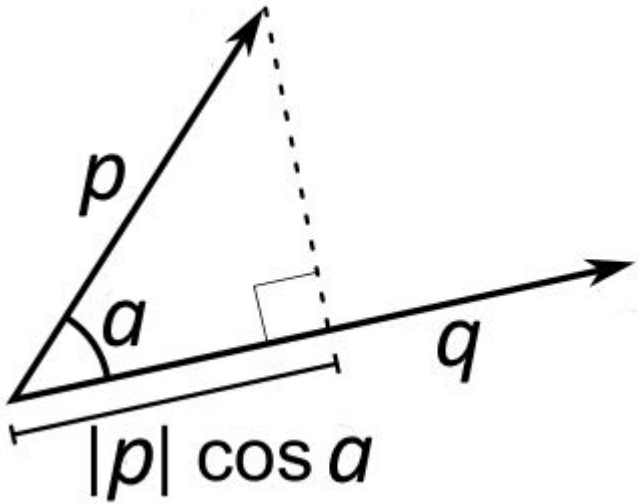
```
rotate(p, a) { return {cos(a)*p.X-sin(a)*p.Y,
    sin(a)*p.X+cos(a)*p.Y};}
```

- Add/subtract points/vectors, scale on factor.

Dot (Scalar) Product

$$p \cdot q = \text{dot}(p,q) = p.X * q.X + p.Y * q.Y = \cos(pq) * |p| * |q|$$

- Returns scalar, not vector
- Equals 0 if and only if vectors *p* and *q* are orthogonal (perpendicular)
- Positive is angle is acute, negative is angle is obtuse
- Signed length of projection of *p* on *q* multiplied by |*q*|
- Symmetric: $p \cdot q = q \cdot p$



Dot (Scalar) Product: Applications

- Given a segment AB and a point P. Check if projection of P on line AB belongs to the segment AB.
- Given a solid circle and a point P on the circle. Also a point A outside circle is given. Is the point P visible from the point A assuming the circle non-transparent (opaque).
- Given a vector *w* and two non-collinear vectors *p* and *q*. Find two such scalars *a* and *b*, that $w = a*p+b*q$.
- Given two vectors, find unoriented angle between them? I.e. value of smallest angle between vectors.
- Find orthogonal (perpendicular) vector.

Cross Product

In 2D we actually use pseudovector product:

$$p \times q = \mathtt{cross}(p,q) = p.X * q.Y - p.Y * q.X = \sin(pq) * |p| * |q|$$

- Returns scalar, not vector
- Equals 0 if and only if vectors are collinear
- Positive if and only if one should rotate *p* counterclockwise (in shortest way) to make the same direction as *q*; negative ⇔ clockwise
- Absolute value of cross product equals area of parallelogram build on two vectors.
- Anti-symmetric: $p \times q = -q \times p$

Cross Product: Applications

- Given a convex polygon $P_0,P_1,...,P_{n-1}$ and a point A. Check if the point A is strictly inside the polygon $P_0,P_1,...,P_{n-1}$.
- Given a line by two points A and B. Check if the given point P is on the line?
- Using cross product and dot product check if a point P is on a segment AB.
- Find area of triangle ABC and order of vertices (CW or CCW).
- Find total area of not necessary convex polygon $P_0,P_1,...,P_{n-1}$ specified by vertices in CW or CCW order. Also find type of order (CW or CCW).
- Test two vectors have the same direction.
- Check given polygon is convex.

Angles

- Good idea to operate only with normalized angles, say, in [-PI, PI) or [0, 2PI).
  Code: **`norm(&a) {while (a<-PI) a+=2*PI; while (a>=PI) a-=2*PI;}`**
- Use **`atan2(y,x)`** to return polar angle (point (x,y) is not an origin!)
- Rotate point *p* over *a*: **`{cos(a)*p.X-sin(a)*p.Y, sin(a)*p.X+cos(a)*p.Y}`**
- To find unoriented angle: **`acos(dot(p,q)/|p|/|q|)`**
- To find oriented angle from *p* to *q*: **`atan2(cross(p, q), dot(p, q))`**

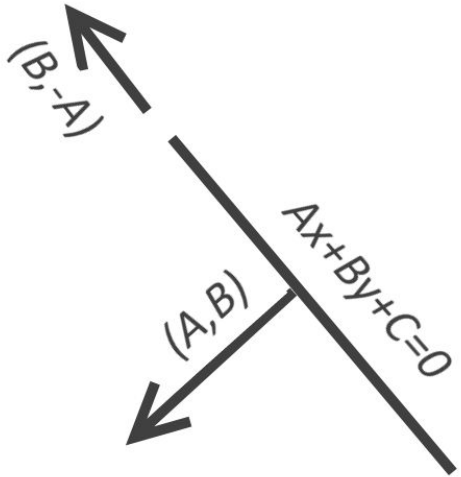
Vector normalization: transform vectors to be equal if and only if they have the same direction:

- for floating point calculations: divide vector by its length
- for integer calculations: find *d*=`gcd(|`*p*`.X|,|`*p*`.Y|)`, divide both coordinates by *d*.

Lines

Lines are described by equation:

$$A \cdot x + B \cdot y + C = 0, \text{ where } A \neq 0 \text{ or/and } B \neq 0.$$



- Vector (A,B) is a normal vector. Also vector (-A,-B) is other normal vector.
- Vector (B,-A) is a directing vector, also (-B,A) is a directing vector.
- Distance from point to a line is $|A \cdot x + B \cdot y + C| / \text{sqrt}(A^2+B^2)$. Sign matters. Usage example: find projection of a point to a line.
- Line by two points: $A := y_1 - y_2$, $B := x_2 - x_1$, $C := -A * x_1 - B * y_1$.

Line-Line Intersection

- Lines are parallel or coincide if and only if their normal vectors are collinear. I.e. cross product of normal vectors is zero: `A1*B2-A2*B1=0`.
- Cross product is also 2D determinant `det(a,b,c,d)=a*d-b*c`. For example, `cross(p,q)=det(p.X,p.Y,q.X,q.Y)`.
- Lines are coincide if and only if 3 determinants are zero:

  `det(A1,B1,A2,B2)=det(C1,B1,C2,B2)=det(A1,C1,A2,C2)=0`
- If lines are not parallel and not coincide, the intersection is (Cramer's rule):

  `x=-det(C1,B1,C2,B2)/det(A1,B1,A2,B2)`

  `y=-det(A1,C1,A2,C2)/det(A1,B1,A2,B2)`
- Remember, that `Ax+By+C=0` is not always a line (it could be empty set or whole plane).

Segment-Segment Intersection: Yes/No Algorithm

- Check bounding boxes intersect (4 lines in total), i.e.:
  - If the first segment strictly to the left than the second ⇒ NO

    `if (max(p1.X, q1.X) < min(p2.X, q2.X)) return false`
  - If the second segment strictly to the left than the first ⇒ NO

    `if (max(p2.X, q2.X) < min(p1.X, q1.X)) return false`
  - And the similar 2 lines about *y*'s.
- Check if the line through the first segment intersects the second segment.

  `sign(cross(q1-p1, p2-p1))*sign(cross(q1-p1, q2-p1))<=0`
- Check if the line through the second segment intersects the first segment.

  `sign(cross(q2-p2, p1-p2))*sign(cross(q2-p2, q1-p2))<=0`

Segment-Segment Intersection

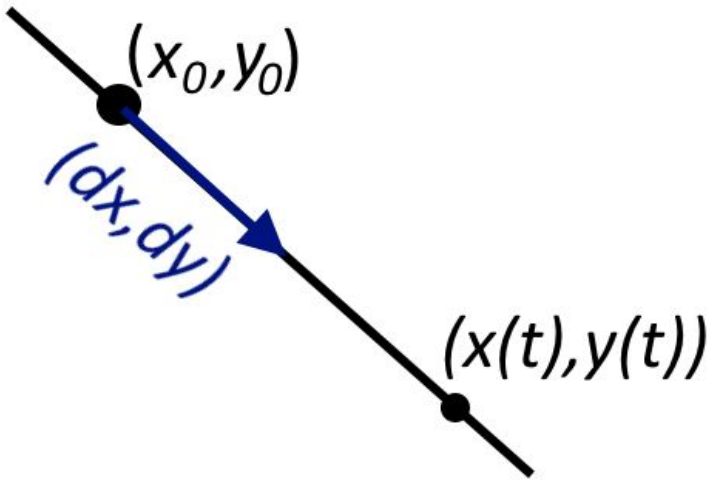
1. Check bounding boxes to intersect (as on the previous slide, 4 lines of code).
2. Build lines L1=(A1,B1,C1) and L2=(A2,B2,C2).
3. Find det=A1*B2-A2*B1 (=0 if and only if some line degenerates or lines parallel/coincide)
4. Check if lines coincide (do not test determinants, but test an endpoint of the second segment to lie on the first line AND an endpoint of the first segment to lie on the second line). If det=0 and not coincide => no intersections.
5. If coincide => they intersect by segment AB, there A=max(min(p1, q1), min(p2, q2)) and B=min(max(p1, q1), max(p2, q2)). If A==B then exactly 1 intersection else segments intersect by segment AB.
6. The only case has left: det ≠ 0. Find intersection of lines and check if it lies on both segments (use bounding boxes): `in(c.X, p1.X, q1.X) && in(c.Y, p1.Y, q1.Y) && in(c.X, p2.X, q2.X) && in(c.Y, p2.Y, q2.Y)`.

Polygon and Point

- Convex Polygon Case:
  - Slow but easy algorithm O(n): check all cross products to have same sign: PQ×PA, where PQ is a polygon side and A is the point to check
  - Fast algorithm O(log n): cut polygon on triangles from the first vertex, using binary search find sector where the point A is, check if A in the correspondent triangle.
- Not-convex Polygon Case O(n):
  - Check boundary separately. Find some point B outside polygon with fractional coordinates. Find number of intersections of the segment AB with the polygon sides. If even ⇒ outside , if odd ⇒ inside.

Circle-Line Intersection

Let’s move origin to the center of the circle. Now assume that the circle center is in (0,0).



Let’s use parametric equations to represent a point on the line:

$$x(t)=x_0+dx\cdot t,\ y(t)=y_0+dy\cdot t$$

where (x_0,y_0) is a arbitrary point on the line and (dx,dy) is a directing vector, i.e. you may use $dx=B$, $dy=-A$ where (A,B) is a normal vector.

Solve an equation: $x(t)^2+y(t)^2=r^2$, square equation for parameter t.

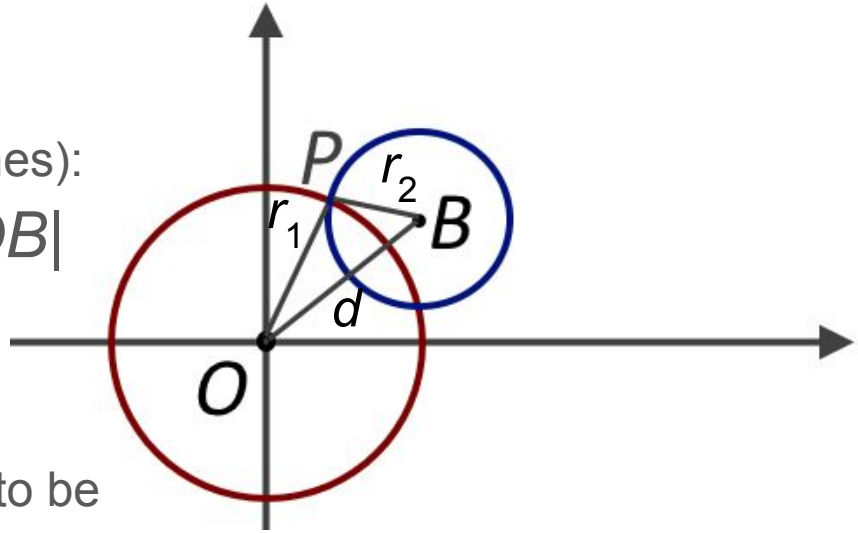
Circle-Circle Intersection

1. Let's move origin to the center of the circle. Just subtract coordinates of first circle center from the second circle center. Now assume that the first circle center is in (0,0).
2. Check if no intersections at all.
3. Find cosine of angle BOP (law of cosines):

$$cosa=(OB^2+OP^2-PB^2)/2/|OP||OB|$$

$$cosa=(d^2+r_1^2-r_2^2)/2/d/r_1$$

4. Find $sina=sqrt(1-cosa^2)$
5. Rotate OB using *sina* and *cosa*, scale to be of length r_1 instead of *d*. Move back adding coordinates of the first circle center.



Thank you!

Questions?