Some Basics

- Use integer arithmetic as much as possible.
- When using floating-point variables, compare by `fabs(a-b) < EPS`.
- Use some "small" `EPS` (e.g. `1e-8`).
- Be aware of degrees vs. radians.
- There are many special cases: parallel lines, colinear points, point on a line, etc.
- Often solves linear or quadratic equations in two unknowns (2D).
- Reduce problem to previously solved problems.

Basic Point Operations

- Take a point (2D) and rotate it about the origin.
- Given two points, compute the distance. Sometimes the squared distance is sufficient.

Orientation Test

- Given three points a, b, and c, if we were to travel from a to b to c, are we going clockwise or counterclockwise?
- What if the three points are colinear?
- This is computed by cross-products of vectors, and can be done with integer arithmetic only if input coordinates are integers.
- No divisions are needed.
- Basis of many geometry routines.
- See `ccw.cc`

Lines

- Defined by two points, or one point and direction.
- Distinguish between line segments and infinite lines.
- Best to represent lines in implicit form:

$$ax + by + c = 0$$

This avoids problems with vertical lines, for example.

Basic Line Operations

- Line intersection (infinite lines): `intersect_iline.cc` (solve equations).
- Line segment intersection: `intersectTF.cc` (orientation tests) and `intersect_line.cc` (solve equations)
- Distance from point to infinite line: `dist_line.cc` (dot product)
- Distance from point to line segment: see text.
- There are many special cases to handle. Best to use the code as is.

Example: Lining Up (270)

- Given n points (integer coordinates), find the size of the biggest subset of points that are colinear.
- For each point, find the "slopes" with respect to each of the other points.
- Sort the slopes, and look for duplicates.
- Vertical lines can be handled.
- $O(n^2 \log n)$

Circles

- A circle with radius r and centre (a, b) is defined by

$$(x - a)^2 + (y - b)^2 = r^2$$

- Checking whether a point is in a circle: compute distance to centre. Be careful with boundary.
- Recall: `pi = acos(-1.0);`
- A circle is uniquely defined by 3 non-colinear points: `circle_3pts.cc`
- Intersect two circles: `intersect_circle_circle.cc` (solve quadratic equations)

Example: Bounding Box (10577)

- Given 3 vertices of an n-sided regular polygon, find the smallest rectangle (parallel to the axes) that encloses the polygon.
- Need to find the coordinates of all n vertices.
- Find circle from given vertices. Rotate by $2\pi/n$ to get all vertices.

Triangles

- Many problems on polygons can be solved by partitioning the polygon into triangles (triangulation).
- Area of triangle given three sides (Heron's formula):

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s = (a+b+c)/2$$

See `heron.cc`

- Recall sine law and cosine law.

Spheres

- Coordinate systems: Cartesian or spherical.
- Special spherical system: longitude, latitude, radius (of Earth).
- Great circle distance: given two points on Earth, what is the shortest distance between them along the surface?
- See `greatcircle.cc`

Polygons

- A polygon can be represented as a list of vertices.
- Usually in counterclockwise order.
- Can be convex and concave.
- Remember that the list of vertices is "cyclic".

Convexity Test

- Look at the orientation of every group of three consecutive vertices.
- The polygon is convex if and only if all of them have the same orientation (ignore colinear points).
- Complexity: $O(n)$

Point in Polygon Test

- Given a point and a polygon, is the point inside the polygon?
- One approach: draw a ray from the point upwards (to infinity), and count the number of times the ray intersects with the polygon.
- If the number of intersections is odd, the point is in the polygon.
- Have to be careful if ray intersects a vertex or an edge.
- Works for both convex and concave polygon.
- Complexity: $O(n)$
- See `pointpoly.cc`

Area of Polygon

- Use the "surveyor's algorithm": sum up (signed) areas of triangles.
- Area is negative if the vertices are in clockwise order.
- Complexity: $O(n)$.
- See `areapoly.cc`

Convex Hull

- Given a set of n points, find the smallest convex polygon containing these points.
- It is also the polygon of minimum perimeter containing these points.
- Imagine releasing a rubber band around pegs at these points. The resulting figure is the convex hull.
- Graham scan: `convex_hull.cc`
- Complexity: $O(n \log n)$.

Example: Doors and Penguins (3581)

- Given two set of points (up to 500 each), is there a line that separates the two sets of points?
- If it is possible, then the convex hulls of each set of points do not intersect.
- The converse is also true.
- To check if convex hulls intersect, we can perform line intersection tests on pairs of boundaries.
- There is a more efficient way, but that is not required here.