This course material is now made available for public usage. Special acknowledgement to School of Computing, National University of Singapore for allowing Steven to prepare and distribute these teaching materials.

CS3233
Competitive Programming
Dr. Steven Halim
Week 11 – (Computational) Geometry
Outline

• Mini Contest #9 + Discussion + Break + Admins
• Geometry Basics + Prepare Your Libraries
  – Points, Lines, Circles, Triangles, Polygons (Focus)
• Not discussed tonight:
  – Quadrilaterals
  – 3D Objects: Spheres
  – Other 3D Objects: Cones, Cylinders, etc
  – Plane Sweep technique
  – Intersection problems
  – Divide and Conquer in geometry problems
The major part of the hard copy material of a top ICPC team is usually a collection of geometric libraries...

GEOMETRY BASICS AND LIBRARIES
Some Comp Geometry Principles

- Whenever possible, we prefer test (predicates) than computing the exact numerical answers
- Tests:
  - Avoid floating point operations (division, square root, and any other operations that can produce numerical errors)
  - Preferably, all operations are done in integers
  - If we really need to work with floating point, we do floating point equality test this way: \( \text{fabs}(a - b) < \text{EPS} \)
    where \( \text{EPS} \) is a small number like \( 1e-9 \) instead of \( a == b \)
Geometry Basics – 0D (1)

• Point, representation + sorting feature

struct point_i { int x, y }; // use this whenever possible
struct point { double x, y }; // but I will use this form now

struct point { double x, y;
    point(double _x, double _y) { x = _x, y = _y; }
    bool operator < (point other) {
        if (fabs(x - other.x) > EPS) // useful for sorting
            return x < other.x; // first criteria, by x-axis
        return y < other.y; // second criteria, by y-axis
    }
};
Geometry Basics – 0D (2)

• Comparing Points

```cpp
bool areSame(point p1, point p2) { // floating point version
  // use EPS when testing equality of two floating points
  return fabs(p1.x - p2.x) < EPS && fabs(p1.y - p2.y) < EPS; }
```

• Euclidean Distance between two points

```cpp
double dist(point p1, point p2) { // Euclidean distance
  // hypot(dx, dy) returns sqrt(dx * dx + dy * dy)
  return hypot(p1.x - p2.x, p1.y - p2.y); } // return double
```
• Lines (ch7_01_points_lines.cpp/java)
  – Poor line equation, \( y = mx + c \) (vertical line \( \rightarrow \) special case)
  – Better line equation, \( ax + by + c = 0 \)

```c
struct line { double a, b, c; }; // a way to represent a line

// the answer is stored in the third parameter (pass byref)
void pointsToLine(point p1, point p2, line *l) {
  if (p1.x == p2.x) { // vertical line is handled nicely here
    l->a = 1.0;   l->b = 0.0;   l->c = -p1.x;
  } else {
    l->a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l->b = 1.0; // fix the value of b to 1.0
    l->c = -(double)(l->a * p1.x) - (l->b * p1.y);
  }
}
```

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS
Geometry Basics – 1D (2)

• Interaction between two lines

```cpp
bool areParallel(line l1, line l2) { // check coefficient a + b
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) { // also check coefficient c
    return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS); }
```
Geometry Basics – 1D (3)

• Interaction between two lines – continued
  – Simple linear algebra: \(a_1x + b_1y + c_1 = a_2x + b_2y + c_2\)

```cpp
// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point *p) {
    if (areSame(l1, l2)) return false; // all points intersect
    if (areParallel(l1, l2)) return false; // no intersection
    // solve system of 2 linear algebraic equations with 2 unknowns
    p->x = (double)(l2.b * l1.c - l1.b * l2.c) /
           (l2.a * l1.b - l1.a * l2.b);
    if (fabs(l1.b) > EPS) // test for vertical line
        p->y = - (l1.a * p->x + l1.c) / l1.b; // avoid div by zero
    else // this is another special case in geometry problem...
        p->y = - (l2.a * p->x + l2.c) / l2.b;
    return true; }
```
• Line segments: line with two endpoints (finite length)
• Vector: line segment with a direction
• We can translate (move) a point w.r.t a vector

```c
struct vec { double x, y; // similar to point
    vec(double _x, double _y) { x = _x, y = _y; } 
};
vec toVector(point p1, point p2) { // convert 2 points to vector
    return vec(p2.x - p1.x, p2.y - p1.y); }
vec scaleVector(vec v, double s) { // s = [<1 ... 1 ... >1]
    return vec(v.x * s, v.y * s); } // shorter v same v longer v
point translate(point p, vec v) { // translate p according to v
    return point(p.x + v.x, p.y + v.y); }
```
Geometry Basics – 2D/Circles (1)

• Circles (ch7_02_circles.cpp/java)
  – A circle centered at \((a, b)\) and radius \(r\) is the set of all points \((x, y)\) such that
    \[(x - a)^2 + (y - b)^2 = r^2\]

  ```
  int in_circle(point p, point c, int r)
  // 0 – inside, 1 – at border, 2 – outside
  ```

  – \(\pi = 2 \cdot \text{acos}(0.0)\)
  – Diameter \(d = 2 \cdot r\)
  – Circumference \(c = \pi \cdot d\)
  – Area of circle \(A = \pi \cdot r \cdot r\)
Geometry Basics – 2D/Circles (2)

- Arc length: $\alpha / 360.0 \times c$
- Chord length:
  - $\frac{\sqrt{2 \times r \times r / (1 - \cos(\alpha))}}{2}$
- Sector area: $\alpha / 360.0 \times A$
- Segment area: sector area – isosceles triangle area
Geometry Basics – 2D/Triangles (1)

• Triangles (ch7_03_triangles.cpp/java)
  – Polygon with three vertices and three edges
  – Area of Triangle 1: \( A = 0.5 \times b \times h \)
  – Perimeter \( p = a + b + c \)
    • where \( a, b, c \) are the length of the 3 edges
  – Area of Triangle 2: \( A = \sqrt{s \times (s - a) \times (s - b) \times (s - c)} \)
    • where semi-perimeter \( s = 0.5 \times p \)
    • This is called the Heron’s formula
    • Safer from overflow: \( A = \sqrt{s} \times \sqrt{s - a} \times \sqrt{s - b} \times \sqrt{s - c} \)
      – But can be slightly more imprecise
Geometry Basics – 2D/Triangles (2)

- Given three points p, q, r
  - Determine the circumcenter $c_1$ and radius $R_1$ of the inner/inscribed circle/incircle and $(c_2, R_2)$ of the outer/circumscribed circle/circumcircle

\[ \text{Diagram showing points p, q, r, c1, c2 and circles with radii R1 and R2.} \]
Geometry Basics – 2D/Triangles (3)

- Trigonometry/Law of Cosines
  - \( c^2 = a^2 + b^2 - 2ab \cos(\gamma) \)

- Trigonometry/Law of Sines
  - \( \frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} \)

- Trigonometry/Phytagorean Theorem
  - \( c^2 = a^2 + b^2 \) because \( \cos(90.0 \text{ degrees/right angle}) = 0 \)
Geometry Basics – 2D/Others

• Quadrilaterals (no sample code)
  – Rectangles/Squares
  – Trapeziums/Parallelograms/Rhombus
  – Area
  – Perimeter
  – Etc...
Focus for CS3233 this semester

ALGORITHMS ON POLYGON
Polygon (1)

- Sample code (ch7_05_polygon.cpp/java)
  - Plane figure that is bounded by a closed circuit composed of a **finite sequence of straight line segments**
  - Basic form, vertices are ordered *either* in **cw** or **ccw** order
  - Usually the first = the last vertex

```cpp
vector<point> P;
P.push_back(point(1, 1));
P.push_back(point(3, 3));
P.push_back(point(9, 1));
P.push_back(point(12, 4));
P.push_back(point(9, 7));
P.push_back(point(1, 7));
P.push_back(P[0]); // loop back
```
Polygon (2)

- Perimeter of polygon (trivial)

```cpp
// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(vector<point> P) {
    double result = 0.0;
    for (int i = 0; i < (int)P.size(); i++)
        result += dist(P[i], P[(i + 1) % P.size()]);
    return result; }
```
Area of a Polygon

- Given the vertices of a polygon in a circular manner (cw or ccw), its area is given by

\[
A = \frac{1}{2} \sum_{i=1}^{n} (x_i y_{i+1 \mod n} - x_{i+1 \mod n} y_i)
\]
Polygon (3)

- Area of polygon

```cpp
// returns the area, which is half the determinant
double area(vector<point> P) {
    double result = 0.0, x1, y1, x2, y2;
    for (int i = 0; i < (int)P.size(); i++) {
        x1 = P[i].x; x2 = P[(i + 1) % P.size()].x;
        y1 = P[i].y; y2 = P[(i + 1) % P.size()].y;
        result += (x1 * y2 - x2 * y1);
    }
    return fabs(result) / 2.0;
}
```
Polygon/Convex Hull (1)

- The Convex Hull of a set of points P is the smallest convex polygon CH(P) for which each point in P is either on the boundary of CH(P) or in its interior.
Polygon/Convex Hull (2)

- Graham’s Scan algorithm
  1. Find pivot (bottom most, right most point)
  2. Angular sorting w.r.t pivot (easy with library)
  3. Series of ccw tests (with help of stack)
Summary

• In this lecture, you have seen:
  – Basic geometry routines (quite substantial)
    • But still... many others routines are skipped :O
  – Focus on (some) algorithms on polygon

• But... you need to practice using them!
  – Especially, scrutinize ch7_05_polygon.cpp/java
  – Solve one UVa problem involving polygon
  – We will have a comp geo contest next week 😊
References

- CP2.9, Chapter 7
- Introduction to Algorithms, $2^{nd}/3^{rd}$ ed, Chapter 33