Programming Challenges (GB21802)

Week 9 - Computational Geometry

Claus Aranha
caranha@cs.tsukuba.ac.jp

University of Tsukuba, Department of Computer Sciences

2020/6/23

(last updated: June 23, 2020)

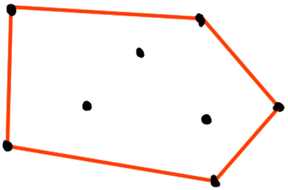
Version 2020.1

What is Computational Geometry?

In programming challenges, Computational Geometry problems involve answering questions about **lines, points and angles**. Some examples of pure Comp. Geometry problems:

Example 1

Given a set of N points $(s_1, s_2, s_3, \ldots, s_N)$, what is the area of the smallest polygon that covers all points in the set?

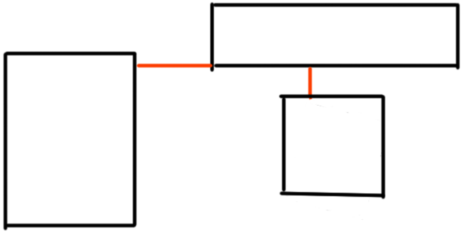


What is Computational Geometry?

In programming challenges, Computational Geometry problems involve answering questions about **lines, points and angles.** Some examples of pure Comp. Geometry problems:

Example 2

Given N rectangles, $\{x_1, y_1, w_1, h_1\}; \ldots; \{x_N, y_N, w_N, h_N\}$, what is the smallest length of line segments needed to connect them?

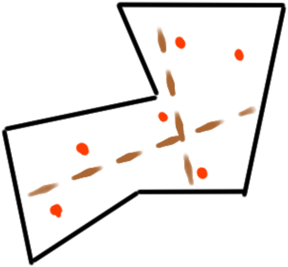


What is Computational Geometry?

In programming challenges, Computational Geometry problems involve answering questions about **lines, points and angles**. Some examples of pure Comp. Geometry problems:

Example 3

Given a polygon and a set of N points, find a line that divides the polygon in equal areas, with the same number of points in each area?



Computational Geometry

The good and the bad

Computational Geometry problems have some merits and demerits when compared to other problems that we studied until now.

Positive Points

- Geometry problems are fun, and you draw pretty pictures when thinking about them (ok, maybe this one is a bit personal);
- A large part of geometry problems can be solved with algorithms and techniques that you learned in high school;
- The code for techniques is highly re-usable;

Negative Points

- You have to write a lot of code (in the beginning, at least);
- Easy to get WE for small mistakes;
- Many special cases in the input data;

Common Mistakes in Geometry Problems

Errors because of special cases in input data

- Multiple points in the same position;
- Collinear points (three points in the same line);
- Vertical lines (bad tangent value, division by 0);
- Parallel Lines (bad intersection value);
- Intersection at end of a segment;
- etc;

Floating Number Precision Errors

- Wrong Answer because of poor rounding of final result;
- Error Propagation inside functions (multiplication, division);

Common Mistakes in Geometry Problems

How to avoid these mistakes in Geometry Problems?

- **Special Cases**:
  - Make sure to think which special cases affect your tecnique, and add checks for these cases;
  - When testing your problem, include input with the special case;
- **Precision Errors**:
  - If possible, convert values to integers before calculation;
  - When testing equality of two values, use an **Epsilon Constant**:

```
if (float.1 == float.2) then             // NO
if (fabs(float.1 - float.2) < EPS) then // YES!
```

Class Outline

In this lecture, we will focus on:

- Discussion of implementation of geometric operations;
- Discussion of problem examples;

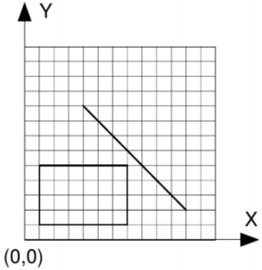
Specific topics will be:

- Intersection and Rotation of points and lines;
- Circle representation and components;
- Triangles (area, angles, triangles and circles);
- Polygon representation and Convex Hull;

Problem Example: 191 – Intersection

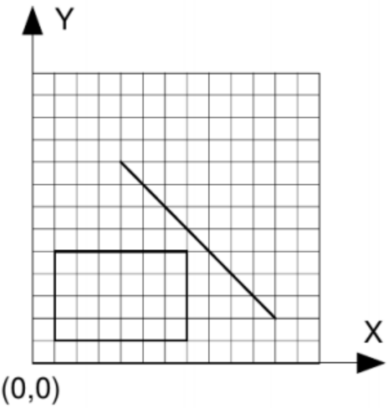
Summary

- **Input**: A rectangle and a line segment:
  - Rectangle: $x_s y_s x_e y_e$
  - Line: x_0, y_0, x_1, y_1
- **Output**
  T - if the line segment intersects the rectangle
  F - if the line segment does not intersect the rectangle



How do you solve it?

Problem Example: 191 – Intersection

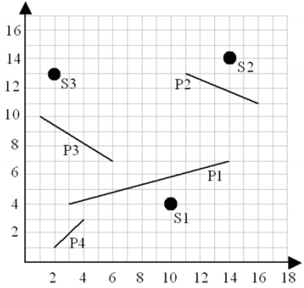


- Test if p_1 or p_2 are inside the rectangle;
- For each segment $\overline{AB}$, test if $\overline{p_1, p_2}$ intersects $\overline{AB}$.
- (optional) make a videogame;

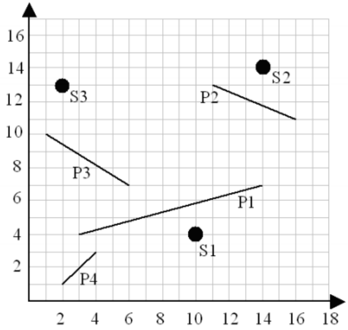
Problem Example: 833 – Waterfalls

Summary

- **Input**
  - List of line segments that block water;
  - List of water sources;
- **Output**
  - For every waterfall w, the end position X_w where $Y_w = 0$



Problem Example: UVA 833 – Waterfalls



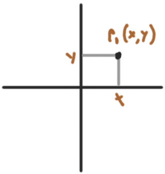
Full Search:

- For each water source S_i:
  - Calculate which segment intersects $\overline{S_i 0}$ first.
  - Adjust the position X_i and repeat until $Y_i = 0$.

- This is a bit slow if there are many sources and segments.
- Can you pre-calculate something?
- Remember that all inputs are integers!

Basic Functions – Points and Lines

Point Representation




```
struct point_i { int x, y;  // integer coordinates
  point_i() { x = y = 0; }
  point_i(int _x, int _y) : x(_x), y(_y) {}};

struct point { double x, y; // double coordinates
  point() { x = y = 0.0;}
  point(double _x, double _y) : x(_x), y(_y) {}};
```

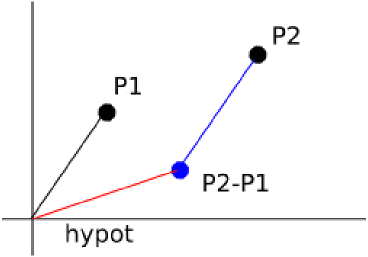
Overloading Point Operators

Point Operators

```
struct point { double x, y;
   point() { x = y = 0.0;
   point(double _x, double _y) : x(_x), y(_y) {}

   // Overloading "<" to sort by coordinate
   bool operator < (point other) const {
      if (fabs(x - other.x) > EPS)
         return x < other.x;
      return y < other.y; }
   // Overloading "==" for equality testing
   bool operator == (point other) const {
      return (fabs(x - other.x) < EPS &&
             (fabs(y - other.y) < EPS)); }
   }
```

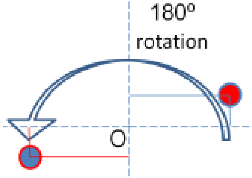
Point Distance

```
            // Euclidean Distance: "normal" distance
#define hypot(dx,dy) sqrt(dx*dx + dy*dy)
double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y); }
            // Taxicab Distance: Distance on a grid
double taxicab(point p1, point p2) {
  return fabs(p1.x - p2.x) + fabs(p1.y - p2.y); }
```



Point Rotation

```
#define PI          3.14159265358979323846 // Pi constant
double PI = 2 * acos(0.0)                  // Better Pi
#define DEG_to_RAD(X) (X*PI)/180.0

// angle t is in degrees (0--360)
point rotate(point p, double t) {
   double rad = DEG_to_RAD(t);
   return point(p.x * cos(rad) - p.y * sin(rad),
                p.x * sin(rad) + p.y * cos(rad));}
```



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

Quiz: How do you rotate a point around x_1, y_1?

Line Basics

How to represent a line:

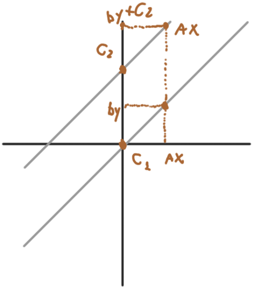
- $ax + by + c = 0$ (a,b,c) – useful for most cases
- $y = mx + c$ (m,c) – useful for angle manipulation
- x_0, y_0, x_1, y_1 (p_1, p_2) – harder to use, but common input

How to convert from two points to a line

```
struct line { double a,b,c; };

void pointsToLine(point p1, point p2, line &l) {
   if (fabs(p1.x - p2.x) < EPS {
      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; l.c = -(double) (l.a*p1.x) - p1.y;}
}
```

Line Equality

- We define a line as $a, b, c | (ax + by = c)$
- Two lines are parallel if their coefficients (a, b) are the same;
- Two lines are identical if all coefficients (a, b, c) are the same;




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

bool areSame(line l1, line l2) {
   return areParallel(l1,l2) &&
          (fabs(l1.c - l2.c) < EPS); }
```

Line Intersection

The intersection point x_I, y_I can be found by solving:

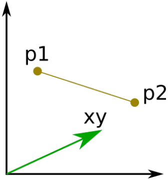
$$a_1 x_I + b_1 y_I + c_1 = 0$$
$$a_2 x_I + b_2 y_I + c_2 = 0$$

Remember that when we create a line i, we set $b_i = 0$ or $b_i = 1$

```
bool areIntersect(line l1, line l2, point &p) {
   if (areParallel(l1,l2)) return False;

   p.x = (l2.b * l1.c - l1.b * l2.c) /
         (l2.a * l1.b - l1.a * l2.b);

   // Test for vertical case:
   if (fabs(l1.b) > EPS)       p.y = -(l1.a * p.x + l1.c);
   else                        p.y = -(l2.a * p.x + l2.c);
   return True;
}
```

Segments and Vectors

- A **line segment** is a line with two endpoints (p_1, p_2) and finite length;
- A **Vector** is a line segment with a direction;
  - Usually represented as "direction" and "magnitude";
  - Direction is a point with distance 1 from $(0, 0)$
  - Magnitude is a multiplier to the size of the vector;
  - Represent movement, translation, speed, etc;




```
struct vec { double x, y;
    vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {
    return vec(b.x - a.x, b.y - a.y); }
vec scale(vec v, double s) {
    return vec(v.x * s, v.y * s); }
point translate(point p, vec v) {
    return point(p.x + v.x , p.y + v.y); }
```

Distance between point and line, point and segment

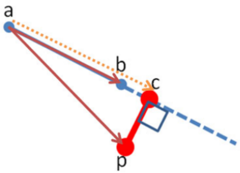
For a point p and a line l (represented by $\overline{ab}$), the distance between both is given by the segment pc, where c is the projection of p in l.

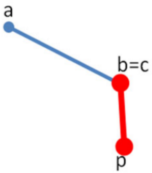
Calculating c:

- calculate scalar proj. u of $\vec{ap}$ in l.
- change magnitude of $\vec{ab}$ to u to obtain $\vec{ac}$.
- calculate distance between p and c.

Distance between p and $\overline{ab}$:

- calculate scalar proj. u of $\vec{ap}$ in l.
- if $u < 0$ or $u > |\vec{ab}|$, then the distance is $\min(d(a,p), d(b,p))$.
- else, calculate $\vec{ac}$ and calculate distance $d(p,c)$.





CODE: Distance between point and line

```
double dot(vec a, vec b) {                    // dot product
   return (a.x * b.x + a.y * b.y); }
double norm_sq(vec v) {                       // norm squared
   return v.x * v.x + v.y * v.y; }

// Given points a,b,p, calculate distance from p to line ab.
double distToLine(point p, point a, point b, point &c) {
  // point c: c = a + u * |ab|
  vec ap = toVec(a, p), ab = toVec(a, b);

  // dot product calculates size of ap in ab
  // norm square will calculate the scale to ab
  double u = dot(ap, ab) / norm_sq(ab);

  // translate a by u to find point c.
  c = translate(a, scale(ab, u));
  return dist(p, c);
}
```

CODE: Distance between point and segment

This function uses the same idea as the previous one. However, we must first test if the point *c* falls inside or outside of $\overline{ab}$.

```
double distToSegment(point p, point a, point b, point &c) {
  // next two lines is exact same as last slide
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);

  // test if the magnitude $u$ is bigger or smaller than ab.
  if (u < 0.0) { c = point(a.x, a.y); // closer to a
                 return dist(p, a); }
  if (u > 1.0) { c = point(b.x, b.y); // closer to b
                 return dist(p, b); }

  // c is inside AB, same as last slide
  c = translate(a, scale(ab, u));
  return dist(p,c);
}
```

Angle between segments

Given three points, a, b and o, we can calculate the angle between $\overline{oa}$ and $\overline{ob}$ using the dot product.

Given that: $oa \cdot ob = |oa| \times |ob| \times \cos(\theta)$, we have $\theta = \arccos(\frac{oa \cdot ob}{|oa| \times |ob|})$

```
#import <cmath>

// angle in radians (0..2*PI)
double angle(point a, point o, point b) {
  vec oa = toVector(o, a), ob = toVector(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}
```

Left, Right and Collinear Points

Given a line defined by points p and q, we are interested in knowing if point r is on the left/right side of the line, or if the three ponts are collinear.

Let $\vec{pq}$ and $\vec{pr}$ be two vectors, the **cross product** $\vec{pq} \times \vec{pr}$ is a vector that is perpendicular to both vectors. The magnitude of the cross product is positive / zero / negative if $p \rightarrow q \rightarrow r$ is left turn / collinear / right turn.

```
double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }

collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
```

Basic Functions – Circles and Triangles

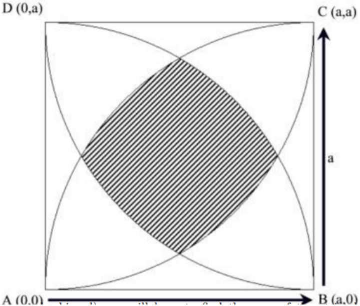
Circles

- A circle is stored as its center point c, and its radius r.
- The circle contains all points (x, y) where $(x-a)^2 + (y-b)^2 \leq r^2$
- No square root, so less chance of floating point errors.

Test if Point p is inside Circle – Integer Version

```
int insideCircle(point_i p, point_i c, int r) {
   int dx = p.x-c.x, dy = p.y-c.y;
   int Euc = dx*dx + dy*dy, rSq = r*r;
   return Euc < rSq ? 0 : Euc == rSq ? 1 : 2;
   // 0 - inside, 1 - border, 2- outside
}
```

Problem Example – UVA 10589 Area



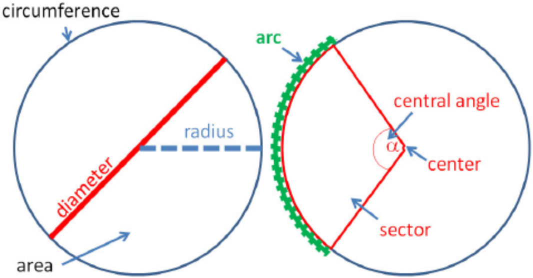
QUIZ:

- What is the area of the shaded part?
- You know *a*, the radius of the 4 circles;

Monte Carlo Approach

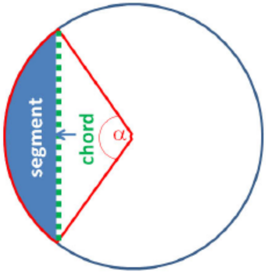
- Sample *N* random points;
- Calculate proportion *p* of points in area;
- Shaded area is $\frac{a^2}{p}$
- Mote Carlo approach is useful in several problems!

Other circle properties



- **radius:** r, **diameter:** 2r, **circumference:** $2 \times \pi \times r$
- You can obtain π from the problem, or with $\pi = 2 \times \arccos(0)$
- Given a central angle α:
  - **Arc:** $r \times \alpha$ (if in rad) or $r \times \frac{\alpha}{360} \times 2\pi$ (if degrees)
  - **Sector:** $\frac{\alpha r^2}{2}$ (if in rad) or $2\pi r^2 \times \frac{\alpha}{360}$ (if degrees)

Other circle properties – chord



- **chord:** Line segment with two ends in the circle's border.
- If you know p_1, p_2 and c, you can find α from the "angle" function;
- If you know α and r, you can find the size of the chord by: $|p_1 p_2| = 2 \times r \times \sin(\alpha/2)$
  - **Quiz:** If you know a line and a circle, how do you find p_1 and p_2?
- If you know p_1, p_2, and r (but not α or c), you can find the center of the circle using the code in the next page;

Circle Center from points and radius

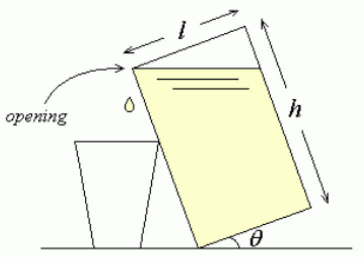
You have two points p_1, p_2 that form a chord in a circle, and the radius of that circle. How do you find the center?

```
bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;

  if (det < 0.0) return false; // Can't make circle

  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true;
}
// to get the other center, reverse p1 and p2
```

Problem Example: 11909 - Soya milk

How do you solve it?

- **Input**:
  The dimensions of a Milk box, and its inclination:
  l, w, h, θ
- **Output**:
  The amount of milk left in the box.



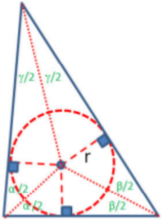
Example: 10577 - Bounding Box

- **Input**: Three points that are the vertices of a **regular polygon**, and number *n* of sides in the polygon;
- **Output**: Area of smallest **axis aligned** rectangle that bounds this polygon.
- How do you solve it?

Triangle Basic Facts

- **Triangle Inequality**: If a, b, c are sides of a triangle, then $a + b > c$; $a + c > b$; $b + c > a$;
- **Perimeter, Semiperimeter**: $p = a + b + c$, $s = p/2$
- **Area**: $A = \frac{bh_b}{2}$
- **Area (Heron's formula)**: $A = \sqrt{s(s-a)(s-b)(s-c)}$
- **Triangulation**: Any 2D polygon can be decomposed into triangles;

Incircle of a Triangle



- An **Inscribed Circle (incircle)** is the largest circle that fits inside a triangle;
- The radius of the incircle is: $r = A/s$
- The center of the circle can be found by the intersection of two **angle bisectors.**

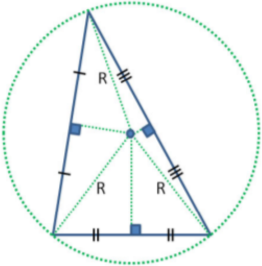
Radius of the Incircle: $r = \text{area}/s$

```
double rInCircle(double ab, double bc, double ca) {
  return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca)); }

double rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c), dist(c, a)); }
```

Finding the center of the Incircle of a triangle

```
int inCircle(point p1, point p2, point p3,
             point &ctr, double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0; // colinear points;
  line l1, l2; // compute these two angle bisectors

  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3),
                      ratio / (1 + ratio)));
  pointsToLine(p1, p, l1);                // bisector 1

  ratio = dist(p2, p1) / dist(p2, p3);
  p = translate(p1, scale(toVec(p1, p3),
                ratio / (1 + ratio)));    // bisector 2
  pointsToLine(p2, p, l2);

  areIntersect(l1, l2, ctr);              // find center (ctr)
  return 1;
}
```

Circumcircle of a Triangle



- The radius of the circumcircle in a triangle with sides a, b, c and area A is $R = \frac{abc}{4A}$;
- The radius R is also related to the **Law of Sines**:

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma} = 2R$$

- To find the center of the circumcircle:
  - Use a similar algorithm as the center of the incirle (last slide);
  - Instead of angle bisectors, use **perpendicular bisectors**;

Polygons – Definition and data structure

A polygon is a plane figure bounded by a finite sequence of line segments.

Polygon Representation

- In general, we store an array of points of the segments;
- We want to sort the points in CW or CCW order;
- Add the first point at the end of the array to avoid special cases;

```
// 6 points, entered in counter clockwise order;
vector<point> P;
P.push_back(point(1, 1)); // P0
P.push_back(point(3, 3)); // P1
P.push_back(point(9, 1)); // P2
P.push_back(point(12, 4)); // P3
P.push_back(point(9, 7)); // P4
P.push_back(point(1, 7)); // P5
P.push_back(P[0]); // important: loop back
```

Characteristics of a Polygon

Perimeter of a Poligon – add the distances of the segments

```
double perimeter(const vector<point> &P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++)
     // remember: P[0] = P[P.size()-1]
     result += dist(P[i], P[i+1]);
  return result; }
```

Area of a Poligon – half of the determinant of the XY matrix of segments

```
double area(const vector<point> &P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    x1 = P[i].x; x2 = P[i+1].x;
    y1 = P[i].y; y2 = P[i+1].y;
    result += (x1 * y2 - x2 * y1); }
  return fabs(result) / 2.0; }
```

Testing if a Polygon is Convex

- A convex polygon has no "holes";
- For any 2 points p_1, p_2 inside the polygon, segment is inside polygon too.

Easier Convex Testing: Every angle turns the same way

```
bool isConvex(const vector<point> &P) {
  // Returns true if every 3 neighb vertices turn the same way;
  int sz = (int)P.size();
  if (sz <= 3) return false; // Not a polygon

  bool isLeft = ccw(P[0], P[1], P[2]); // described earlier

  for (int i = 1; i < sz-1; i++)
    if (ccw(P[i],P[i+1],P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;            // not same direction as isLeft.

  return true;
}
```

Polygon – Test point inside the polygon

We can use the same idea to test if a point is inside the polygon: The direction of the point in relation to every edge should be the same.

Winding Algorithm Code for point in polygon detection

```
bool inPolygon(point pt, const vector<point> &P) {
  if ((int)P.size() == 0) return false;
  double sum = 0;

  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(pt, P[i], P[i+1]))
      sum += angle(P[i], pt, P[i+1]);       //left turn/ccw
      else sum -= angle(P[i], pt, P[i+1]);  //right turn/cw
  }

  return fabs(fabs(sum) - 2*PI) < EPS;
}
```

QUIZ: What happens if the point is at an edge segment?

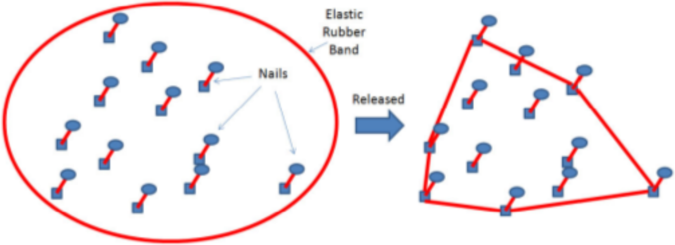
Polygon – Cutting

To cut *P* along a line *AB*, we separate the points in *P* to the left and right of the line.

```
point lineIntersectSeg(point p, point q, point A, point B) {
  double a=B.y-A.y; double b=A.x-B.x; double c=B.x*A.y-A.x*B.y;
  double u=fabs(a*p.x+b*p.y+c); double v=fabs(a*q.x+b*q.y+c);
  return point((p.x*v + q.x*u)/(u+v),
               (p.y*v + q.y*u)/(u+v)); }

vector<point> cutPolygon(point a, point b, const vector<point> &Q){
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
    if (i != (int)Q.size()-1)
      left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
    if (left1 > -EPS)
      P.push_back(Q[i]); //Q[i] is on the left of ab
    if (left1*left2 < -EPS) //edge (Q[i], Q[i+1]) crosses line ab
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b)); }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front()); // make P's first point = P's last point
  return P; }
```

Polygon – Convex Hull

- A common problem: Given a set of points S, what is the **smallest convex polygon** that includes all points in S?
- One way to find the Convex Hull: for each point $p \in S$, determine if the point is at the edge of the polygon (in the hull) or inside the polygon (not in the hull).
- We will introduce the $O(n \log n)$ algorithm "Graham's Scan"



Polygon – Graham's Scan

Helper Functions – sort two points based on their angle against the X axis

```
point pivot(0, 0);

bool angleCmp(point a, point b) {
  // special case: if collinear, choose closet to pivot;
  if (collinear(pivot, a, b)) // special case
    return dist(pivot, a) < dist(pivot, b);

  // calculate angle against the X axis:
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;

  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
}
```

Polygon – Graham's Scan

Convex Hull – Initializing the algorithm

```
vector<point> CH(vector<point> P) {
  int i, j, n = (int)P.size();
  // Special Case: Polygon with 3 points
  if (n <= 3) {
    if (!(P[0]==P[n-1])) P.push_back(P[0]);
    return P; }

  // Find Initial Point: Low Y then Right X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y ||
        (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;
  point temp = P[0]; P[0] = P[P0]; P[P0] = temp;
```

Polygon – Graham's Scan

Convex Hull – More initialization

```
// second, sort points by angle with pivot P0
pivot = P[0];
sort(++P.begin(), P.end(), angleCmp);

// S holds the Convex Hull
// We initialize it with first three points
vector<point> S;
S.push_back(P[n-1]);
S.push_back(P[0]);
S.push_back(P[1]);

// We start on the third point
i = 2;
```

Polygon – Graham's Scan

Convex Hull – Main Loop

Now that we selected a pivot and sorted the points, we test every three points (following the sort) if they are in the convex hull.

```
  while (i < n) {
    j = (int)S.size()-1;

    // If the next point is left of CH, keep it.
    // Else, pop the last CH point and try again.

    if (ccw(S[j-1], S[j], P[i]))
      S.push_back(P[i++]);
    else
      S.pop_back();
    }
  return S;
}                  // End Graham's Scan CH
```

Class Summary

In this class we saw several algorithms for calculating computational geometry constructs:

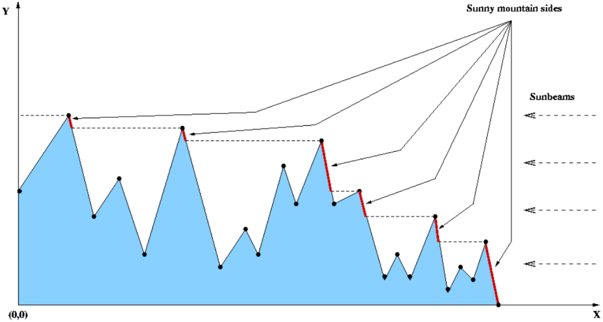
- Points and Lines and Intersections;
- Circles and Triangles, areas and intersections;
- Convex Hull testing and Convex Hull construction;

Harder geometry problems will require you to perform search on geometry constructs, graph search, etc. Having a library with the functions of this class ready will be useful!

This Week's Problems

- Sunny Mountains – Line and Points
- Waterfall – Line and Points
  Discussed in Class
- Elevator – Circles and Rectangles
- Colorful Flowers – Circles and Triangles
- Bounding Box – Circles, Triangles and Polygons
  Discussed in Class
- Soya Milk – Rectangle and Triangle
  Discussed in Class
- Trash Removal – Polygon Manipulation
- Board Wrapping – Convex Hull

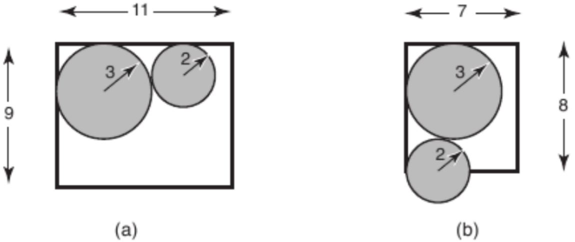
Sunny Mountains



- Given segment points as the input, calculate the area illuminated by the sun.

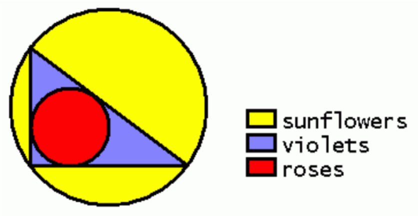
- **Hint:** This problem is about calculating line/segment intersections.
- **Hint:** Because the line is always HORIZONTAL, you can write a function that is simpler than the one I introduced in this class.

Elevator



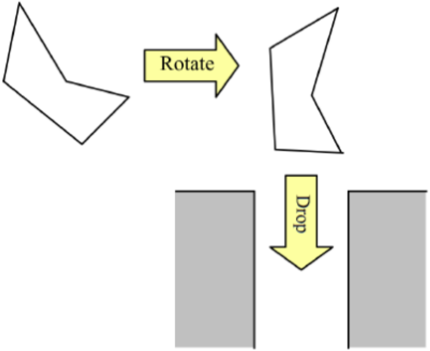
- **Input:** Size of the elevator and two Radius: R_1, R_2.
- **Output:** Do the two circles fit in the elevator? Y/N?
- **Hint:** The code is very simple, solve this program on paper first.

Colorful Flowers



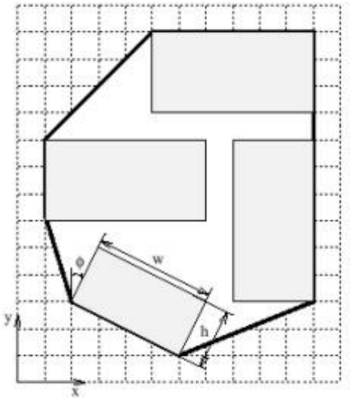
- **Input**: Three sides of blue triangle
- **Output**: Area of Yellow Zone, Blue Zone and Red Zone
- **Hint**: Practice the code for incircle and circumcircle!

Trash Removal



- What is the smallest trash box that can fit the polygon (trash)
- **Input**: Vertices of the polygon
- **Output**: Size of the smallest trash that fits the polygon
- **Hint**: It might help to think of the convex version of the polygon

Board Wrapping



- Convex Hull problem;
- Don't forget to rotate the rectangles correctly during input!

About these Slides

Image Credits I

[Page 9] Problem image from `https://onlinejudge.org`
[Page 11] Problem image from `https://onlinejudge.org`
[Page 17] Rotation Image from "Competitive Programming 3", Steven Halim
[Page 30] Circle Properties Images from Stephen Halim "Competitive Programming 3"
[Page 44] Convex Hull image by Steven Halim "Competitive Programming 3"