GB21802 - Programming Challenges

Week 8 - Computational Geometry

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

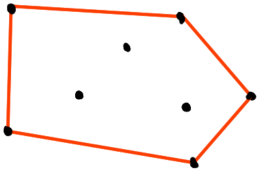
2019-06-14,17

Last updated June 12, 2019

Computational Geometry
What is it?

Computational Geometry problems involve answering questions about lines, points and angles. Some example questions:

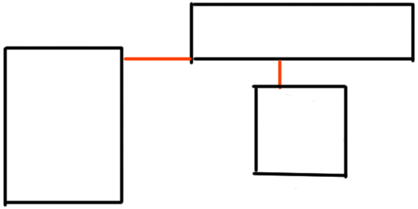
Given N points $(s_1, s_2, s_3, \ldots, s_N)$, what is the area of the poligon that covers all the points?



Computational Geometry
What is it?

Computational Geometry problems involve answering questions about lines, points and angles. Some example questions:

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

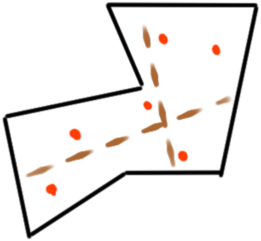


Computational Geometry

What is it?

Computational Geometry problems involve answering questions about lines, points and angles. Some example questions:

Given a polygon, and N points, what is the line that divides the polygon in equal areas, so that the same number of points are in each area?



Computational Geometry
The good and the bad

- Good: Geometry problems are fun
- Good: You have to draw pretty pictures
- Good: Mostly algorithms from high school
- Good: Code is highly re-usable

- Bad: You have to write a lot of code (in the beginning!)
- Bad: Very easy to get WE...

Easy Mistakes in Geometry Problems

Problem 1 – Special Cases

- Multiple points in the same place;
- Collinear points;
- Vertical lines;
- Parallel Lines;
- Intersection at end of segment;
- etc;

Problem 2 – Precision Errors

- Functions require many multiplications and divisions;
- Easy to propagate floating point errors;

Easy Mistakes in Geometry Problems

Dealing Special Cases

- Make sure to add special cases to your library functions;

Solving Precision Errors

- If possible, convert all values to integers
- Use an EPSILON constant for comparisons:

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

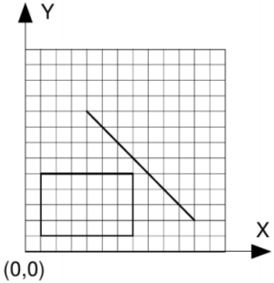
Class Outline

- Example Problems;
- Basic Geometric Functions;
- Circles;
- Triangles;
- Polygons;

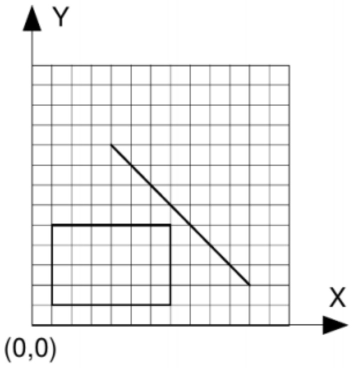
Problem Example: UVA 191 – Intersection

Summary

- **Input**
  A rectangle and a line:
  xstart ystart xend yend xleft ytop xright ybottom
- **Output**
  T - if the line intersects the rectangle
  F - if the line does not intersect the rectangle



Problem Example: UVA 191 – Intersection



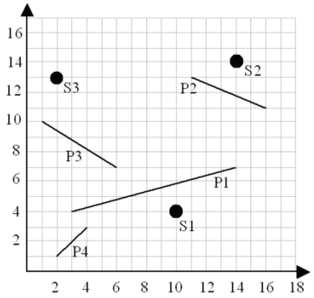
Steps to calculate the solution:

- Test if p_1 or p_2 are inside the rectangle;
- Test if the segment intersects a side of the rectangle;
- (optional) make a bullet hell game;

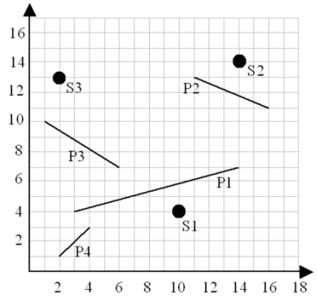
Problem Example: UVA – Waterfalls

Summary

- **Input**
  List of line segments in the waterfall
  List of water sources
- **Output**
  X position where each water source falls



Problem Example: UVA 833 – Waterfalls



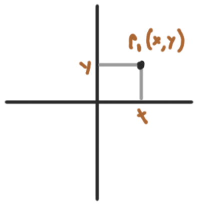
For each water source:

- Identify all segments that it intersects with;
- Select the highest segment;
- Move the source to the bottom of the segment;
- Repeat;

Many opportunities for pruning and pre-computing! (if necessary)

Implementing Graphical Problems

Point Representation



Point Representation

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

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

Implementing Graphical Problems

Comparing and Sorting Points

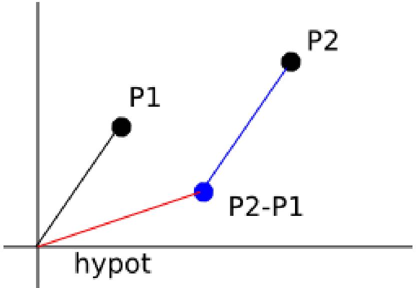
Point Comparison

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

   // Sorting by coordinate
   bool operator < (point other) const {
      if (fabs(x - other.x) > EPS)
         return x < other.x;
      return y < other.y; }

   // Equality testing -- Note the use of EPS
   bool operator == (point other) const {
      return (fabs(x - other.x) < EPS &&
             (fabs(y - other.y) < EPS)); }
   }
```

Basic Library – Points 3

Most common distance measure: Euclidean distance. Sometimes Manhattan distance (Taxicab distance) is also used.

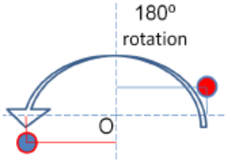
```
#define hypot(dx,dy) sqrt(dx*dx + dy*dy)

double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y); }

double taxicab(point p1, point p2) {
  return fabs(p1.x - p2.x) + fabs(p1.y - p2.y); }
```



Basic Library – Points 4

Rotating a point around the origin

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

// theta is in degrees
point rotate(point p, double theta) {
   double rad = DEG_to_RAD(theta);
   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: What do you do if you want to rotate a point around x_0, y_0?

Basic Library – Lines 1

There are many ways to specify a line:

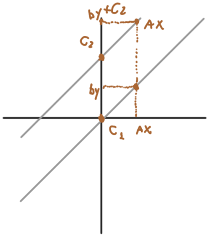
- $ax + by + c = 0$ – useful for most cases.
- $y = mx + c$ – useful for angle manipulation, but special cases
- x_0, y_0, x_1, y_1 – two points, not very useful for programming.

Point to 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;}
}
```

Basic Library – Line 2

- 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;
- Remember that we force b to be 0 or 1;



Parallel and identical lines

```
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); }
```

Basic Library – Line 3

The **intersection** point x_I, y_I is where two lines meet. We can find this point by solving the following system of linear equations:

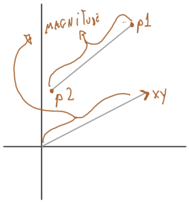
$$\begin{aligned} a_1 x + b_1 y + c_1 &= 0 \\ a_2 x + b_2 y + c_2 &= 0 \end{aligned}$$

Computing the intersection

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

Basic Library – Vectors

- A Vector indicates direction and length;
- Represented as a *x*, *y* point in relation to the origin;
- Operations: Scale, Translation, Addition, Product;

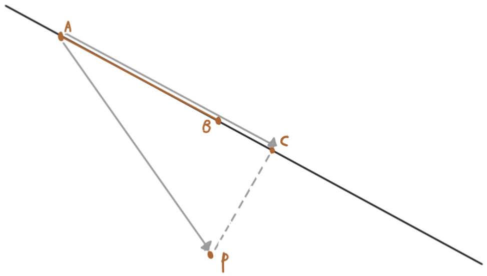


```
struct vec { double x, y;
    vec(double _x, double _y) : x(_x), y(_y) {} };

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

Given a point *p* and a line *l*, the distance between the point and the line is the distance between *p* and the *c*, the closest point in *l* to *p*.

We can calculate the position of *c* by taking the projection of $\bar{ac}$ into *l* (*a*, *b* are points in *l*).



Distance between point and line

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

// Calculates distance of p from line, given
// a,b different points in the line.
double distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u * ab
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));
  // translate a to c
  return dist(p, c); }
```

Distance between point and segment

If we have a segment *ab* instead of a line, the procedure to calculate the distance is similar, but we need to test if the intersection point falls in the segment.

```
double distToLineSegment(point p, point a,
                         point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(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); }

  return distToLine(p, a, b, c); }
```

Angles between segments

angle between two segments ao and ob

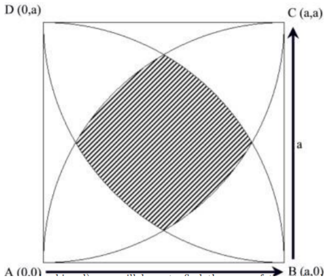
```
#import <cmath>

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

Left/Right test: We can calculate the position of point *p* in relation to a line *l* using the cross product.

Take *q*, *r* points in *l*. Magnitude of the cross product *pq* x *pr* being positive/zero/negative means that $p \rightarrow q \rightarrow r$ is a 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;
```

Problem Example: UVA 10589 Area



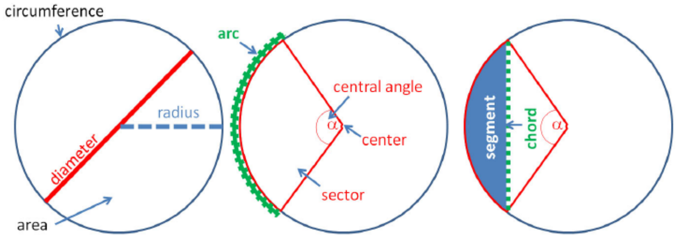
- What is the area of the shaded part of the rectangle?
- You are given the radius of 4 circles, each centered in the corners of the rectangle.

Circles

- A circle is defined by its center (a, b) an its radius r
- The circle contains all points such (x, y) such as $(x - a)^2 + (y - b)^2 \leq r^2$

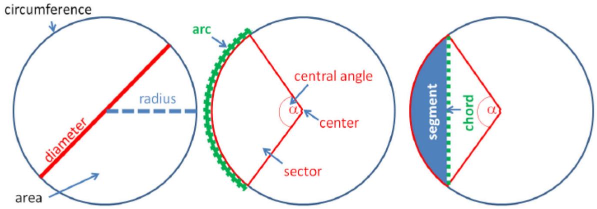
```
int insideCircle(point_i p, point_i c, int r) {
   int dx = p.x-c.x, dy = p.y-c.y;
   int Euc = dx*dx + dy*dy, rSq = r*r;
   return Euc < rSq ? 0 : Euc == rSq ? 1 : 2;
   // 0 - inside, 1 - border, 2- outside
}
```

Circles (2)



- If you are not given π, use $pi = 2{*}\text{acos}(0.0)$;
- Diameter: $D = 2r$; Perimeter/Circumference: $C = 2\pi r$; Area: $A = \pi r^2$;
- To calculat the Arc of an angle α (in Degrees), $\frac{\alpha}{360} * C$;

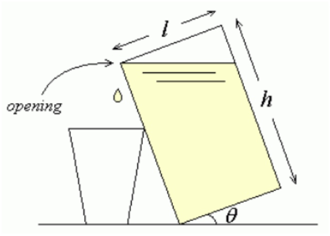
Circles (3)



- A chord of a circle is a segment composed of two points in the circle's border. A circle with radius r and angle α degrees has a chord of length sqrt($2r^2(1 - \cos\alpha)$)
- A Sector is the area of the circle that is enclosed by two radius and and arc between them. Area is: $\frac{\alpha}{360}A$
- A Segment is the region enclosed by a chord and an arc.

Example: UVA 11909 - Soya milk

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



Example: UVA 10577 - Bounding Box

Given three vertices of a regular polygon, calculate the minimal square necessary to cover the polygon.

Hint: You don’t actually need to calculate any polygons

Triangle Basics

Any 2 dimensional polygon can be expressed as a combination of triangles. So triangles are important constructs in computational geometry.

Common Characteristics

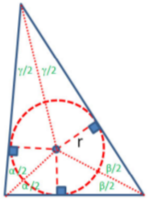
- Triangle Inequality: Sides a, b, c obey $a + b > c$
- Triangle Area: Be b one side of the triangle and h its height, $A = 0.5bh$
- Perimeter: $p = a + b + c$
- Semiperimeter: $s = 0.5p$

Heron's Formula

We can calculate the area of a triangle based on its sides:

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

Incircle Triangle



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

```
def radiusInCircle(p1,p2,p3):
   ab, bc, cd = dist(p1,p2),dist(p2,p3),
                     dist(p3,p1)
   A = area(ab,bc,ca) % Heron's formula
   P = ab+bc+ca
   return A/(0.5*P)
```

Finding the center point of the Incircle

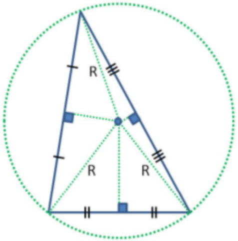
- Check that the three points are not colinear;
- Find the bisection *AP* of the *AB*-*AC* angle;
  - Calculate the point *P* in *BC* that bisects *A*
  - The proportion of *BP* is $(AB/AC)/(1 + AB/AC)$
- Find the bisection BP' of the *BA*-*BC* angle;
- Fint the intersection of *AP*-BP'

Incircle Triangle

Calculating the Center (Code)

```
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);
  ratio = dist(p2, p1) / dist(p2, p3);
  p = translate(p1, scale(toVec(p1, p3),
                ratio / (1 + ratio)));
  pointsToLine(p2, p, l2);
  areIntersect(l1, l2, ctr);
  return 1; }
```

Excircle Triangle



Radius of the excircle

A triangle with sides a, b, c and area A has an excircle with radius: $R = abc/4A$.

The center of the excircle is the intersection of the *perpendicular bisectors*.

Trigonometry

- Law of Cosines:
  $c^2 = a^2 + b^2 - 2ab\cos(\gamma)$
  $\gamma = \text{acos}((a^2 + b^2 - c^2/2ab)$
- Law of Sines: (R is the radius of the excircle):
  $a/\sin(\alpha) = b/\sin(\beta) = c/\sin(\gamma) = R$

Polygons

Definition

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

Polygon Representation

- In general we want to sort the points in CW or CCW order
- Adding the first point at the end of the array helps 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
```

Polygon Algorithms

Perimeter of a Poligon – sum of distances

```
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 the determinant of the XY matrix

```
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; }
```

Polygon – Concave and Convex check

Convex Polygons

Has NO line segment with ends inside itself that intersects its edges.

Another definition is that all inside angles “turn” the same way.

Testing for a convex polygon

```
bool isConvex(const vector<point> &P) {
  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; // works for both left and right
      // different sign -> this polygon is concave
  return true; }
```

Polygon – Testing Inside or outside

There are many ways to test if a point *P* is in a polygon.

- Winding Algorithm: Sum the angles of all angles *APB* (*A*, *B*) are points in the polygon. If the sum is 2π. Point is in polygon.
- Ray Casting Algorithm: Draw an segment from *P* to infinity, and count the number of polygon edges crossed. Odds: Inside. Even: Outside.

Winding Algorithm Code

```
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; }
```

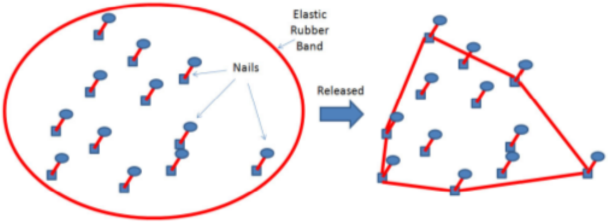
Polygon – Cutting

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

Given a set of points S, the convex hull is the polygon P composed of a subset of S so that every point of S is either part of P, or inside it.



The main algorithm for calculating the convex hull is *Graham's Scan*.

It's idea is to test each point angle order, to see if the point belongs to the hull.

Polygon – Graham's Scan

Helping Functions

```
point pivot(0, 0);

bool angleCmp(point a, point b) { // angle-sorting
  if (collinear(pivot, a, b)) // special case
    return dist(pivot, a) < dist(pivot, b);
  // check which one is closer to 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 or 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

```
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; }
```

This Week's Problems

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