Geometric Algorithms

overview
primitives
convex hull algorithms
context

Geometric algorithms

Important and far-reaching applications

- models of physical world
  examples: maps, architecture, medical imaging
- computer graphics
  examples: movies, games, virtual reality
- mathematical models
  stay tuned

Ancient mathematical foundations, but
most geometric algorithms are less than 30 years old

Knowledge of fundamental algorithms is critical

- use them directly
- use the same design strategies for harder problems
- learn how to compare and evaluate algorithms

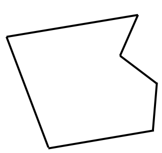
Warning: intuition may not be helpful

Humans have spatial intuition in 2D and 3D: computers do not!

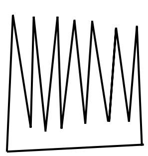
Example: Is a given polygon convex?

we see these

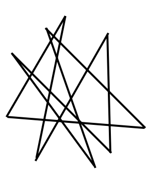
programs see these



1	6	5	8	7	2
7	8	6	4	2	1



1	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	2	18	4	18	4	19	4	19	4	20	3	20	3	20



1	10	3	7	2	8	8	3	4
6	5	15	1	11	3	14	2	16

Elementary geometric primitives (2D)

Point

- two numbers (x, y)
  `#typedef struct {double x; double y;} Point;`

Line

- two numbers a and b [ax + by = 1] ← lines through origin are exceptional

Line segment

- four numbers (x1, y1) and (x2, y2)
- two points p0 and p1
  `#typedef struct {Point x; Point y;} LineSegment;`

Polygon

- sequence of points
  `Point p[N];`

No shortage of other geometric shapes
triangle, square, circle, quadrilateral, parallelogram, ...

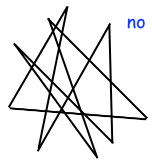
3D and higher dimensions more complicated

Building layers of abstraction

Typical scenario in algorithm design: Use a more primitive operation!

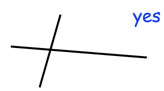
Example:

Is a given polygon simple?



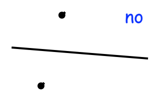
⬇

Do two given line segments intersect?



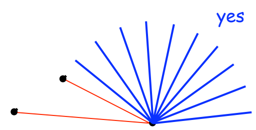
⬇

Are two given points on the same side of a given line?



⬇

Is the route connecting three given points a ccw turn?

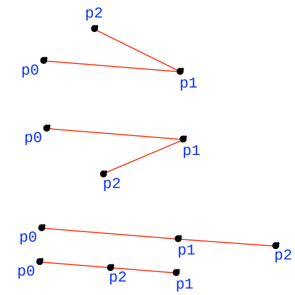


CCW implementation

Input: points `p0`, `p1`, and `p2`

Output:

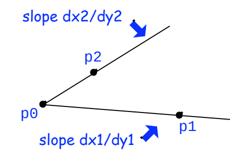
- `1` if `p0-p1-p2` is a ccw turn
- `-1` if `p0-p1-p2` is a cw turn
- `0` if `p0`, `p1`, `p2` are collinear



Approach: compare slopes

```
int ccw(Point p0, Point p1, Point p2)
  {
    int dx1, dx2, dy1, dy2;
    dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
    dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
    if (dx1*dy2 > dy1*dx2) return 1;
    if (dx1*dy2 < dy1*dx2) return -1;
    return 0;
  }
```

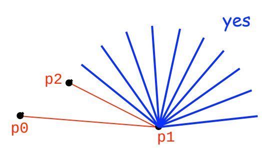
slopes are equal



Layers of abstraction example (continued)

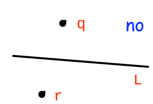
Is the route connecting `p0`, `p1`, and `p2` a ccw turn?

```
ccw(p0, p1, p2)
```



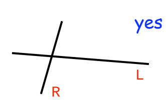
Are points q and r on the same side of line L?

```
ccw(L.p0, L.p1, q) == ccw(L.p0, L.p1. r)
```



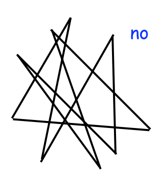
Do line segments L and S intersect?

```
!same(L.p0, L.p1, S) && !same(S.p0, S.p1, L)
```



Is a given polygon simple?

```
for(i = 0; i < N; i++)
  for(j = i+1; j < N; j++)
    if (intersect(p[i], p[j]) return 0;
return 1;
```



Stay tuned (next lecture)
for faster implementation

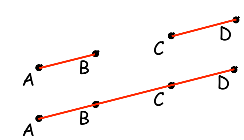
Line-segment intersection implementation bug

Still not quite right!

Bug in degenerate case with four collinear points

Does AB intersect CD?

- on the line in the order ABCD: NO
- on the line in the order ACDB: YES



Need more careful CCW implementaton

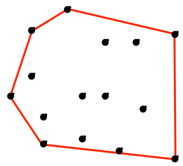
- more work when `dx1*dy2 == dx2*dy1` (see book)

Lessons:

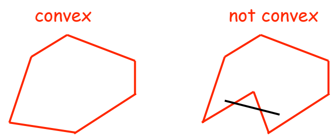
- geometric primitives are tricky to implement
- can't ignore degenerate cases

Convex hull of a point set

Convex hull: smallest polygon enclosing a given set of points



A polygon is convex iff every line whose endpoints are within the polygon falls entirely within the polygon



Lemma: Hull must be convex

Running time of convex hull algorithms can depend on

- N: number of points
- M: number of points on the hull
- point distribution

Incremental convex hull algorithms

Idea: consider points one by one

- next point inside current hull—ignore
- next point outside current hull—update

Two subproblems to solve

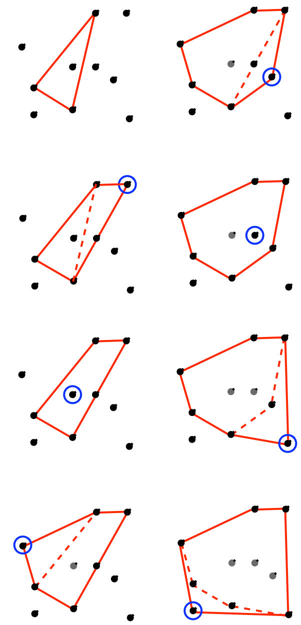
- test if point inside or outside polygon
- update hull for outside points

Both subproblems

- brute force: O(M) to check all hull points
- can be improved to O(log M) with binary search
- relatively cumbersome to code

Randomize: take points in random order

Total running time: O(N log M)



Sweep-line convex hull algorithm

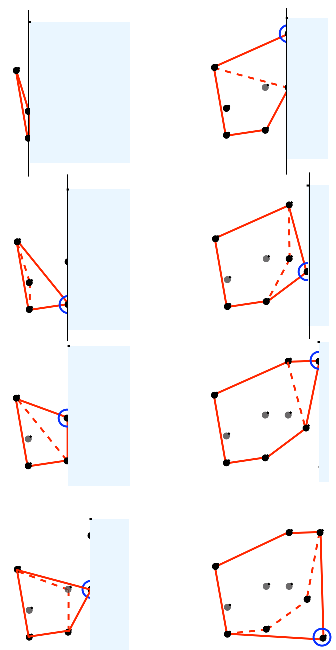
Idea: presort on x for incremental algorithm

Equivalent to imagining sweep line moving from left to right through points

plus: eliminates “inside” test

minus: have to pay cost of sort

Total cost: O(N log N)

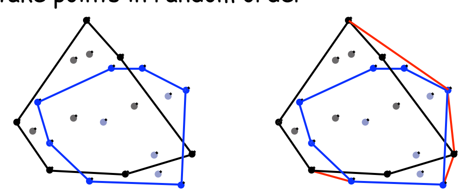


Divide-and-conquer convex hull algorithms

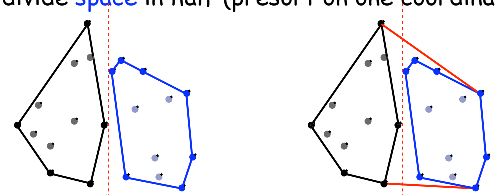
Divide point set into two halves

- solve subproblems recursively
- merge results

Idea 1: take points in random order



Idea 2: divide space in half (presort on one coordinate)

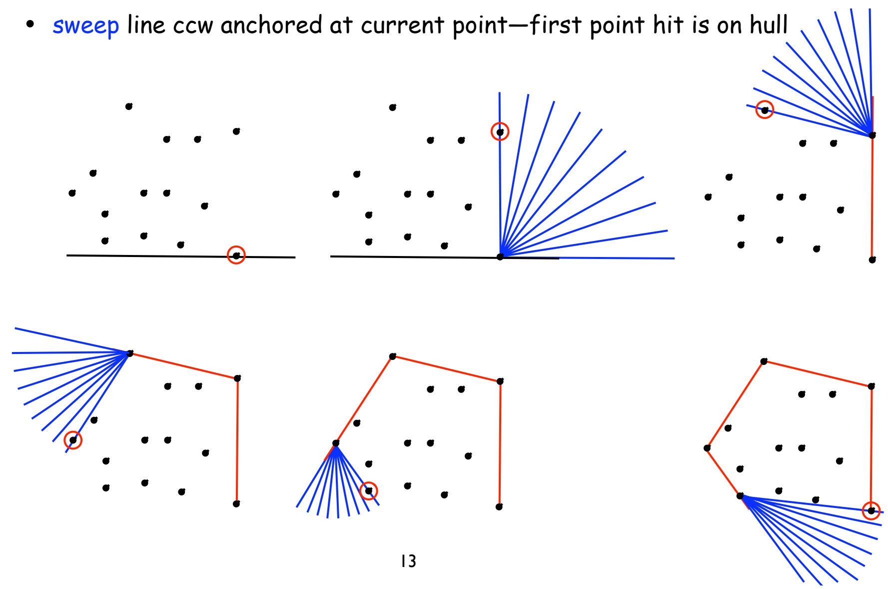


Both O(N log N) but relatively cumbersome to code

Package-wrapping convex hull algorithm

Idea:

- point with lowest y coordinate is on the hull
- sweep line ccw anchored at current point—first point hit is on hull



Implementation of package-wrapping algorithm

Input: polygon (represented as an array of N points)

Output: M (array rearranged such that first M points are convex hull)

```
int wrap(Point p[], int N)
  { int i, min, M; double th, v; Point t;
    for (min = 0, i = 1; i < N; i++)
      if (p[i].y < p[min].y) min = i;
    p[N] = p[min]; th = 0.0;
    for (M = 0; M < N; M++)
    {
      t = p[M]; p[M] = p[min]; p[min] = t;
      min = N; v = th; th = 360.0;
      for (i = M+1; i <= N; i++)
        if (theta(p[M], p[i]) > v)
          if (theta(p[M], p[i]) < th)
            { min = i; th = theta(p[M], p[min]);}
      if (min == N) return M;
    }
  }
```

find point with min y coordinate

implementation of theta omitted (can use slope)

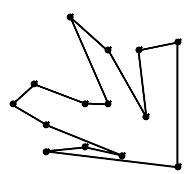
find min angle > v

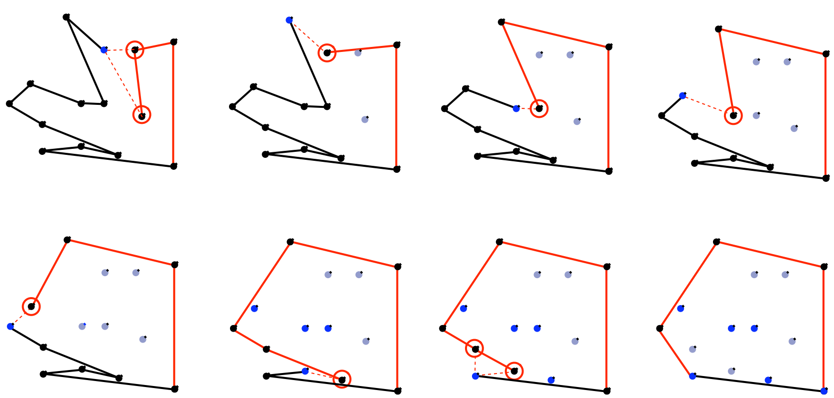
2D analog of selection sort: O(NM) running time

Graham scan convex hull algorithm

Idea:

- sort points by angle to get simple closed polygon
- scan polygon—discard points causing cw turn





Implementation of Graham scan algorithm

Input: polygon (represented as an array of N points) points in p[1]...p[N]

Output: M (array rearranged such that first M points are convex hull)

```
int grahamscan(Point p[], int N)
  { int i, min, M; Point t;
    for (min = 1, i = 2; i <= N; i++)
      if (p[i].y < p[min].y) min = i;
    for (i = 1; i <= N; i++)
      if (p[i].y == p[min].y)
        if (p[i].x > p[min].x) min = i;
    t = p[1]; p[1] = p[min]; p[min] = t;
    quicksort(p, 1, N);
    p[0] = p[N];
    for (M = 3, i = 4; i <= N; i++)
      {
        while (ccw(p[M],p[M-1],p[i]) >= 0) M--;
        M++; t = p[M]; p[M] = p[i]; p[i] = t;
      }
    return M;
  }
```

swap "lower left" point with first

implementation of less uses angle with p[1]

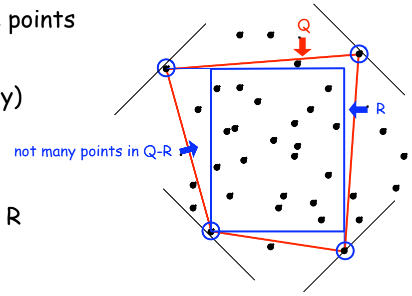
p[0] is sentinel

back up to include i on hull

add i to putative hull

Total cost: O(N log N) (for sort).

Quick-elimination convex hull algorithms



Idea: fast test to eliminate most inside points

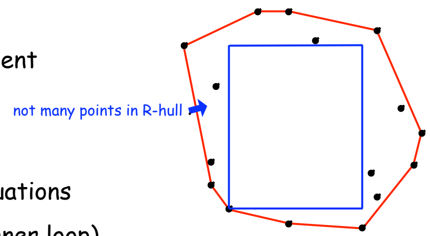
quick: use quadrilateral Q
min (x+y), max(x+y), min(x-y), max(x-y)
quicker: use inscribed rectangle R

Three-phase algorithm

- pass through all points to compute R
- eliminate points inside R
- find convex hull of remaining points

Option 1: use recursion ("quickhull")

- relatively cumbersome to implement
- O(N) worst case



Option 2: use Graham scan

- few points remaining in many situations
- O(N + M lg M) avg case (+ fast inner loop)

Convex hull algorithms cost summary

"Guaranteed" asymptotic cost to find M-point hull in N-point set

Package wrap	NM
Graham scan	N log N (sort time)
Divide and conquer *	N log N
Quick elimination *	N
Incremental elimination	N log M
Sweep line	N log N (sort time)

* assumes "reasonable" known point distribution
* leading coefficient higher than for sorting

How many points on hull?

- Worst case: N
- Average case: difficult problems in stochastic geometry
  - uniform in a convex polygon with O(1) edges: log N
  - uniform in a disc: $N^{1/3}$

Higher dimensions

Multifaceted (convex) polytope encloses points

NOT a simple object

- vertices, edges, facets
- return extreme points (hull vertices)—no natural order

Example: N points d dimensions

- d=2: convex hull
- d=3: Euler's formula (v - e + f = 2)
- d>3: exponential number of facets at worst

Some of the same approaches work (costs higher)

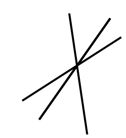
- Package-wrap
- Divide-and-conquer
- Randomized
- Interior elimination

Context: mathematics

Geometric models of mathematical problems extend impact of geometric algs far beyond direct application to physical models

Example 1:

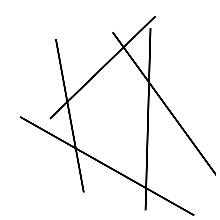
geometric problem	mathematical equivalent
intersect two lines (2D)	solve 2 equations in 2 unknowns
intersect three planes (3D)	solve 3 equations in 3 unknowns



algorithm: gaussian elimination

Example 2:

geometric problem	math equivalent
find convex polytope defined by intersecting half-planes	solve simultaneous inequalities
is given point inside polytope?	linear programming



algorithm: simplex

Vast number of applications (stay tuned)

Context: algorithm design paradigms

Draw from knowledge about fundamental algorithms

Design and use levels of abstraction

- use fundamental algorithms and data structures
- know their performance characterisitics

Carefully implement primitives

Recognize intrinsically difficult problems

For many important problems

- classical approaches give good algorithms
- need research to find best algorithms
- no excuse for using dumb algorithms

all possibilities	double recursion	2^N
brute force	nested for loops	N^2
divide-and-conquer	recursion, trees	N log N
elegant idea	single for loop	N
randomization	random choices	N