Important applications involve geometry

- models of physical world
- computer graphics
- mathematical models

Ancient mathematical foundations
Most geometric algorithms less than 25 years old

Knowledge of fundamental algorithms is critical

- use them directly
- use the same design strategies
- know how to compare and evaluate algs
Warning: intuition may mislead

Humans have spatial intuition in 2D and 3D
- computers do not!
- neither have good intuition in high dimensions

**Ex:** Is a polygon convex?

we think of this     alg sees this     or even this
Ex: Find intersections among set of rectangles

- we think of this

algorithm sees this
Geometric algorithms: overview

New primitives
- points, lines, planes; polygons, circles

Primitive operations
- distance, angles
- “compare” point to line
- do two line segments intersect?

Problems extend to higher dimensions
- (algorithms sometimes do, sometimes don’t)

Higher level intrinsic structures arise

Basic problems
- intersection
- proximity
- point location
- range search
Approaches to solving geometric problems

- incremental (brute-force)
- divide-and-conquer
- sweep-line algs
- multidimensional tree structures
- randomized algs
- discretized algorithms
- online and dynamic algs
Algorithm design paradigms

Draw from knowledge about fundamental algs
Move up one level of abstraction
  • use fundamental algs and data structures
  • know their performance characteristics

More primitives lead to wider range of problems
Some problems too complex to admit simple algorithms

For many important problems
  • classical approaches give good algorithms
  • need research to find “best” algorithms
  • no excuse for using “dumb” algorithms
Progression of algorithm design (oversimplified)

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>all possibilities</td>
<td>double recursion</td>
</tr>
<tr>
<td>brute force</td>
<td>nested for loops</td>
</tr>
<tr>
<td>divide-and-conquer</td>
<td>recursion, trees</td>
</tr>
<tr>
<td>elegant idea</td>
<td>1 &quot;for&quot; loop</td>
</tr>
<tr>
<td>randomization</td>
<td>random choices</td>
</tr>
</tbody>
</table>

Many examples in geometric algorithms
Geometric primitives (2D)

POINT

two numbers \((x, y)\)

LINE

two numbers \(a\) and \(b\) \([ax + by = 1]\)

LINE SEGMENT

four numbers \((x_1, y_1)\) \((x_2, y_2)\)

POLYGON

sequence of points

No shortage of other geometric shapes

TRIANGLE

SQUARE

CIRCLE

- 3D and higher dimensions more complicated
Building algorithms from geometric primitives

First, need good implementations of primitives!
  • is polygon simple?
  • is point on line?
  • is point inside polygon?
  • do two line segments intersect?
  • do two polygons intersect?

Algorithms search through SETS of primitives
  • all points in specified range
  • closest pair in set of points
  • intersecting pairs in set of line segments
  • overlapping areas in set of polygons
Do two line segments intersect?

To implement \textsc{INTERSECT}(l_1, l_2)

- use simpler primitive \textsc{SAME}(p_1, p_2, l):
  Given two points \( p_1 \), \( p_2 \) and a line \( l \),
  are \( p_1 \) and \( p_2 \) on the same side of \( l \)?

To implement \textsc{SAME}

- use simpler primitive \textsc{CCW}(p_1, p_2, p_3):
  Given three points \( p_1 \), \( p_2 \), \( p_3 \),
  is the route \( p_1\text{-}p_2\text{-}p_3 \) a ccw turn?

two ccw tests to implement \textsc{SAME}
four ccw tests to implement \textsc{INTERSECT}
CCW implementation

compare slopes

- less:
- greater:
- equal: points are collinear

```c
#typedef struct point POINT
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;
}
```
CCW implementation (continued)

Still not quite right! Bug in degenerate case

- four collinear points
- Does AB intersect CD?
  - on the line in the order ABCD: NO
  - on the line in the order ACDB: YES

Can't just return 0 if dx1*dy2 = dx2*dy1 (see book)

CCW is an important basic primitive

Ex: is point inside convex N-gon? N CCW tests

Lesson:
- geometric primitives are tricky to implement
- can't ignore degenerate cases
Convex hull of a point set

Basic property of a set of points

CONVEX HULL:
- smallest convex polygon enclosing the points
- shortest fence surrounding the points
- intersection of halfplanes defined by point pairs

Running time of algorithm can depend on
- $N$: number of points
- $M$: number of points on the hull
- point distribution
Package-wrap algorithm

Operates like selection sort

Abstract idea
- sweep line anchored at current point CCW
- first point hit is on hull

Implementation
- compute angle to all points
- pick smallest angle larger than current one
int wrap(POINT p[], int N) {
    int i, min, M; float th, v; struct 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;
    }
}

Use pseudo-angle theta to save time (see text)
Package-wrap example
Graham Scan

Sort points on angle with bottom point as origin
- forms simple closed polygon

Proceed through polygon
- discard points that would cause a CW turn

```c
int grahamscan(struct point p[], int N)
{
    int i, min, M; struct 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;
}
```
Graham scan example
Divide-and-conquer convex hull algorithms

divide points

divide space
Consider next point
- if inside hull of previous points, ignore
- if outside, update hull

Two subproblems to solve
- test if point inside or outside polygon
- update hull for outside points

Both subproblems
- can be solved by looking at all hull points
- can be improved with binary search

Randomized algorithm
- consider points in random order
- $N + M \log M$
"Sweep line" convex hull algorithm

Sort points on x-coordinate first

Eliminates "inside" test

Total time proportional to \( N \log N \) (for sort)
Quick elimination

Improve the performance of any convex hull algorithm by quickly eliminating most points (known not to be on the hull).

Use points at "corners": max, min x+y, x-y

Check if point inside quadrilateral: four CCW tests
Check if point inside rectangle: four comparisons

Almost all points eliminated if points random
  - number of points left proportional to $N^{(1/2)}$

LINEAR algorithm
Summary of 2D convex hull algos

Package wrap
  • NM

Graham scan
  • \( N \log N \) (sort time)

Divide-and-conquer
  • \( N \log N \) (with work)

Quick elimination
  • \( N \) (fast average-case)

One-by-one elimination
  • \( N \log M \)

Sweep line
  • \( N \log N \) (sort time)

How many points on the hull?
Worst case: \( N \)
Average case: depends on distribution
  • uniform in a convex polygon: \( \log N \)
  • uniform in a circle: \( N^{(1/3)} \)

requires understanding of basic properties of DATA
Higher dimensions

Multifaceted (convex) polytope encloses points

NOT a simple object

Ex: 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

EXTREME POINTS
- return points on the hull, not necc in order

Package-wrap
Divide-and-conquer
Randomized
Interior elimination
Geometric models of mathematical problems

Impact of geometric algs extends far beyond physical models

Geometric problem
- find point where two lines intersect in 2D
- find point where three planes intersect in 3D

Mathematical equivalent
- solve simultaneous equations
- algorithm: gaussian elimination

Geometric problem
- find convex polytope defined by intersecting half-planes
- find vertex hit by line of given slope moving in from infinity

Mathematical equivalent
- LINEAR PROGRAMMING
- algorithm: SIMPLEX (stay tuned)
Linear programming example

Maximize $a+b$ subject to the constraints

- $b - a < 5$
- $a + 4b < 45$
- $2a + b < 27$
- $3a - 4b < 24$
- $a > 0$
- $b > 0$