Geometric Algorithms

overview
primitives
convex hull algorithms
context

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

Humans have spatial intuition in 2D and 3D: computers do not!
Example: Is a given polygon convex?

Point
• two numbers (x, y)

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

Polygon
• sequence of points

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?

Layers of abstraction example (continued)

Is the route connecting \( p_0, p_1, \) and \( p_2 \) a ccw turn?

\[ \text{ccw}(p_0, p_1, p_2) \]

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

\[ \text{ccw}(L.p0, L.p1, q) = \text{ccw}(L.p0, L.p1, r) \]

Do line segments \( L \) and \( S \) intersect?

\[ \text{same}(L.p0, L.p1, S) \land \text{same}(S.p0, S.p1, L) \]

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

CCW implementation

Input: points \( p_0, p_1, \) and \( p_2 \)

Output:

1 if \( p_0-p_1-p_2 \) is a ccw turn

-1 if \( p_0-p_1-p_2 \) is a cw turn

0 if \( p_0, p_1, p_2 \) are collinear

Approach: compare slopes

\[
\begin{align*}
\text{int } \text{ccw}(\text{Point } p0, \text{Point } p1, \text{Point } p2) \\
\{ \\
\quad \text{int } dx1, dx2, dy1, dy2; \\
\quad dx1 = p1.x - p0.x; \quad dy1 = p1.y - p0.y; \\
\quad dx2 = p2.x - p0.x; \quad dy2 = p2.y - p0.y; \\
\quad \text{if } (dx1*dy2 > dy1*dx2) \text{ return } 1; \\
\quad \text{if } (dx1*dy2 < dy1*dx2) \text{ return } -1; \\
\text{return } 0; \quad \Leftarrow \text{slopes are equal}
\}
\end{align*}
\]

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 implementation

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

Lessons:

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

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: \(O(N \log N)\)

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: \(O(N \log M)\)

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)

```c
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;
    }
}
```

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)

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

```c
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 = ... >= 0) M--;
    M++; t = p[M]; p[M] = p[i]; p[i] = t;
    } return M;
```

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
  - $O(N + M \log M)$ avg case (+ fast inner loop)

---

**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

---

**Convex hull algorithms cost summary**

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package wrap</td>
<td>$NM$</td>
</tr>
<tr>
<td>Graham scan</td>
<td>$N \log N$ (sort time)</td>
</tr>
<tr>
<td>Divide and conquer</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Quick elimination</td>
<td>$N$</td>
</tr>
<tr>
<td>Incremental elimination</td>
<td>$N \log M$</td>
</tr>
<tr>
<td>Sweep line</td>
<td>$N \log N$ (sort time)</td>
</tr>
</tbody>
</table>

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

---

**Context: mathematics**

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

**Example 1:**

<table>
<thead>
<tr>
<th>geometric problem</th>
<th>mathematical equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersect two lines (2D)</td>
<td>solve 2 equations in 2 unknowns</td>
</tr>
<tr>
<td>intersect three planes (3D)</td>
<td>solve 3 equations in 3 unknowns</td>
</tr>
</tbody>
</table>

algorithm: **gaussian elimination**

**Example 2:**

<table>
<thead>
<tr>
<th>geometric problem</th>
<th>math equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>find convex polytope defined by intersecting half-planes</td>
<td>solve simultaneous inequalities</td>
</tr>
<tr>
<td>is given point inside polytope?</td>
<td>linear programming</td>
</tr>
</tbody>
</table>

algorithm: **simplex**

Vast number of applications (stay tuned)
Draw from knowledge about fundamental algorithms

Design and use levels of abstraction
  • use fundamental algorithms and data structures
  • know their performance characteristics

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

<table>
<thead>
<tr>
<th>Context: algorithm design paradigms</th>
</tr>
</thead>
<tbody>
<tr>
<td>all possibilities</td>
</tr>
<tr>
<td>brute force</td>
</tr>
<tr>
<td>divide-and-conquer</td>
</tr>
<tr>
<td>elegant idea</td>
</tr>
<tr>
<td>randomization</td>
</tr>
</tbody>
</table>