

Data Structures and Algorithms II

Lecture 15: Geometric Algorithms

- Motivation
- Representing points, lines and polygons
- Line intersection
- Inclusion in a polygon

1

Motivation

Many applications involve manipulating *geometric objects*: points, lines, polygons and 3d shapes:

- Drawing packages
- User interfaces
- CAD tools
- Geographic information systems.

Geometric problems easy for humans, but algorithms hard. e.g., given complex polygon, how do we determine if a point is within it?

Important in many applications: which object has the user clicked on?

We need efficient algorithms for tasks like this.

3

Some Notes on Assignment 1

- Most people used KMP - this doesn't make the search much more efficient than brute force, as even with an "alphabet" of only 4 characters, most initial character comparisons fail.
- Generally (in this and current assignment) try and avoid having i/o in general purpose functions (e.g., search, ADT ops,...). Instead, pass in/back values/error codes.
- For error messages better practice to use cerr stream, not cout. (Then sensible operating systems can easily, for example, direct all error messages to a file).
- Converting int to char: Can either use atoi library function, or write your own function using, e.g., `int(mychar) - int('0')`.

2

Representing Points, Lines and Polygons

Basic data structures easy:

```
struct point
{ float x, y;};
struct line
{ point p1, p2;};
point polygon[NMax+1];
```

(We could declare classes and methods, but may not be needed.)

It may be convenient to make the final point in a polygon the first point (so we know when polygon finished).

Using just above declarations we could create a triangle:

```
point a, b, c;
polygon tri;
a.x=1; a.y=2; b.x=1; ..
tri[0]=a; tri[1]=b; tri[2]=c; tri[0]=a;
```

4

Line Intersection

Given this representation, how do we tell if two lines intersect?

Method1: 'School Maths'

line1: $y = m_1 x + c_1$

line2: $y = m_2 x + c_2$

Solve for x, y to find point of intersection; check if point of intersection is within end points of both lines.

5

Opposite Sides

Once we have a ccw function, can determine if two points are opposite sides of a line:

p3 and p4 opposite sides of line p1 p2 if:

```
ccw(p1,p2,p3) != ccw(p1,p2,p4)
```

So we could have a function:

```
boolean opposite(line l, point p3, point p4)
{
    return boolean(ccw(l.p1, l.p2, p3) !=
                   ccw(l.p1, l.p2, p4));
}
```

7

Line Intersection: Another Method

This method is simpler to code.. but a bit non-intuitive.

Define a function ccw that, given 3 points, returns true if you have to turn *counter clockwise* when going from point 1 to 2 to 3.

6

Line Intersection

Two lines intersect if.. line1's points are on opposite sides of line2, and line2's points are on opposite sides of line1.

```
boolean intersect(line l1, line l2)
{
    return boolean(opposite(l1, l2.p1, l2.p2) &&
                   opposite(l2, l1.p1, l1.p2));
}
```

8

CCW function

How do we define the counter clockwise function needed for all this?

Just compare slopes. If we ignore equal/infinite slopes we have:

```
boolean ccw(point p1, point p2, point p3)
{
    float dx1, dx2, dy1, dy2;
    dx1=p2.x-p1.x; dy1=p2.y-p1.y;
    dx2=p3.x-p2.x; dy2=p3.y-p2.y;
    return boolean(dy1/dx1 < dy2/dx2);
}
```

(just needs checks for dx1 or dy=0)

9

Algorithm

- Create test line lt from given point out to infinity (or close!)
- For each point i in the polygon p:
 - Create a line segment lp from p[i] to p[i+1].
 - If lp intersects lt, increment counter.
- Return count mod 2

Test line lt can be created as:

```
lt.p1 = p;
lt.p2.y = p.y;
lt.p2.x = MAXINT;
```

11

Inclusion in a Polygon

How do we find out if a given point is inside a given polygon?

Simple, non-intuitive method exists.

Draw line out from point. Count how many times it intersects line in polygon. If odd, point is within polygon; if even point is outside:

10

Special Cases

What if test line intersects vertex? Do we count it once or twice?

Depends whether two points either side of vertex are on same side of the test line.

Same side: count twice; Opposite sides: count once.

12

Summary

- Geometric algorithms vital for many applications.
- Easy to represent the data; easy for humans to solve.
- Simple algorithms have been developed; but somewhat non-intuitive.