Computational Geometry
Chapter 33

Interesting area of algorithmics:

Applications in .. Graphics, VLSI, etc.

Cover a few basic problems (in 2 D)

Simple problem:

- In 1D, finding two closest point is trivial; in 2D?
- In 1 D, bounding the region of values (Max,Min)is easy; in 2D "convex hull"

Sharpen algorithm design tools and

Introduce a few basic techniques in Euclidean space

Major problem (hassle) getting arithmetic right … or close enough

Line Segments

Lots of definitions:

Convex combination of two points p_1,p_2: any point on the line segment, $\overline{p_1p_2}$, joining them.

$\overrightarrow{p_1p_2}$ a directed segment, a vector if p_1 origin

So, a few small problems on line segments:

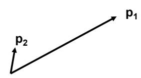
Use only +, - , *, no ÷ or trig functions; we want exact (correct) answer

Cross product: $p_1 \times p_2 = x_1y_2 - x_2y_1$ (= - $p_2 \times p_1$

Cross product = 0 ⇒ line segments $\overline{0\ p_1}$ and $\overline{0\ p_2}$ colinear.

Cross product > 0 ⇒ +ve angle (<π) from 0 p_1 to p_2 (counterclockwise)

Similarly Cross product < 0 ⇒ -ve angle



Extending

If the lines don't start at the origin ..

p_0p_2 relative to p_0p_1.

Compute $(p_1 - p_0)\times(p_2 - p_0)$

Consecutive segments: Similarly to check $p_0p_1p_2$ involves a left or right turn, compute

$(p_2 - p_0)\times(p_1 - p_0)$

Do two segments intersect: Check whether each "straddles" line of the other. (must check both ways, plus boundary condition)

Straddles: one end on one side, one on the other.

.. Several lines of code

(ok .. That was the "boring stuff")

The Sweep Line Approach
Do any segments intersect?

Given a set of n lines segments, do any of them intersect? If so find some or all.

There could be $\Theta(n^2)$ intersections, but we'll stick to "do any intersect"?

General approach:

Consider a vertical line, sweeping across the segments (left to right).

At each "event point" (end of a segment) update the data structures

and report interesting events (like a discovered intersection)

Assumptions

Assume for ease of presentation:

No vertical line segment. Trivial fix

No three line segments intersect at a single point. Method needs slight fix.

At position x of sweep line, segments s_1 and s_2 are *comparable* if both intersect the sweep line, and s_1 is *above* s_2 , $s_1 >_x s_2$.

As sweep line moves: update

- Sweep line status (relationship among objects, e.g. vertical order)

  and
- Event point schedule (when to do the next thing, could be much more complex than our simple queue)

Sweep Line Status

Retain total order, T, the vertical order of the segments "in play" with operations:

Insert(T,s): insert segment s into T

Delete(T,s): delete segment s from T

Above(T,s): return segment immediately above s, if any

Below(T,s): return segment immediately below s, if any

All can be done in O(lg n) time with a balanced search tree (e.g. AVL tree)

Segment intersection

```
T ← Ø
Sort endpoints of S by x-coord (tie protocol:
  inserts first, lower points first)
for each endpoint p
  do
  {  if p left endpoint of s
       then { Insert(T,s)
          if (Above(T,s) & intersects s)
            or (Below(T,s) & intersects s)
          then return true}
     if p right endpoint of s
       then { if (Above(T,s) & Below(T,s))
          & (Above(T,s) intersects Below(T,s))
          then return true
          Delete(T,s)}
  }
return false
[Overloading above/below as Boolean/segment]
```


Runtime and Correctness

Runtime is immediate: O(n) basic operations each taking O(lg n) time

Correctness:

The algorithm exhibits a crossing pair if it reports there is one.

If there is one, consider leftmost:

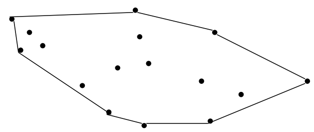
Vertical order of segments "in play", is correct up to first event after crossing.

Check for crossings at each event.

Care has to be taken to handle several crossings at same x coordinate.

Convex Hull

Smallest convex polygon containing the set of points



There are many O(n lg n) methods

We will look at one based on a rotational scan

Graham Scan Convex Hull

S is an initially empty stack

$p_0 \leftarrow$ lowest point (if tie leftmost of these)

$\langle p_1, p_2, \ldots p_m \rangle$ are remaining points sorted in counterclockwise direction by polar angle around p_0 (if tie keep only farthest)

{this can be done in O(n lg n) time}

Push(p_0,S)

Push(p_1,S)

Push(p_2,S)

for i← 3 to m do

{ while angle Next-to-top(s), Top(S), p_i makes nonleft turn do Pop(S)

Push(p_i,S)

}

return S

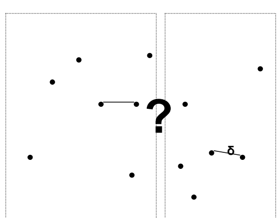
Closest Pair of Points

Finding the closest pair of points ..

In 1 dimensions, trivial: sort and compare difference between consecutive values

In two dimensions, much more interesting

Our solution .. A classic divide and conquer

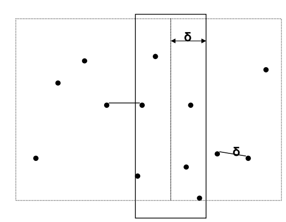


The Algorithm

Preamble: Sort points by x coordinate, and also by y coordinate (point in each list retains reference to position in the other)

Divide: Find closest pair in first half by x coordinate, and in second half. Let δ be the smaller distance (retain these end points).

Conquer: The closest pair is either that pair or a pair with one value in the left half and one in the right, each within δ of the splitting line



Continuing to Conquer

2 Issues:

- Get the points in the strip, in sorted order by y coordinate. Scan both x sorted and y sorted lists to do this. (this takes O(n) and avoids a sort)
- Consider point p in the strip. How many can be at its height or lower and vertical distance at most δ? Answer: at most 4 on the other side, but it is easier if we say 7 (4 on other side, 3 on its)
- Scan elements in the strip by y coordinate. Find distance between each and the following 7. Take closest pair if they have distance less than δ.