CS 5114: Theory of Algorithms

Clifford A. Shaffer

Department of Computer Science Virginia Tech Blacksburg, Virginia

Spring 2010

Copyright © 2010 by Clifford A. Shaffer

Geometric Algorithms

Potentially large set of objects to manipulate.

- Possibly millions of points, lines, squares, circles.
- Efficiency is crucial.

CS 5114: Theory of Algorithms

Computational Geometry

Will concentrate on discrete algorithms – 2D

Practical considerations

Special cases

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

CS 5114: Theory of Ald

Numeric stability

Definitions

- A **point** is represented by a pair of coordinates (x, y).
- A line is represented by distinct points p and q.
 ▶ Manber's notation: -p q -.
- A **line segment** is also represented by a pair of distinct points: the endpoints.
 - Notation: p q.
- A path *P* is a sequence of points *p*₁, *p*₂, ..., *p_n* and the line segments *p*₁ − *p*₂, *p*₂ − *p*₃, ..., *p_{n-1}* − *p_n* connecting them.
- A closed path has $p_1 = p_n$. This is also called a polygon.
 - Points \equiv vertices.
 - A polygon is a sequence of points, not a set.

Spring 2010 3 / 39

Spring 2010 1 / 39

Spring 2010 2 / 39

Definitions (cont)

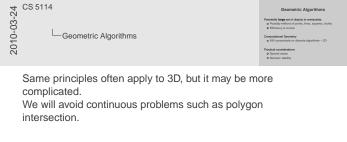
- Simple Polygon: The corresponding path does not intersect itself.
 - A simple polygon encloses a region of the plane INSIDE the polygon.
- Basic operations, assumed to be computed in constant time:
 - Determine intersection point of two line segments.
 - Determine which side of a line that a point lies on.
 - Determine the distance between two points.

CS 5114			



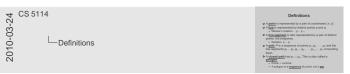
Title page

2010-03-24



Special cases: Geometric programming is much like other programming in this sense. But there are a LOT of special cases! Co-point, co-linear, co-planar, horizontal, vertical, etc.

Numeric stability: Each intersection point in a cascade of intersections might require increasing precision to represent the computed intersection, even when the point coordinates start as integers. Floating point causes problems!



Line alternate representation: slope and intercept. For polygons, order matters. A left-handed and right-handed triangle are not the same even if they occupy the same space.



Point in Polygon

Problem: Given a simple polygon P and a point q, determine whether q is inside or outside P.

Basic approach:

- Cast a ray from q to outside P. Call this L.
- Count the number of intersections between *L* and the edges of *P*.
- If count is even, then q is outside. Else, q is inside.

Problems:

CS 5114: Theory of Algorithms

- How to find intersections?
- Accuracy of calculations.
- Special cases.

Point in Polygon Analysis (1)

Time complexity:

- Compare the ray to each edge.
- Each intersection takes constant time.
- Running time is O(n).

Improving efficiency:

CS 5114: Theory of Algorithms

- O(n) is best possible for problem as stated.
- Many lines are "obviously" not intersected.

Point in Polygon Analysis (2)

Two general principles for geometrical and graphical algorithms:

- Operational (constant time) improvements:
 - Only do full calc. for 'good' candidates
 - Perform 'fast checks' to eliminate edges.
 - ► Ex: If *p*₁.*y* > *q*.*y* and *p*₂.*y* > *q*.*y* then don't bother to do full intersection calculation.
- For many point-in-polygon operations, preprocessing may be worthwhile.
 - Ex: Sort edges by min and max y values.
 Only check for edges covering y value of point q.

Spring 2010 7 / 39

Spring 2010 5 / 39

Spring 2010 6 / 39

Constructing Simple Polygons

Problem: Given a set of points, connect them with a simple closed path.

Approaches:

CS 5114: Theory of Ald

CS 5114: Theory of Algorithms

- Randomly select points.
- Use a scan line:
 - Sort points by y value.
 - Connect in sorted order.
- Sort points, but instead of by y value, sort by angle with respect to the vertical line passing through some point.
 - Simplifying assumption: The scan line hits one point at a time.
 - Do a rotating scan through points, connecting as you go.





Special cases:

- Line intersects polygon at a vertex, goes in to out.
- Line intersects poly. at inflection point (stays in or stays out).
- Line intersects polygon through a line.

Simplify calculations by making line horizontal.

Accuracy of calculations is not a problem with integer coordinates for points and a horizontal line. But think about representing the intersection point for two arbitrary line segements (from a polygon intersection operation). Cascading intersections can lead to ever-increasing demand for precision in coordinate representation.

42-60-0102



no notes



Spatial data structures can help.

"Fast checks" take time. When they "win" (they rule something out), they save time. When they "lose" (they fail to rule something out) they add extra time. Sot hey have to "win" often enough so that the time savings outweighs the cost of the check.

4	CS 5114	Constructing Simple Polygons
2		Problem: Given a set of points, connect them with a simp closed path.
N-0107	Constructing Simple Polygons	Approximate Approximate a scan line: Line on provide the scan line: Consect in some dynamics Storp topics, but instand of by y value, sort by angle is respect to the vertical line passing through some por scan draw scan draws to be also approximate the consection of the scan draws of the scan because the Do a stating scan through points, consecting as you

(1) Could easily yield an intersection.

(2) The problem is connecting point p_n back to p_1 . This could yield an intersection.

Simplifying assumption is that the points are not colinear w.r.t. the scan line.

Validation

Theorem: Connecting points in the order in which they are encountered by the rotating scan line creates a simple polygon.

Proof:

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

CS 5114: Theory of Ala

- Denote the points p_1, \dots, p_n by the order in which they are encountered by the scan line.
- For all *i*, 1 ≤ *i* < *n*, edge *p_i* − *p_{i+1}* is in a distinct slice of the circle formed by a rotation of the scan line.
- Thus, edge $p_i p_{i+1}$ does not intersect any other edge.
- Exception: If the angle between points p_i and p_{i+1} is greater than 180 \circ .

Spring 2010 9 / 39

oring 2010 10 / 39

Spring 2010

12/39

Implementation

How do we find the point for the scanline center?

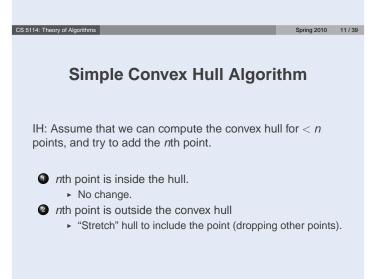
Actually, we don't care about angle - slope will do.

Select *z*; for (*i* = 2 to *n*) compute the slope of line $z - p_i$. Sort points p_i by slope; label points in sorted order;

Time complexity: Dominated by sort.

Convex Hull

- A <u>convex hull</u> is a polygon such that any line segment connecting two points inside the polygon is itself entirely inside the polygon.
- A **convex path** is a path of points p_1, p_2, \dots, p_n such that connecting p_1 and p_n results in a convex polygon.
- The convex hull for a set of points is the smallest convex polygon enclosing all the points.
 - imagine placing a tight rubberband around the points.
- The point **belongs** to the hull if it is a vertex of the hull.
- Problem: Compute the convex hull of n points.



|--|

So, the key is to pick a point for the center of the rotating scan that guarentees that the angle never reachese $180\circ$.



Pick as z the point with greatest x value (and least y value if there is a tie).

The next point is the next largest angle between $z - p_i$ and the vertical line through *z*. It is important to use the slope, because then our computation is a constant-time operation with no transendental functions.

z is the point with greatest x value (minimum y in case of tie)So, time is $\Theta(n \log n)$

2010-03-24 CS 21	LT14	Convex Hull

no notes



Subproblems (1)

Potential problems as we process points:

- Determine if point is inside convex hull.
- Stretch a hull.

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

The straightforward induction approach is inefficient. (Why?)

Our standard induction alternative: Select a special point for the *n*th point – some sort of min or max point.

If we always pick the point with max *x*, what problem is eliminated? Stretch:

- Find vertices to eliminate
- Add new vertex between existing vertices.

Spring 2010 13 / 39

Spring 2010 14 / 39

Spring 2010 15 / 39

Spring 2010

16/39

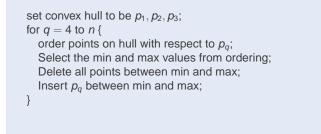
Subproblems (2)

Supporting line of a convex polygon is a line intersecting the polygon at exactly one vertex.

Only two supporting lines between convex hull and max point q.

These supporting lines intersect at "min" and "max" points on the (current) convex hull.

Sorted-Order Algorithm



Time complexity

Sort by x value: $O(n \log n)$.

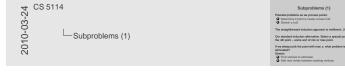
For *q*th point:

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

- Compute angles: O(q)
- Find max and min: O(q)
- Delete and insert points: O(q).

 $T(n) = T(n-1) + O(n) = O(n^2)$



Why? Lots of points don't affect the hull, and stretching is expensive.

Subproblem 1 can be eliminated: the max is always outside the polygon.

+ CS 5114	Subproblems (2)
°Ó -Ó- └─Subproblems (2) N	Begingstring they did stransport of the stransport of the stransport of the polygin of a society one way of the stransport of the stransport of the strangestring the stransport of the stransport of the strangestring the stransport of the stransport of the stransport of the strangestring the stransport of the stranspo

"Min" and "max" with respect to the angle formed by the supporting lines.

7 CS 5114		Sorted-Order Alg
2010-00	Sorted-Order Algorithm	set convex hull to be $p_1,p_2,p_3;$ for $q=4$ to n (order point on hull with respect to p_1 select the min and max values from Delete all points between min and max;)

Sort by x value.



Gift Wrapping Concept

- Straightforward algorithm has inefficiencies.
- Alternative: Consider the whole set and build hull directly.
- Approach:
 - Find an extreme point as start point.
 - ► Find a supporting line.
 - Use the vertex on the supporting line as the next start point and continue around the polygon.
- Corresponding Induction Hypothesis:
 - Given a set of n points, we can find a convex path of length k < n that is part of the convex hull.
- The induction step extends the PATH, not the hull.



Gift Wrapping Analysis

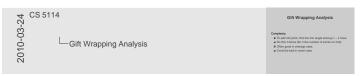
Complexity:

- To add *k*th point, find the min angle among n k lines.
- Do this *h* times (for *h* the number of points on hull).
- Often good in average case.
- Could be bad in worst case.



Straightforward algorithm spends time to build convex hull with points interior to final convex hull.





 $O(n^2)$. Actually, O(hn) where h is the number of edges to hull.



Graham's Scan

• Approach:

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

- Start with the points ordered with respect to some maximal point.
- Process these points in order, adding them to the set of processed points and its convex hull.
- ► Like straightforward algorithm, but pick better order.
- Use the Simple Polygon algorithm to order the points by angle with respect to the point with max *x* value.
- Process points in this order, maintaining the convex hull of points seen so far.

See Manber Figure 8.11.

Spring 2010 19 / 39

Spring 2010

20/39

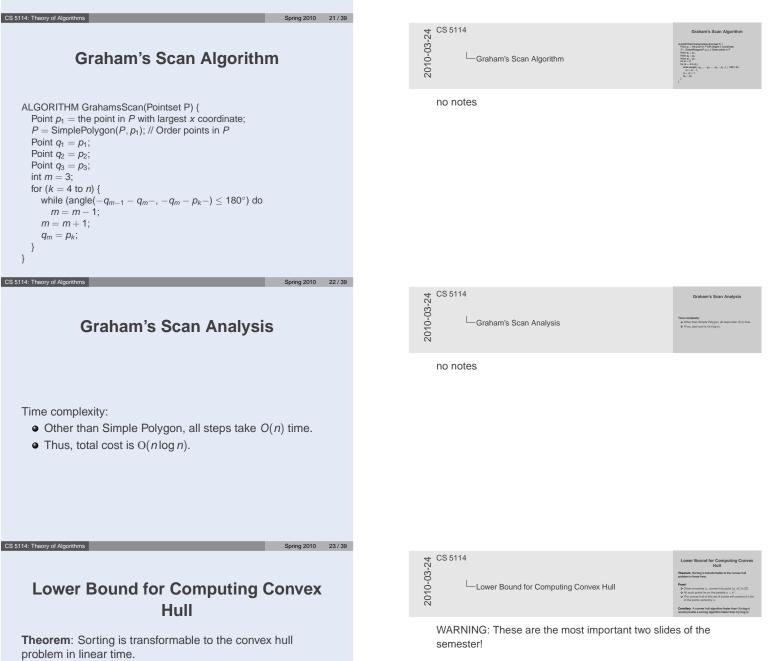
Graham's Scan (cont)

Induction Hypothesis:

• Given a set of *n* points ordered according to algorithm Simple Polygon, we can find a convex path among the first n - 1 points corresponding to the convex hull of the n - 1 points.

Induction Step:

- Add the *k*th point to the set.
- Check the angle formed by p_k, p_{k-1}, p_{k-2} .
- If angle < 180∘ with respect to inside of the polygon, then delete p_{k-1} and repeat.



CS 5114

no notes

Graham's Scan (cont)

2010-03-24

Proof:

CS 5114: Theory of Algorithms

- Given a number x_i , convert it to point (x_i, x_i^2) in 2D.
- All such points lie on the parabla $y = x^2$.
- The convex hull of this set of points will consist of a list of the points sorted by *x*.

Corollary: A convex hull algorithm faster than $O(n \log n)$ would provide a sorting algorithm faster than $O(n \log n)$.

"Black Box" Model

A Sorting Algorithm:

keys \rightarrow points: $O(n)$
Convex Hull
CH Polygon \rightarrow Sorted Keys: $O(n)$

Closest Pair

- **Problem**: Given a set of *n* points, find the pair whose separation is the least.
- Example of a proximity problem
 - Make sure no two components in a computer chip are too close.
- Related problem:
 - ► Find the nearest neighbor (or *k* nearest neighbors) for every point.
- Straightforward solution: Check distances for all pairs.
- Induction Hypothesis: Can solve for n 1 points.
- Adding the *n*th point still requires comparing to all other points, requiring $O(n^2)$ time.

Divide and Co	nquer Alg	orithm
---------------	-----------	--------

- Approach: Split into two equal size sets, solve for each, and rejoin.
- How to split?

CS 5114: Theory of Algorithms

- Want as much valid information as possible to result.
- Try splitting into two disjoint parts separated by a dividing plane.
- Then, need only worry about points close to the dividing plane when rejoining.
- To divide: Sort by x value and split in the middle.

Spring 2010 27 / 39

Spring 2010

28/39

Spring 2010 25 / 39

Spring 2010 26 / 39

Closest Pair Algorithm

Induction Hypothesis:

CS 5114: Theory of Algorithms

• We can solve closest pair for two sets of size *n*/2 named *P*₁ and *P*₂.

Let minimal distance in P_1 be d_1 , and for P_2 be d_2 .

• Assume $d_1 \leq d_2$.

CS 5114: Theory of Algorithms

Only points in the strip of width d_1 to either side of the dividing line need to be considered.

Worst case: All points are in the strip.

CS 5114	
"Black Box" Model	

2010-03-24



This is the fundamental concept of a reduction. We will use this constantly for the rest of the semester.



Next try: Ordering the points by x value still doesn't help.



Assume $n = 2^k$ points.

Note: We will actually compute smallest distance, not pair of points with smallest distance.



See Manber Figure 8.13

Closest Pair Algorithm (cont)

Observation:

CS 5114: Theory of Algorithms

 A single point can be close to only a limited number of points from the other set.

Reason: Points in the other set are at least d_1 distance apart.

Sorting by y value limits the search required.

Closest Pair Algorithm Cost

 $O(n \log n)$ to sort by x coordinates.

Eliminate points outside strip: O(n).

Sort according to *y* coordinate: $O(n \log n)$.

Scan points in strip, comparing against the other strip: O(n).

 $T(n) = 2T(n/2) + O(n \log n).$

 $T(n) = O(n \log^2 n).$

CS 5114: Theory of Algorithms

A Faster Algorithm

The bottleneck was sorting by y coordinate.

If solving the subproblem gave us a sorted set, this would be avoided.

Strengthen the induction hypothesis:

• Given a set of < *n* points, we know how to find the closest distance and how to output the set ordered by the points' *y* coordinates.

All we need do is merge the two sorted sets – an O(n) step.

T(n) = 2T(n/2) + O(n). $T(n) = O(n \log n).$ CS 5114: Theory of Algorithms

Spring 2010 31 / 39

Spring 2010 29 / 39

Spring 2010 30 / 39

Horizontal and Vertical Segments

- Intersection Problems:
 - Detect if any intersections ...
 - Report any intersections ...
 - ... of a set of <line segments>.
- We can simplify the problem by restricting to vertical and horizontal line segments.
- Example applications:

CS 5114: Theory of Ald

- Determine if wires or components of a VLSI design cross.
- Determine if they are too close.
 - * Solution: Expand by 1/2 the tolerance distance and check for intersection.
- Hidden line/hidden surface elimination for Computer Graphics.

Spring 2010 32 / 39

CS 5114	
Closest Pair Algorithm (co	nt)

Closest Pair Algorithm (cont)

See Manber Figure 8.14

2010-03-24



CS 5114 A Faster Algorithm
CA Faster Algorithm
A Faster Algorithm
Gamma and Gamma and

no notes

2010-03-24



Sweep Line Algorithms (1)

Problem: Given a set of *n* horizontal and *m* vertical line segments, find all intersections between them.

• Assume no intersections between 2 vertical or 2 horizontal lines.

Straightforward algorithm: Make all $n \times m$ comparisons.

If there are $n \times m$ intersections, this cannot be avoided.

However, we would like to do better when there are fewer intersections.

Solution: Special order of induction will be imposed by a **sweep line**.

Spring 2010 33 / 39

Spring 2010 34 / 39

Sweep Line Algorithms (2)

Plane sweep or **sweep line** algorithms pass an imaginary line through the set of objects.

As objects are encountered, they are stored in a data structure.

When the sweep passes, they are removed.

Preprocessing Step:

CS 5114: Theory of Algorithms

CS 5114: Theory of Algorithms

• Sort all line segments by x coordinate.

Sweep Line Algorithms (3)

Inductive approach:

- We have already processed the first *k* 1 end points when we encounter endpoint *k*.
- Furthermore, we store necessary information about the previous line segments to efficiently calculate intersections with the line for point *k*.

Possible approaches:

CS 5114: Theory of Algorith

Store vertical lines, calculate intersection for horizontal lines.

Store horizontal lines, calculate intersection for vertical lines.

Spring 2010 35 / 39

Spring 2010

36/39

Organizing Sweep Info

What do we need when encountering line L?

- NOT horizontal lines whose right endpoint is to the left of L.
- Maintain <u>active</u> line segments.

What do we check for intersection?

Induction Hypothesis:

CS 5114: Theory of Algorithms

• Given a list of *k* sorted coordinates, we know how to report all intersections among the corresponding lines that occur to the left of *k*.*x*, and to eliminate horizontal lines to the left of *k*.



This is a "classic" computational geometry problem/algorithm



The induction here is to add a special *n*th element.



Since we processed by *x* coordinate (i.e., sweeping horizontally) do (2). When we process a vertical line, it is clear which horizontal lines would be relevent (the ones that cross that include the *x* coordinate of the vertical line), and so could hope to find them in a data structure. If we stored vertical lines, when we process the next horizontal line, it is not so obvious how to find all vertical lines in the horizontal range.



See Figure 8.17 in Manber.

y coordinates of the active horizontal lines.

Sweep Line Tasks

Things to do:

- (k + 1)th endpoint is right endpoint of horizontal line. Delete horizontal line.
- 2 (k+1)th endpoint is left endpoint of horizontal line. Insert horizontal line.
- (k + 1)th endpoint is vertical line.
 - Find intersections with stored horizontal lines.

Data Structure Requirements (1)

To have an efficient algorithm, we need efficient

- Intersection
- Deletion

CS 5114: Theory of Algorithms

• 1 dimensional range query

Example solution: Balanced search tree

- Insert, delete, locate in log n time.
- Each additional intersection calculation is of constant cost beyond first (traversal of tree).

Data Structure Requirements (2)

Time complexity:

CS 5114: Theory of Algorithms

- Sort by *x*: $O((m+n)\log(m+n))$.
- Each insert/delete: O(log n).
- Total cost is $O(n \log n)$ for horizontal lines.

Processing vertical lines includes one-dimensional range query:

• $O(\log n + r)$ where *r* is the number of intersections for this line.

Thus, total time is $O((m+n)\log(m+n)+R)$, where R is the total number of intersections. CS 5114: Theory of Algorithms

Spring 2010 39 / 39

Spring 2010 37 / 39

Spring 2010 38 / 39

24	CS 5114
0-03-	Sweep Line
201	



Deleting horizontal line is $O(\log n)$.

Tasks

Inserting horizontal line is $O(\log n)$.

Finding intersections is $O(\log n + r)$ for *r* intersections.

no notes

