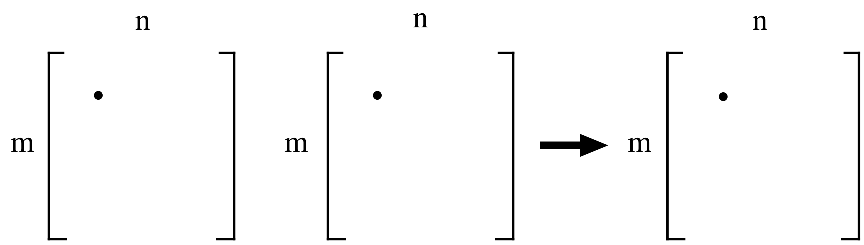
CSE 5311 Notes 16: Matrices

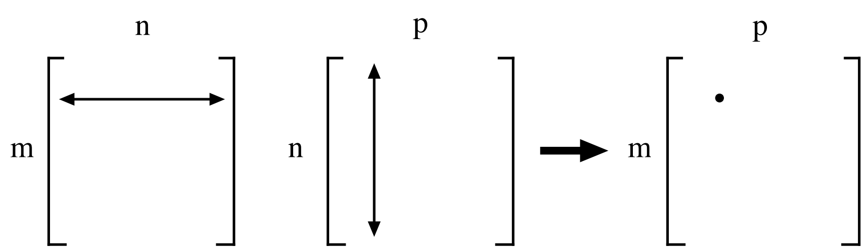
STRASSEN'S MATRIX MULTIPLICATION

Matrix addition:



takes mn scalar additions.

Everyday matrix multiply:



takes mnp scalar multiplies and m(n-1)p scalar additions.

Let m = n = p.

Best lower bound is $\Omega(n^2)$.

For n = 2:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

is done by everyday method using:

8 scalar multiplies

4 scalar additions

but Strassen's method (CLRS, p.735) uses:

7 scalar multiplies

18 scalar additions

When $n = 2^k$: A_{ij} and B_{ij} are $2^{k-1} \times 2^{k-1}$ submatrices that are multiplied *recursively* using Strassen's method.

Suppose n = 4.

Everyday method will take $4^3 = 64$ scalar multiplies and $16 \cdot 3 = 48$ scalar additions.

Strassen's method will use:

7 recursive 2 x 2 matrix multiplies, each using 7 scalar multiplies and 18 scalar additions.

18 2 x 2 matrix additions, each using 4 scalar additions.

This gives 49 scalar multiplies and 198 scalar additions.

Let *M*(k) = number of scalar multiplies for $2^k \times 2^k = n \times n$:

$$M(0) = 1$$
$$M(1) = 7$$
$$M(k) = 7M(k-1) = 7^k = 7^{\lg n} = n^{\lg 7} \approx n^{2.81}$$ Note that the constant is 1.

Let *P*(k) be the number of additions (including subtractions) done for $2^k \times 2^k = n \times n$:

$$P(0) = 0$$
$$P(1) = 18$$
$$P(k) = 18\left(2^{k-1}\right)^2 + 7P(k-1) = 18\left(\frac{2^k}{2}\right)^2 + 7P(k-1)$$
$$P(k) = P'\left(2^k\right)$$
$$P'(n) = 18\left(\frac{n}{2}\right)^2 + 7P'\left(\frac{n}{2}\right) = \frac{9}{2}n^2 + 7P'\left(\frac{n}{2}\right)$$

Master Method:

$$a = 7 \quad b = 2 \quad \log_b a = \lg 7 \approx 2.81 \quad f(n) = \frac{9}{2}n^2$$
$$\frac{9}{2}n^2 = O\left(n^{2.81-\varepsilon}\right)$$
Case 1: $P'(n) = \Theta\left(n^{2.81}\right)$

CSE 5311 Notes 17: Computational Geometry

FUNDAMENTAL PREDICATES

Twice the (signed) area of a triangle $A(T)$ is given by:

$$2A(T) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} = \begin{vmatrix} x_b - x_a & y_b - y_a \\ x_c - x_a & y_c - y_a \end{vmatrix} = (x_b - x_a)(y_c - y_a) - (x_c - x_a)(y_b - y_a)$$

If positive, then points a, b, and c make a left turn (counter-clockwise).

If negative, then points a, b, and c make a right turn (clockwise).

If zero, then points a, b, and c are collinear.

Relationship of a point a to counter-clockwise circle of points b, c, and d? (Based on tetrahedral volume.)

$$\begin{vmatrix} x_a & y_a & x_a^2 + y_a^2 & 1 \\ x_b & y_b & x_b^2 + y_b^2 & 1 \\ x_c & y_c & x_c^2 + y_c^2 & 1 \\ x_d & y_d & x_d^2 + y_d^2 & 1 \end{vmatrix} \quad \begin{matrix} \text{Zero: on circle} \\ \text{Positive: outside} \\ \text{Negative: inside} \end{matrix}$$

PROXIMITY

Closest points in 1-d space (`1dclosest.c`)

1. Find median of point set.
2. Recursively determine closest pair on left side and right side.
3. Check whether rightmost in left side and leftmost in right side are a closer pair than 2.

Worst-case: $\Theta(n \log n)$

Closest points in 2-d space (`2dclosest.c`)

Brute-force: $\Theta\left(n^2\right)$

Divide-and-conquer:

1. Draw vertical line to divide into equal-size subsets.

2. Recursively find closest pair for left and right sides. Let δ be the smaller of the two distances.

3. Find closest pair among points within δ of the dividing line.

Since the point set is not random, details must assure that $\Theta\left(n^2\right)$ behavior is avoided.

Base Case: If $n \le 3$ (or some other constant), use brute-force.

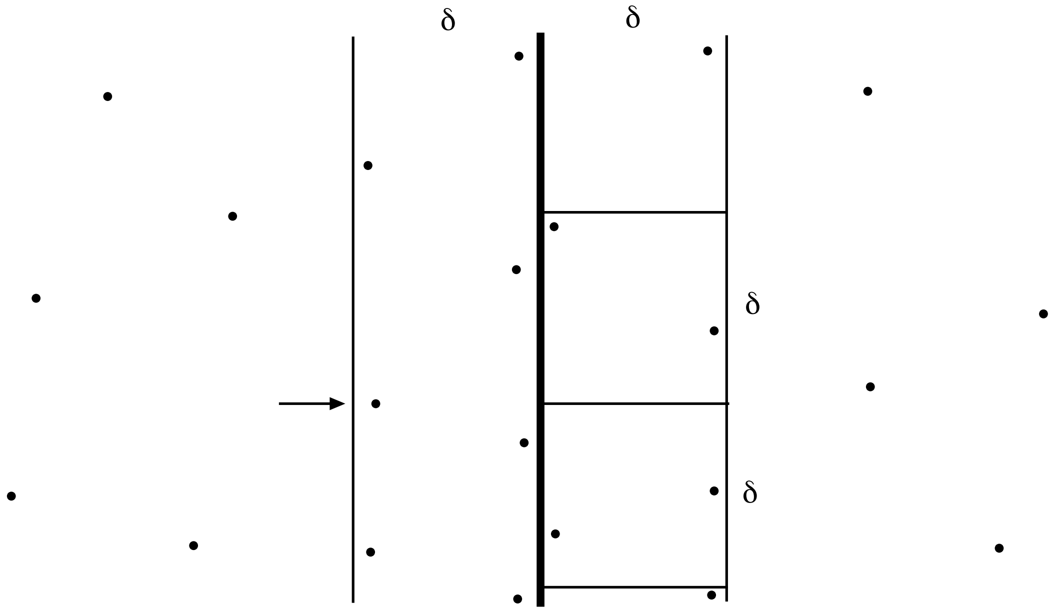
To support the "divides" and the seam processing, the set of points is preprocessed:

1. Create array with points sorted by x-coordinate.

2. Create second array with points sorted by y-coordinate. Also include cross-references to x-ordered array.

When a "divide" by a vertical line is needed, the first array is trivial to split and the second array is split by using the cross-references.

The y-ordered array facilitates finding the closest pair across the seam.



For a given left-side seam point, the distances to at most six right-side seam points are needed.

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n)$$

CONVEX HULLS

Determine smallest convex polygon that includes all points in a 2-d set.

Graham scan - Based on *angular sweep* w.r.t. the (leftmost) bottom point X and maintaining stack with convex hull.

1. Find X.

2. Sort by angle w.r.t. X. Comparisons by testing "turns" and breaking collinear cases by taking farthest point first.

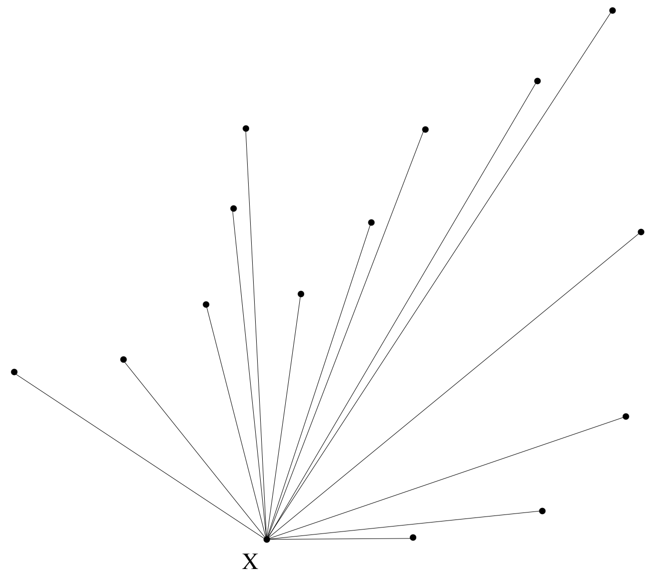
3. Push X and first two sweep points.

4. for each point P in sorted order

    while top-of-stack, next-to-top-of-stack, and P do not make a left turn

        Discard top-of-stack (it's not in convex hull)

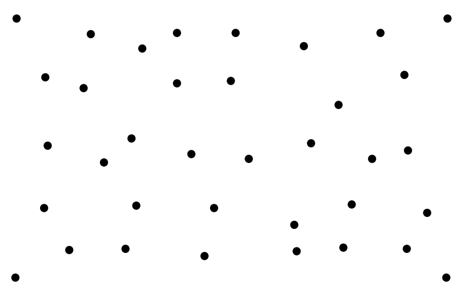
    Push P



Jarvis march (rubberbanding or gift-wrapping)

Runs in $\Theta(nh)$ where h is the number of hull points

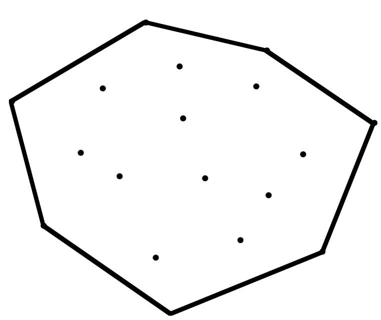
Good for cases like:



Same initial point X as Graham scan.

Also need (leftmost) top point Y.

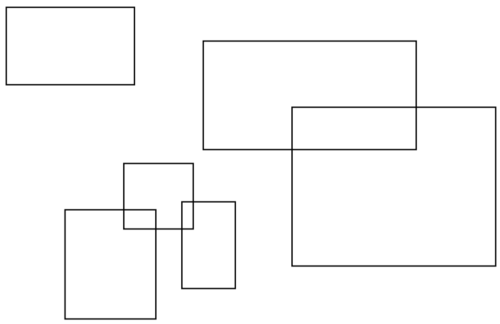
Each successive hull point is found by finding minimum angle WRT two previous points.



Convex hull may be used to find the diameter of a set using $\Theta(h)$ additional time.

SWEEP-LINE ALGORITHMS

Simple example: Intersection of rectilinear rectangles



Idea: Sweep a vertical line from left-to-right and store vertical cross-section (sweep-line status).

Preprocessing: Sort left and right edges by x-coordinate (event-point schedule).

Algorithm: Sweep across x dimension

Left edge: Check for intersection. Insert in interval tree (CLRS, 14.3)

Right edge: Delete from interval tree

Runs in $\Theta(n \log n + n \log m) = \Theta(n \log n)$ (m is max rects in tree)

Difficulty: What if two rectangles "touch"? Treat as intersecting or not by how ties are handled.

More significant example: 2-d closest pairs

Idea: Incrementally determine δ for the leftmost k points. Maintain y-ordered BST of points whose x-distance from point $k + 1$ is $< \delta$.

Preprocessing: Sort points by x-coordinate.

Processing point $k + 1$:

1. Delete BST points that are at least δ to the left.
2. Insert point $k + 1$ into BST.
3. Examine BST predecessors of point $k + 1$ until δ away in y. Check for improving δ.
4. Like 3., but with BST successors.

Time: $\Theta(n \log n)$

Reference: K. Hinrichs, J. Nievergelt, and P. Schorn, "Plane-Sweep Solves the Closest Pair Problem Elegantly", *Information Processing Letters* 26 (1988), 255-261.

EUCLIDEAN MINIMUM SPANNING TREES

Voronoi Diagram - post office problem. Divides plane into convex regions, each containing points closest to some given point (blue lines).

Fortune's Sweep-Line achieves $\Theta(n \log n)$ - applet

Delaunay Triangulation

Connects vertices for adjacent Voronoi regions (black lines between input points).

May transform an arbitrary triangulation to a DT in $\Theta\left(n^2\right)$ time using flips based on incircle test and the following property:

Three points are vertices of a Delaunay triangle iff the circle that passes through the three points is empty.