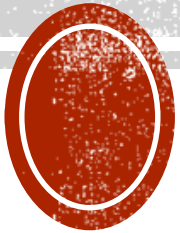
Lect09. Geometry Algorithm-1

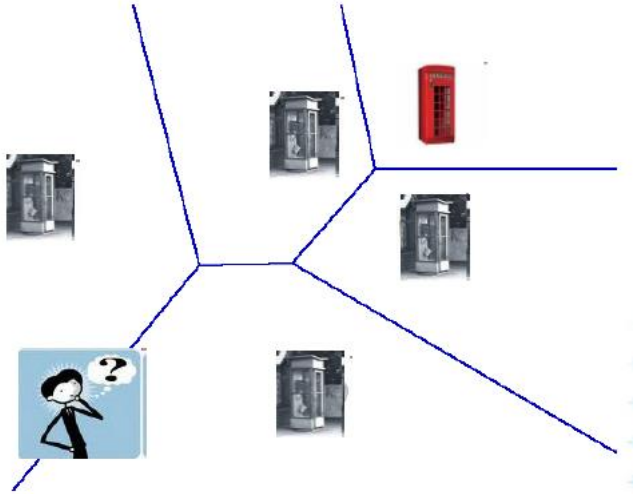


Computational Geometry

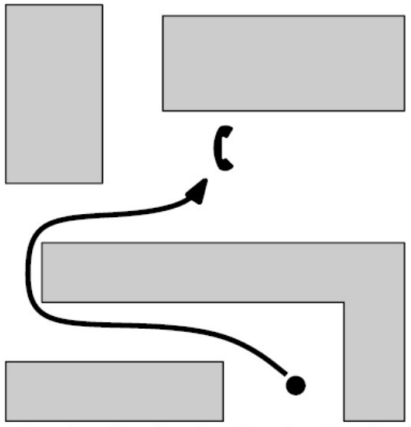
- is a subfield of the *Design and Analysis of Algorithms*
- deals with efficient data structures and algorithms for geometric problems
- Systematic objects (points, lines, line segments, n-gons, …) with focus on exact algorithms that are asymptotically fast
- started out by developing solid theoretical foundations, but became more and more applied over the last years
- "Born" in 1975(Shamos), boom of papers in 90s (first papers sooner:1850 Dirichlet, 1908 Voronoi, …)
- Many problems can be formulated geometrically (e.g., range queries in databases)

Computational Geometry-examples

- Where is the nearest phone, metro, pub … ?

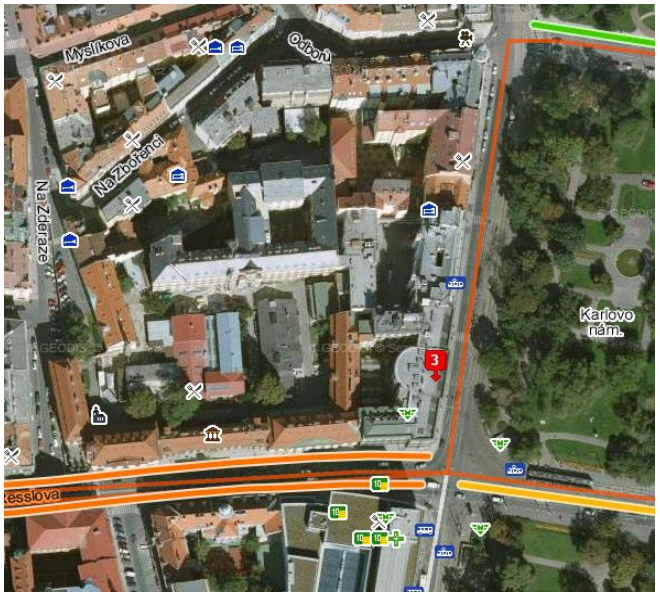


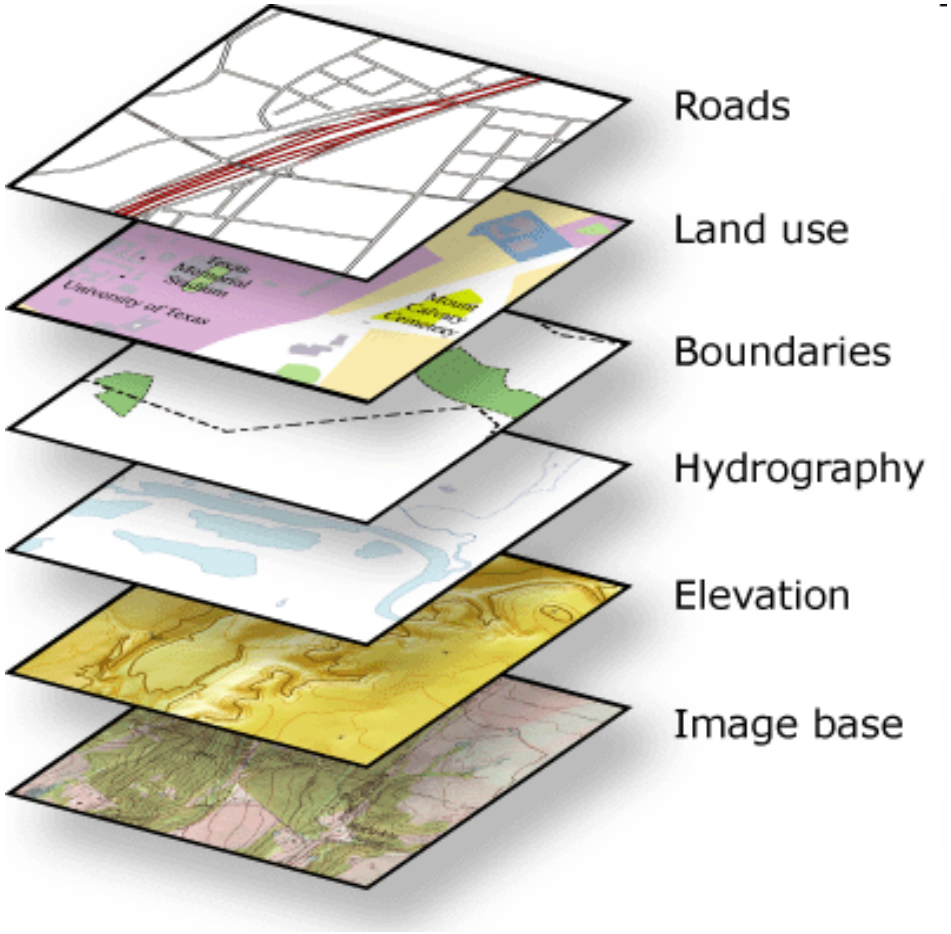
- How to get there ?



Computational Geometry-examples(cont.)

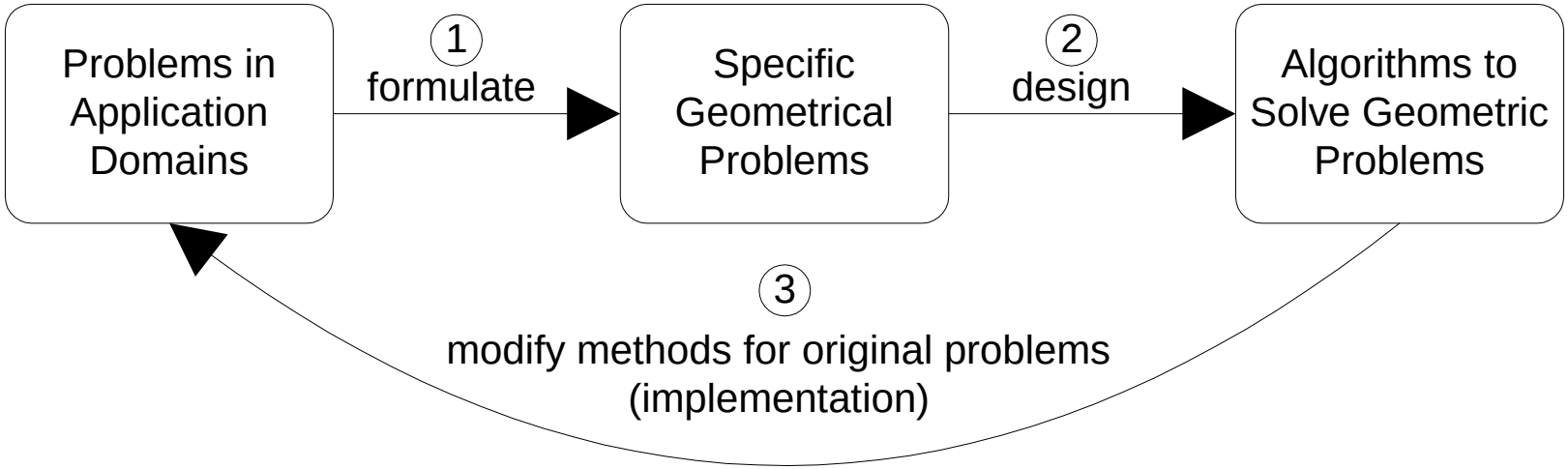
- Map overlay





What is Computational Geometry?

- Study of Data Structures and Algorithms for Geometric Problems



- 1970's: Need for Computational Geometry Recognized; Progress made on 1 and 2
- Today: "Mastered" 1 and 2, not so successful with 3

What is Computational Geometry?

- Good solutions need both:
  - Understanding of the geometric properties of the problem
  - Proper applications of algorithmic techniques (paradigms) and data structures

Computational Geometry

- Inclusion problems:
  - locating a point in a planar subdivision,
  - reporting which point among a given set are contained in a specified domain, etc.
- Intersection problems:
  - finding intersections of line segments, polygons, circles, rectangles, polyhedra, half spaces, etc.
- Proximity problems:
  - determining the closest pair among a set of given points,
  - computing the smallest distance from one set of points to another.
- Construction problems:
  - identify the convex hull of a polygon,
  - obtaining the smallest box that includes a set of points, etc.

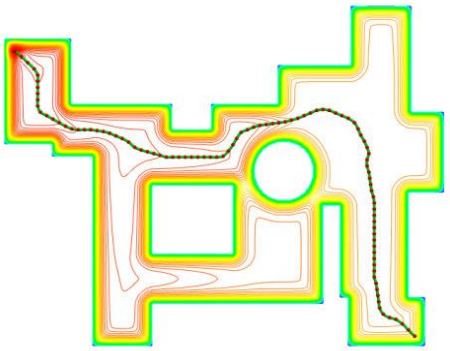
Elementary Geometric Methods

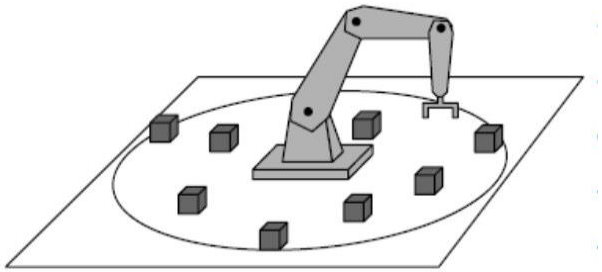
- Easy to visualize, hard to implement (but sometimes it will lead to much more efficient algorithms than other methods).
- A graph is planar if it can be drawn in the plane so that no two of its edges intersect.
- A planar subdivision is the drawing of a planar graph in straight-line segments.
- A polygon is simple if no two edges of it intersect.
- Two polygons Q and R are said to be intersect if an edge of Q cross an edge of R.
- Basic geometric data structures:

```
type point = record x, y: integer end;
     line = record p1, p2: point end;
var  polygon: array[0..Nmax] of point;
```

Application Domains

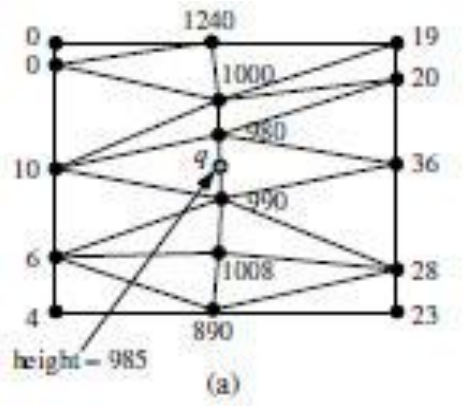
- Computer Graphics and Virtual Reality
  - 2-D & 3-D: intersections, hidden surface elimination, ray tracing
  - Virtual Reality: collision detection (intersection)
  - Mouse localization
  - Selection of objects in region
  - Visibility in 3D(hidden surface removal)
  - Computation of shadows
- Robotics
  - Motion planning(find path-environment with obstacles)
  - Task planning(motion+planning order of subtasks)
  - Design of robots and working cells

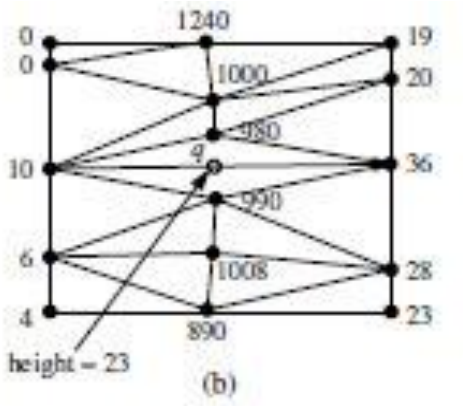


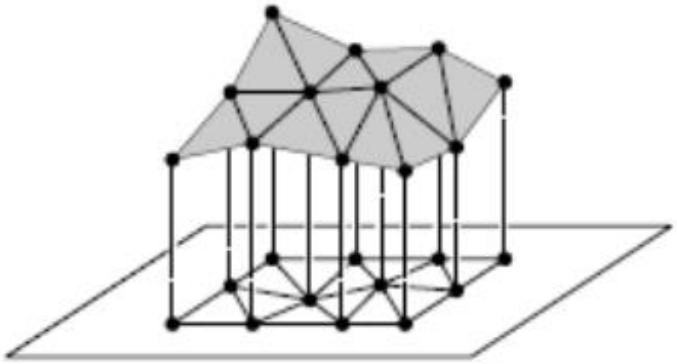


Application Domains

- Geographic Information Systems (GIS)
  - How to store huge data and search them quickly ➔ data structure design
  - Interpolation of heights : Find additional points based on values of known points
  - Overlays ➔ Find points in multiple layers
    - Extract information about regions or relations between data (pipes under the construction site, plants x average rainfall,…)
    - Detect bridges on crossings of roads and rivers…
  - Voronoi Diagrams of points

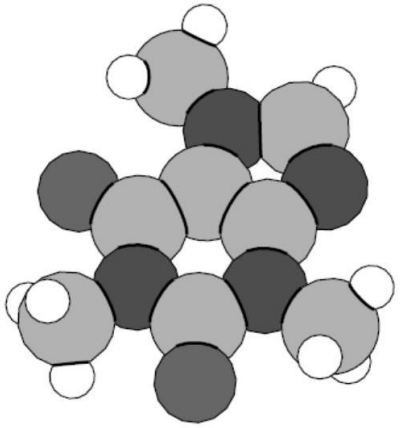






Application Domains(cont.)

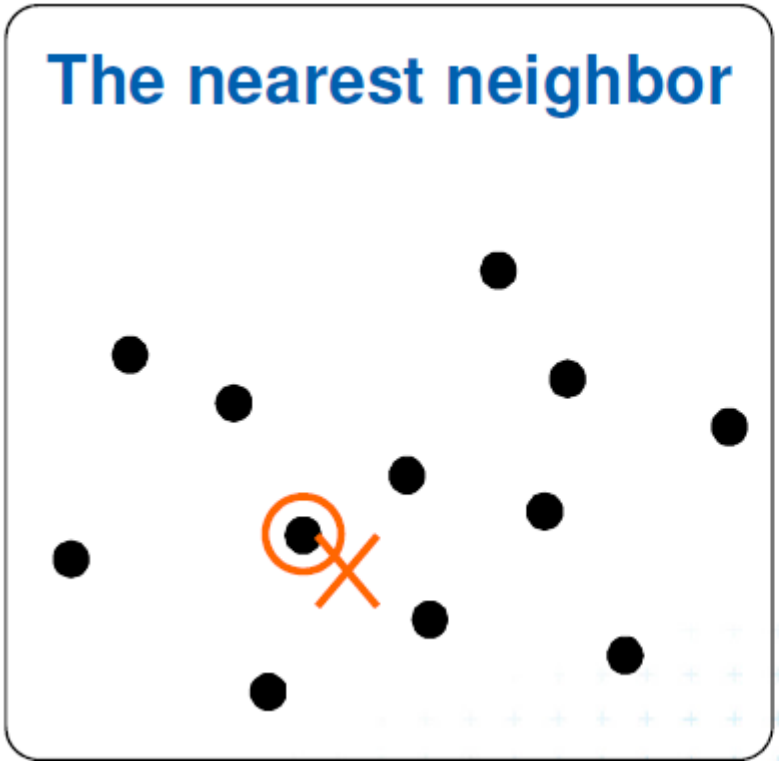
- Computer Aided Design and Manufacturing (CAD / CAM)
  - Design 3-D objects and manipulate them
    - Possible manipulations: merge (union), separate, move
  - "Design for Assembly"
    - CAD/CAM provides a test on objects for ease of assembly, maintenance, etc.
- Computational Biology
  - Determine how proteins combine together based on folds in structure
    - Surface modeling, path finding, intersection

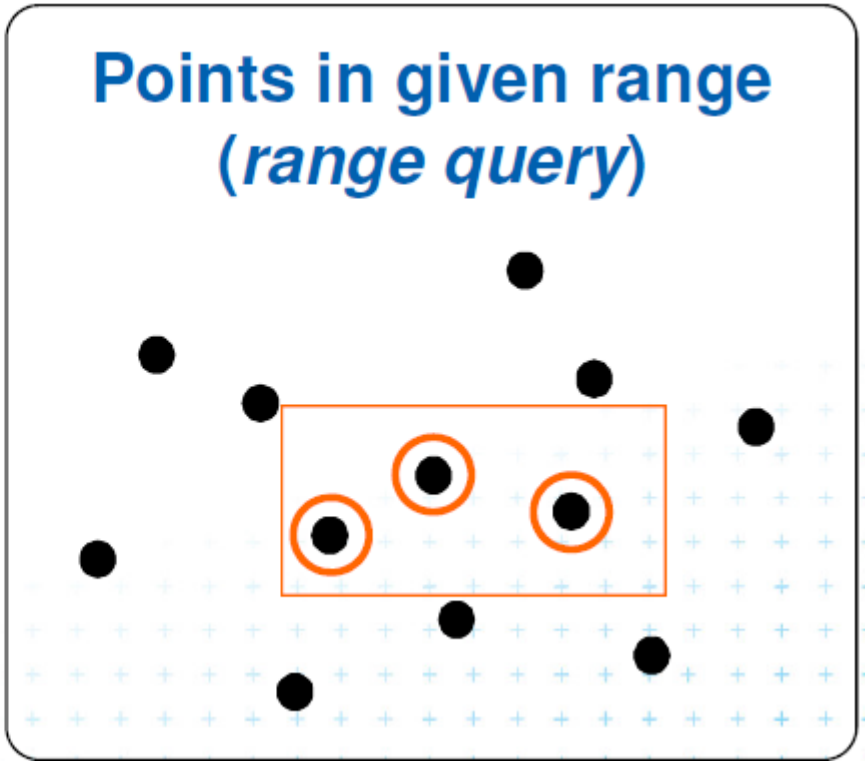


caffeine

Typical tasks in CG(computational geometry)

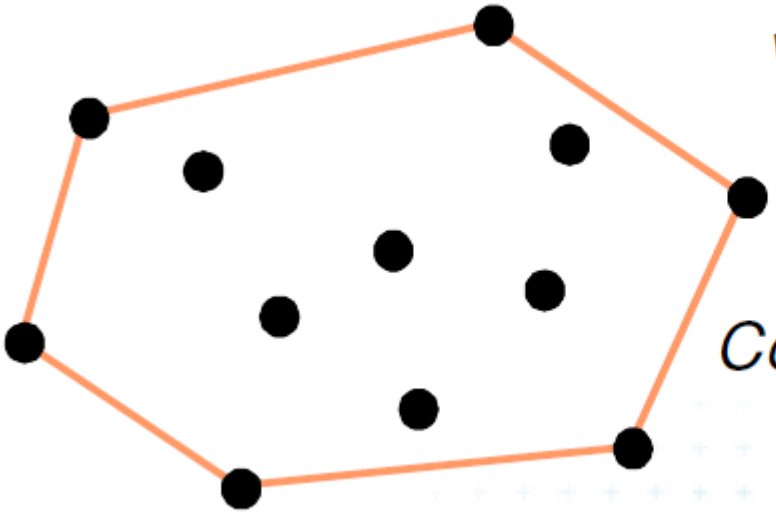
- Geometric searching – fast location of :





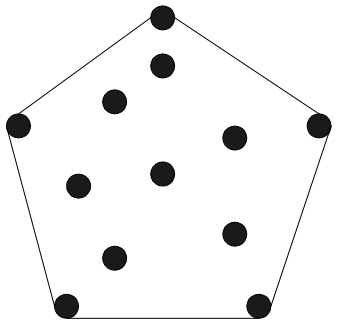
Typical tasks in CG(computational geometry)

- Convex hull : smallest enclosing convex polygon containing all the points



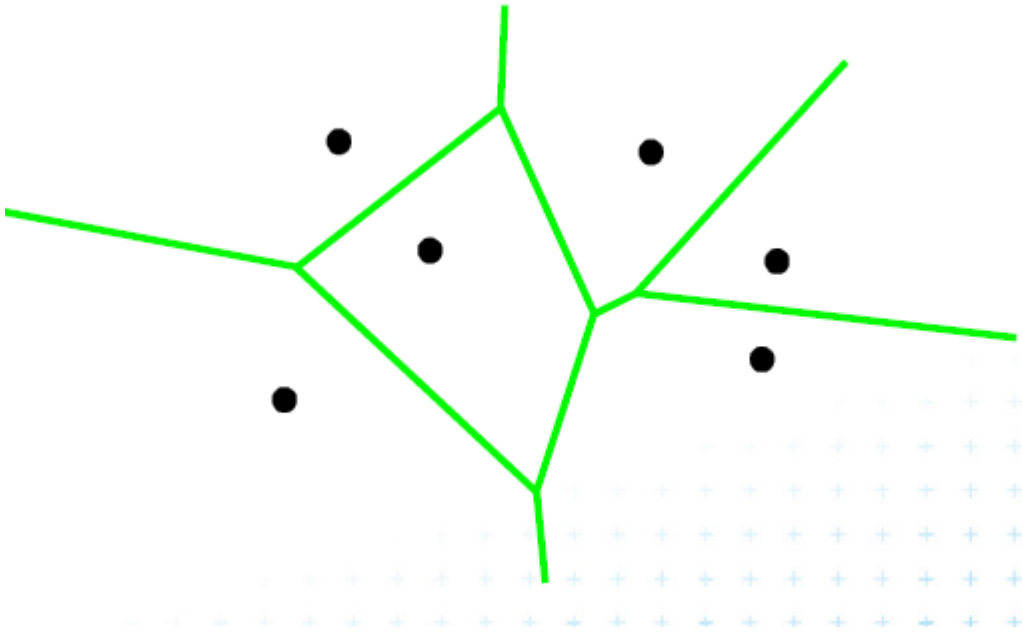
Example: 2-D Convex Hulls

- Definitions
  - A subset P of the plane is **convex** iff for every p,q $\in$ P line segment pq is completely contained in P.
  - The **convex hull** of a set P, denoted **CH(P)**, is the smallest convex set containing P.
    - The intersection of all convex sets containing P.
    - Analogy: If the points are nails in a board, a rubber band (convex hull) encloses all of the nails in the board.
    - Alternative Definition: CH(P) is the unique convex polygon whose vertices are points of P that contains all points in P



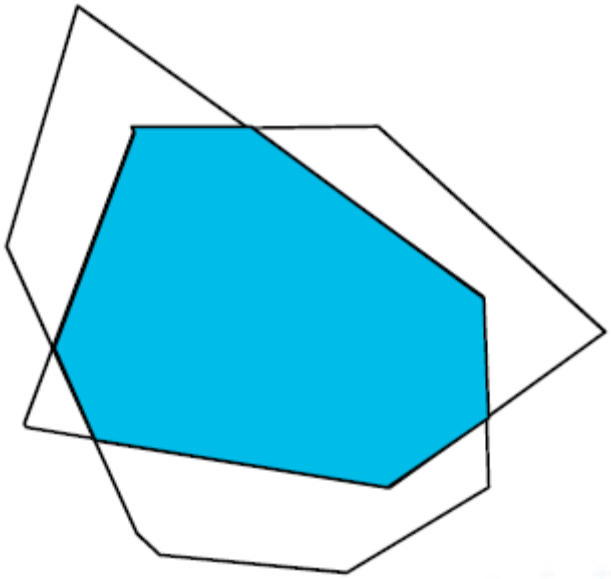
Typical tasks in CG(computational geometry)

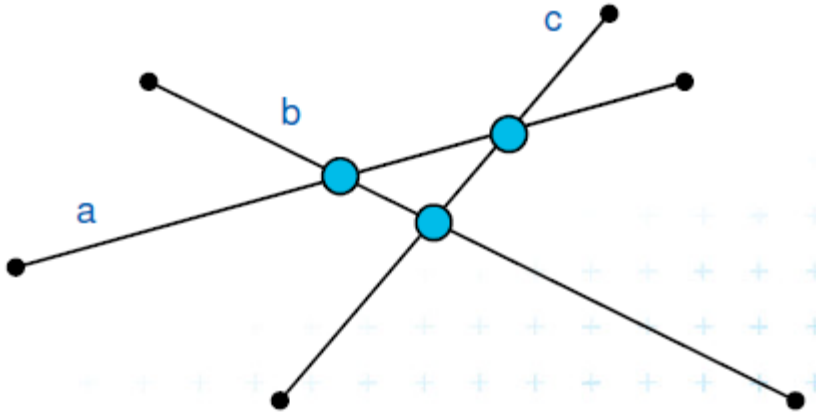
- Voronoi diagrams
  - Spaces(plane) partitioning into regions whose points are nearest to the given primitive(most usually a point)



Typical tasks in CG(computational geometry)

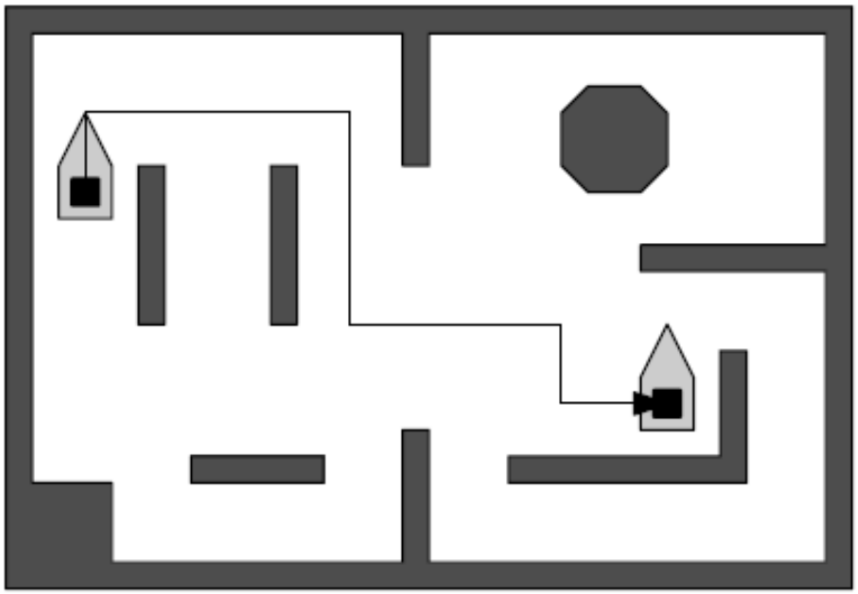
- Intersection of objects
  - Detection of common parts of objects
  - Usually linear (line segments, polugons, n-gons, …)



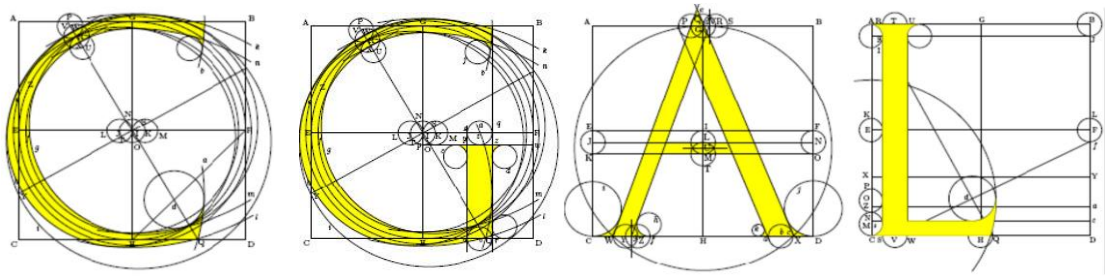


Typical tasks in CG(computational geometry)

- Motion planning
  - Search for the shortest path between two points in the environment with obstacles

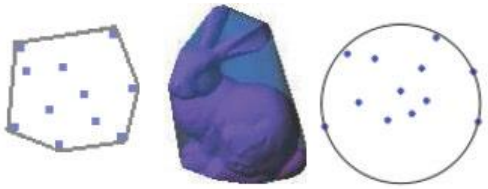


CGAL

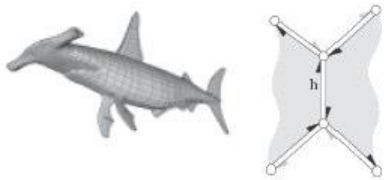


- Computational Geometry Algorithms Library
  - Large library of geometric algorithms
    - Robust code, huge amount of algorithms
    - Users can concentrate on their own domain
  - Open source project
    - Institutional members (Inria, MPI, Tel-Aviv U, Utrecht U, Groningen U, ETHZ, Geometry Factory, FU Berlin, Forth, U Athens)
    - 500,000 lines of C++ code
    - 10,000 downloads/year (+ Linux distributions)
    - 20 active developers
    - 12 months release cycle

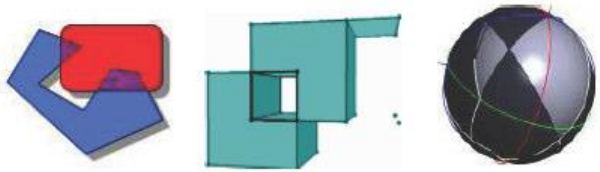
CGAL algorithms and data structures



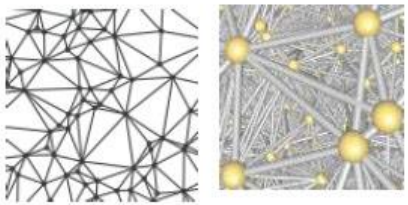
Bounding Volumes



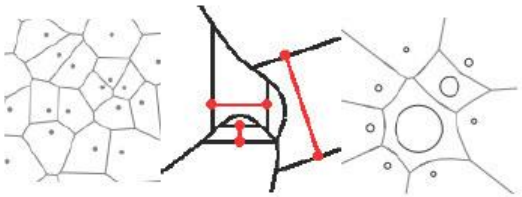
Polyhedral Surface



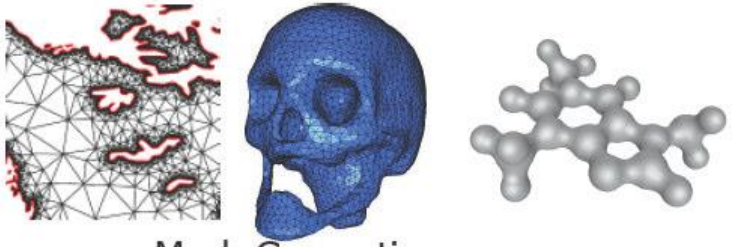
BooleanOperations



Triangulations



Voronoi Diagrams

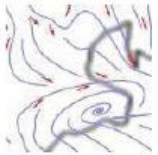


Mesh Generation

Subdivision

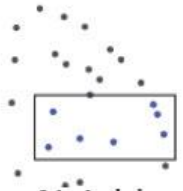
Simplification

Parametrisation

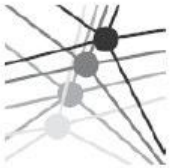


Streamlines

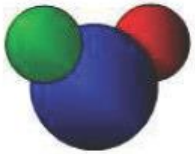
Ridge Detection



Neighbor Search



Kinetic Datastructures



Lower Envelope

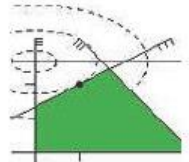
Arrangement

Intersection Detection

Minkowski Sum

PCA

Polytope distance



QP Sover

CGAL : Samples

```
#include "tutorial.h"
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>

int main() {
    Point p( 1.0, 0.0);
    Point q( 1.3, 1.7);
    Point r( 2.2, 6.8);
    switch ( CGAL::orientation( p, q, r)) {
        case CGAL::LEFTTURN:    std::cout << "Left turn.\n";  break;
        case CGAL::RIGHTTURN:   std::cout << "Right turn.\n"; break;
        case CGAL::COLLINEAR:   std::cout << "Collinear.\n";  break;
    }
    return 0;
}
```

Geometric Primitives

Geometric Primitives

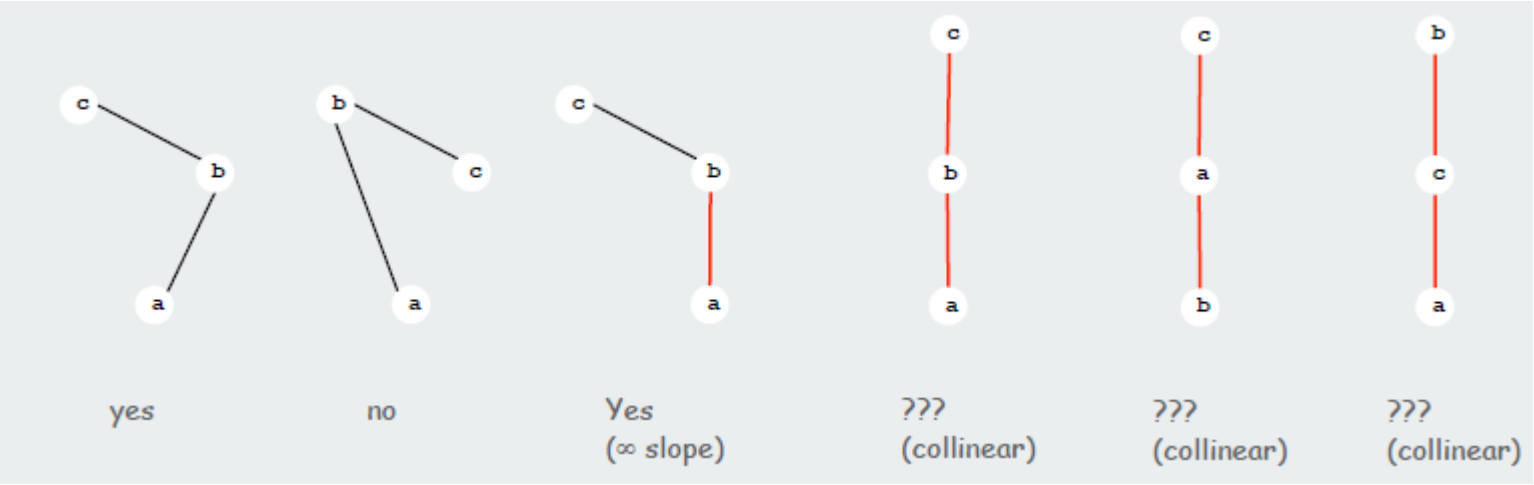
- Point: two numbers (x, y).
- Line: two numbers a and b [ax + by = 1]
- Line segment: two points.
- Polygon: sequence of points.

- Primitive operations.
  - Is a point inside a polygon?
  - Compare slopes of two lines.
  - Distance between two points.
  - Do two line segments intersect?
  - Given three points p1, p2, p3, is p1-p2-p3 a counterclockwise turn?

- Other geometric shapes.
  - Triangle, rectangle, circle, sphere, cone, …
  - 3D and higher dimensions sometimes more complicated.

Implementing CCW

- CCW. Given three point a, b, and c, is a-b-c a counterclockwise turn?
  - Analog of comparisons in sorting.
  - Idea: compare slopes.



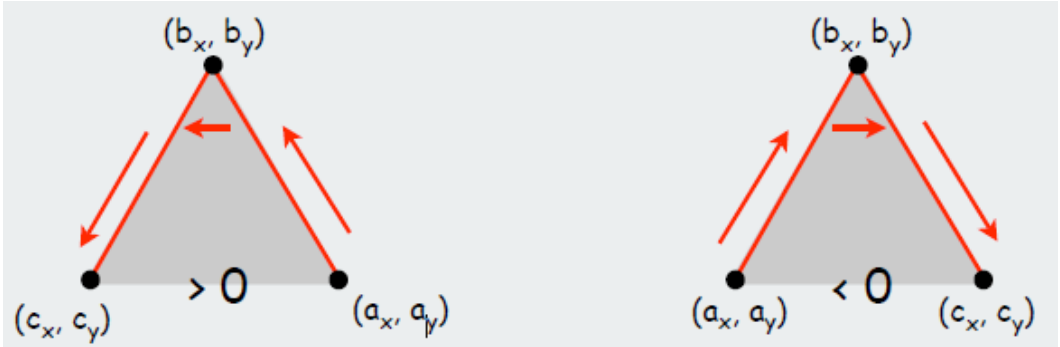
  - Lession : Geometric primitives are tricky to implement.

Implementing CCW(cont.)

- CCW. Given three point a, b, and c, is a-b-c a counterclockwise turn?
  - Determinant gives twice area of triangle.

$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

  - If area > 0 then a-b-c is counterclockwise.
  - If area < 0, then a-b-c is clockwise.
  - If area = 0, then a-b-c are collinear.



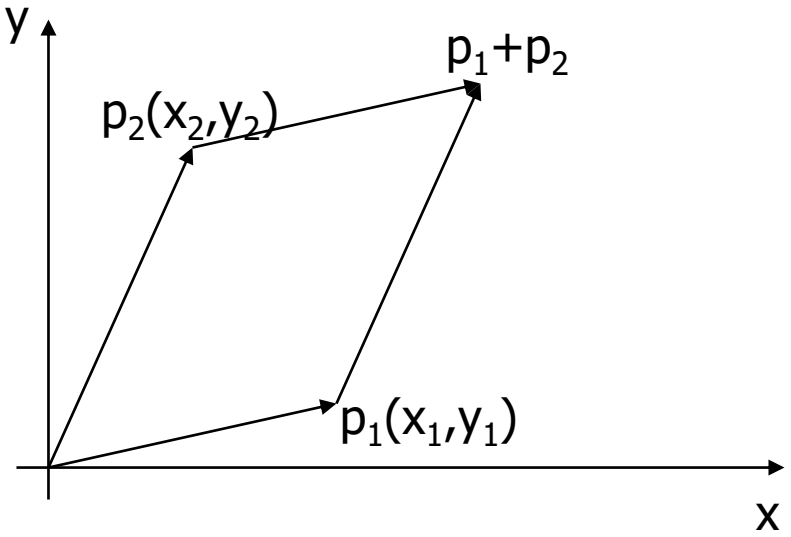
Cross Products

- Can solve many geometric problems using the cross product.
- Two points: $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$
- Cross product of the two points is defined by

  $p_1 \times p_2$ = the determinant of a matrix $\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}$

  $= x_1y_2 - x_2y_1$

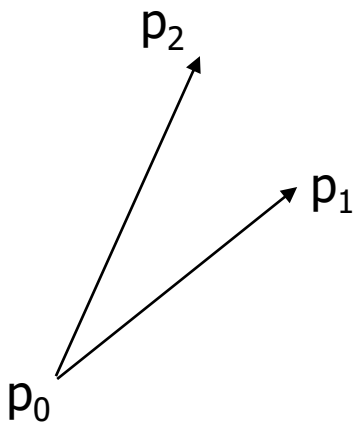
  = the signed area of the parallelogram of four points: $(0,0)$, p_1, p_2, p_1+p_2
- What happens if $p_1 \times p_2 = 0$?
- If $p_1 \times p_2$ is positive then p_1 is clockwise from p_2
- If $p_1 \times p_2$ is negative then p_1 is counterclockwise from p_2



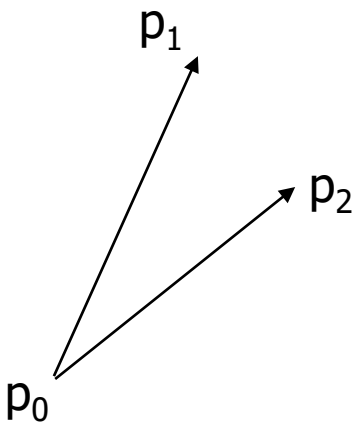
Clockwise or Counterclockwise

- Problem definition
  - Determine whether a directed segment p_0p_1 is clockwise from a directed segment p_0p_2 w.r.t. p_0
- Solution
  1. Map p_0 to (0,0), p_1 to p_1', p_2 to p_2'

     $p_1' = p_1 - p_0, p_2' = p_2 - p_0$
  2. If $p_1' \times p_2' > 0$ then the segment p_0p_1 is clockwise from p_0p_2
  3. else counterclockwise



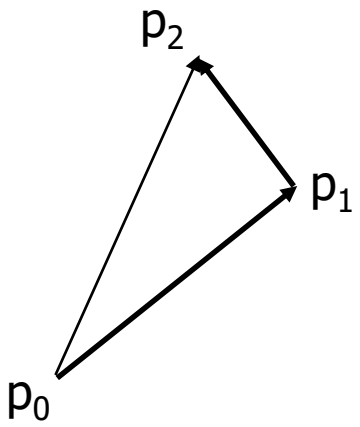
(a) p_0p_2 is counterclockwise from p_0p_1: $(p_2-p_0)\times(p_1-p_0) < 0$



(b) p_0p_2 is clockwise from p_0p_1: $(p_2-p_0)\times(p_1-p_0) > 0$

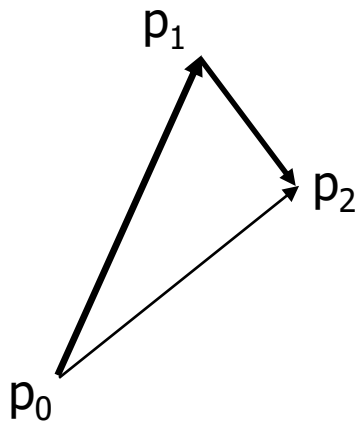
Turn Left or Turn Right

- Problem definition
  - Determine whether two consecutive line segments p_0p_1 and p_1p_2 turn left or right at the common point p_1.
- Solution
  - Determine p_0p_2 is clockwise or counterclockwise from p_0p_1 w.r.t. p_0
  - If counterclockwise then turn left at p_0
  - else turn right at p_0



(a) Turn left at p_1

$(p_2-p_0)x(p_1-p_0) < 0$

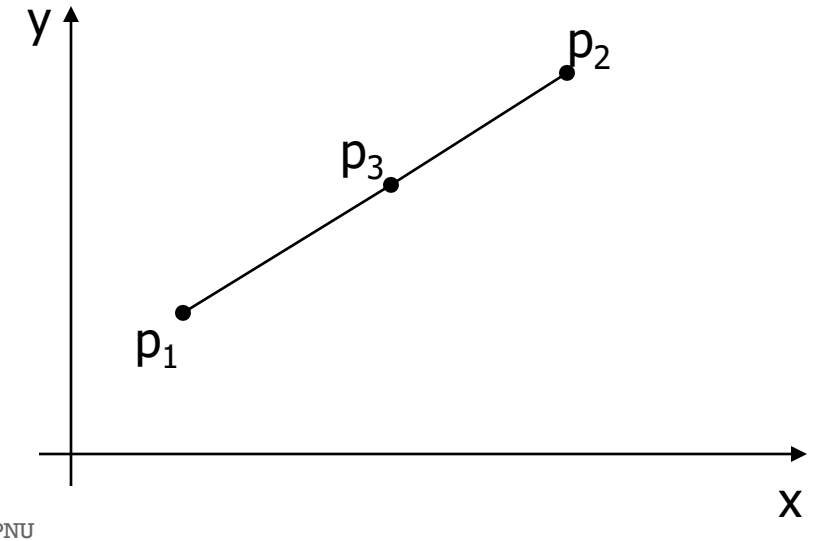


(b) Turn right at p_1

$(p_2-p_0)x(p_1-p_0) > 0$

Line Segment Properties

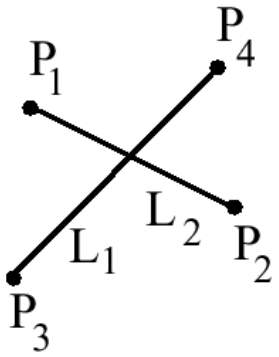
- Two distinct points: $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$
- Define a new point $p_3 = ap_1+(1-a)p_2$, where $0 \le a \le 1$
  - p_3 is any point on the line between p_1 and p_2
  - p_3 is a *convex combination* of p_1 and p_2
- *line segment* p_1p_2 is the set of convex combinations of p_1 and p_2.
  - p_1 and p_2 are the endpoints of segment p_1p_2



Intersection of 2 Line Segments

- Problem: Determine if two given line segments intersect.

Method 1: Find the intersection point of the 2 lines defined by the 2 line segments, and check if it is on both segments.



L_1: $p_1 + t_1(p_2 - p_1)$ $\quad 0 \le t_1 \le 1$

L_2: $p_3 + t_2(p_4 - p_3)$ $\quad 0 \le t_2 \le 1$

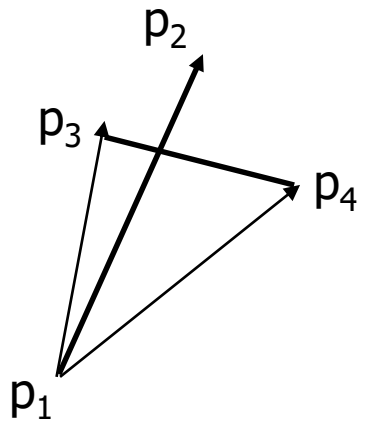
Solve L_1 and L_2 subjext to $0 \le t_1 \le 1$, $0 \le t_2 \le 1$

Solution exists iff L_1 intersects with L_2.

Two Segments Intersect (3)

- Consider two cross products:

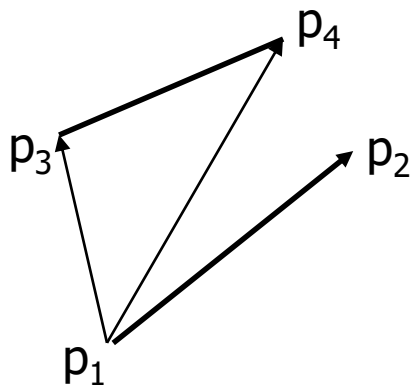
1. $(p_3 - p_1) \times (p_2 - p_1)$
2. $(p_4 - p_1) \times (p_2 - p_1)$



(a) p_1p_2, p_3p_4 intersect:

$(p_3 - p_1) \times (p_2 - p_1) < 0$

$(p_4 - p_1) \times (p_2 - p_1) > 0$



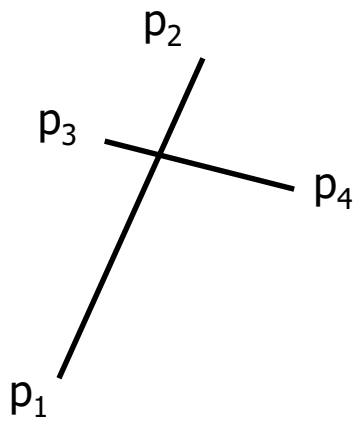
(b) p_1p_2, p_3p_4 do not intersect:

$(p_3 - p_1) \times (p_2 - p_1) < 0$

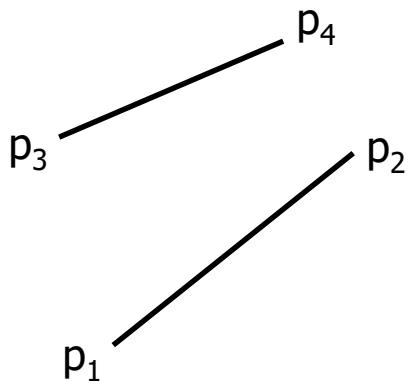
$(p_4 - p_1) \times (p_2 - p_1) < 0$

Two Segments Intersect (2)

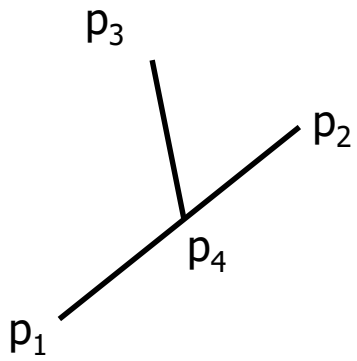
- Five cases



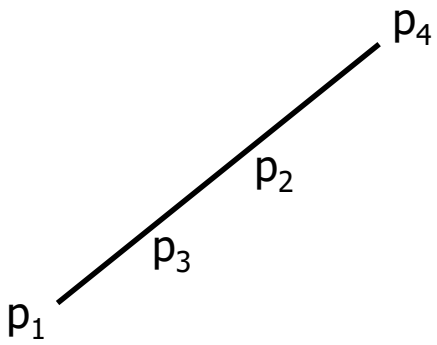
(a) p_1p_2, p_3p_4 intersect



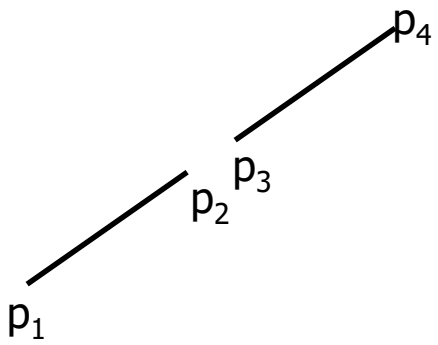
(b) p_1p_2, p_3p_4 do not intersect



(c) p_1p_2, p_3p_4 intersect

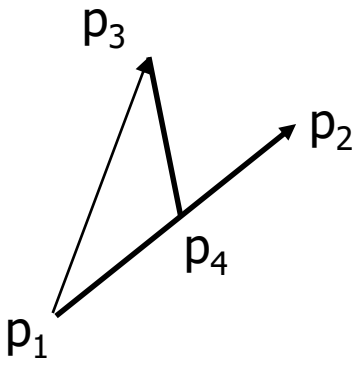


(d) p_1p_2, p_3p_4 intersect



(e) p_1p_2, p_3p_4 do not intersect

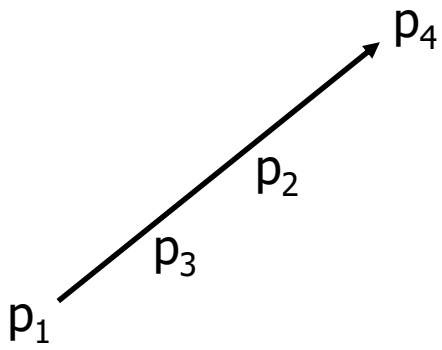
Two Segments Intersect (4)



(c) p_1p_2, p_3p_4 intersect

$(p_3 - p_1) \times (p_2 - p_1) < 0$

$(p_4 - p_1) \times (p_2 - p_1) = 0$



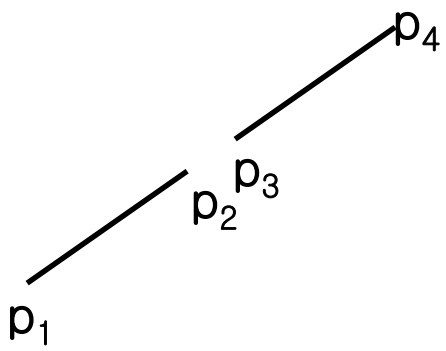
(d) p_1p_2, p_3p_4 intersect

$(p_3 - p_1) \times (p_2 - p_1) = 0$

$(p_4 - p_1) \times (p_2 - p_1) = 0$

Two Segments Intersect (5)

- Case (e)
  - What are the cross products ?
    - $(p_3 - p_1) \times (p_2 - p_1) = 0$
    - $(p_4 - p_1) \times (p_2 - p_1) = 0$
  - The cross products are zero's, but they do not intersect
  - Same result with Case (d)



(e) p_1p_2, p_3p_4 do not intersect

Two Segments Intersect (6)

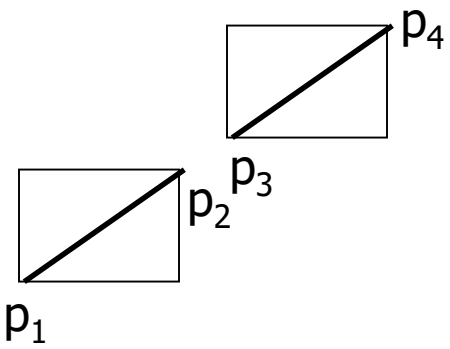
- Case summary
  - (a) p_1p_2, p_3p_4 intersect
    - $(p_3 - p_1) \times (p_2 - p_1) < 0$
    - $(p_4 - p_1) \times (p_2 - p_1) > 0$
  - (b) p_1p_2, p_3p_4 do not intersect
    - $(p_3 - p_1) \times (p_2 - p_1) < 0$
    - $(p_4 - p_1) \times (p_2 - p_1) < 0$
  - (c) p_1p_2, p_3p_4 intersect
    - $(p_3 - p_1) \times (p_2 - p_1) < 0$
    - $(p_4 - p_1) \times (p_2 - p_1) = 0$
  - (d) p_1p_2, p_3p_4 intersect
    - $(p_3 - p_1) \times (p_2 - p_1) = 0$
    - $(p_4 - p_1) \times (p_2 - p_1) = 0$
  - (e) p_1p_2, p_3p_4 do not intersect
    - $(p_3 - p_1) \times (p_2 - p_1) = 0$
    - $(p_4 - p_1) \times (p_2 - p_1) = 0$

Two Segments Intersect - Algorithm

- Two_line_segments_intersect (p_1p_2, p_3p_4)
  1. $(x_1', y_1') = (\min(x_1, x_2), \min(y_1, y_2))$
  2. $(x_2', y_2') = (\max(x_1, x_2), \max(y_1, y_2))$
  3. $(x_3', y_3') = (\min(x_3, x_4), \min(y_3, y_4))$
  4. $(x_4', y_4') = (\max(x_3, x_4), \max(y_3, y_4))$
  5. if not (($x_3' \le x_2'$) and ($x_1' \le x_4'$) and ($y_3' \le y_2'$) and ($y_1' \le y_4'$)) then
  6. return false // case (e) and more
  7. else
  8. if $(p_3-p_1) \times (p_2-p_1)$ or $(p_4-p_1) \times (p_2-p_1)$ are zero then
  9. return true // case (c) and (d)
  10. else if $(p_3-p_1) \times (p_2-p_1)$ and $(p_4-p_1) \times (p_2-p_1)$ have different sign then
  11. return true // case (a)
  12. else
  13. return false // case (b)

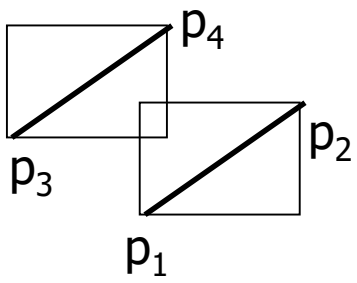
Bounding Boxes

- Definition
  - Given a geometric object, the bounding box is defined by the smallest rectangle that contains the object
- Given two line segments p_1p_2 and p_3p_4
- The bounding box of the line segment p_1p_2
  - The rectangle with lower left point=(x_1', y_1')=$(\min(x_1,x_2), \min(y_1,y_2))$ and
  - upper right point=(x_2', y_2')= $(\max(x_1,x_2), \max(y_1,y_2))$
- The bounding box of the line segment p_3p_4 is
  - The rectangle with lower left point=(x_3', y_3')=$(\min(x_3,x_4), \min(y_3,y_4))$ and
  - upper right point=(x_4', y_4')=$(\max(x_3,x_4), \max(y_3,y_4))$



Bounding Boxes

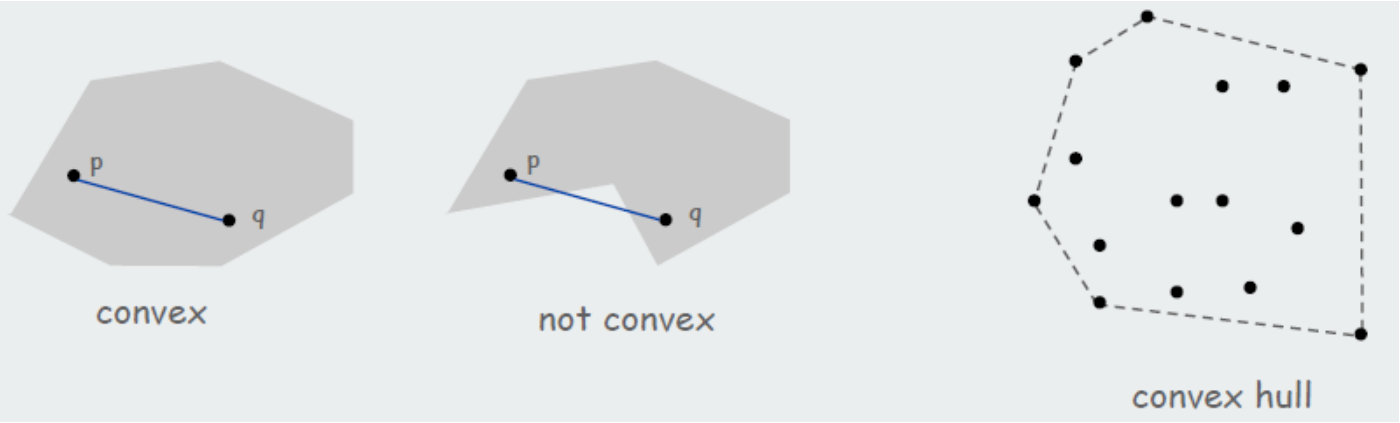
- What is the condition for the two bounding boxes intersect?
  - $(x_3' \le x_2')$ and $(x_1' \le x_4')$ and $(y_3' \le y_2')$ and $(y_1' \le y_4')$
- If two bounding boxes do not intersect, then the two line segments do not intersect.
- But it is not always true that
  - if two bounding boxes intersect, then the two line segments intersect
  - e.g., the figure



Convex Hull

Convex Hull

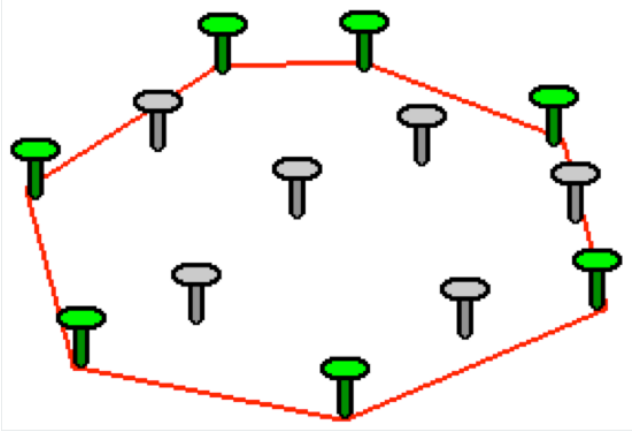
- A set of points is convex if for any two points p and q in the set, the line segment pq is completely in the set.
- Convex hull. Smallest convex set containing all the points.



- Properties
  - "Simplest" shape that approximates set of points.
  - Shortest (perimeter) fence surrounding the points.
  - Smallest (area) convex polygon enclosing the points.

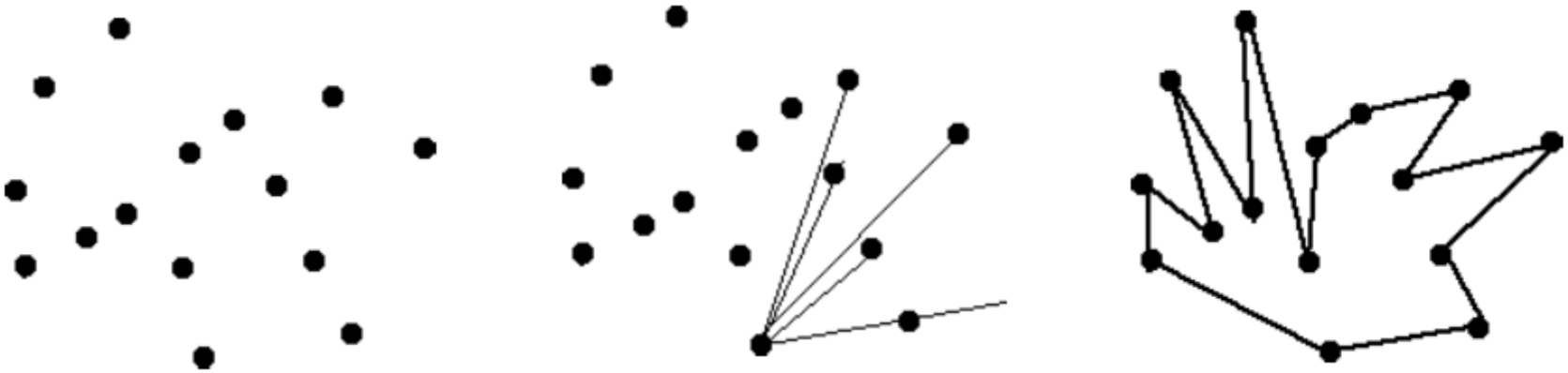
Mechanical Solution

- Mechanical algorithm. Hammer nails perpendicular to plane; stretch elastic rubber band around points.



http://www.dfanning.com/math_tips/convexhull_1.gif

Illustration



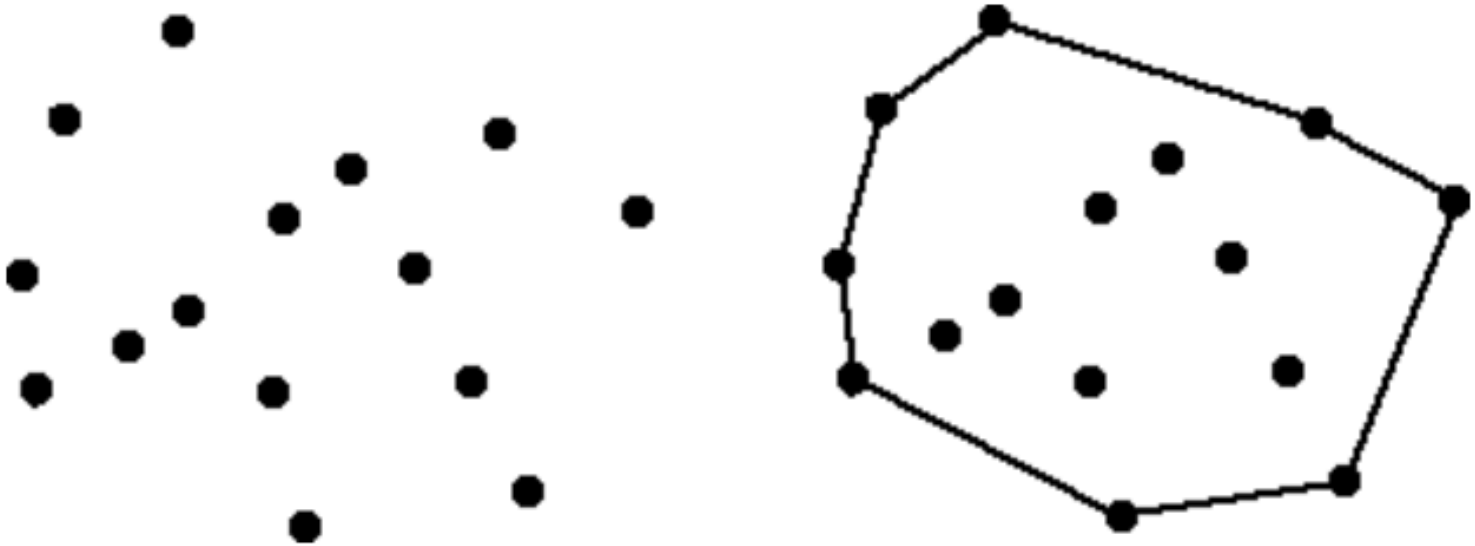
Computing the Angle

- angle = $\tan^{-1}(dy/dx)$.
- It make sense to use a function that is much easier to compute but has the same ordering property as the arctangent.

```
function theta(p1, p2: point): real;
var dx, dy, ax, ay: integer;  t: real;
begin
  dx := p2.x - p1.x;  ax := abs(dx);
  dy := p2.y - p1.y;  ay := abs(dy);
  if ax+ay = 0 then t := 0 else t := dy/(ax+ay);
  if dx < 0 then t := 2 - t else if dy < 0 then t := 4 +  t;
  theta := t*90.0;   /* returns a no. between 0 and 360 */
end;
```

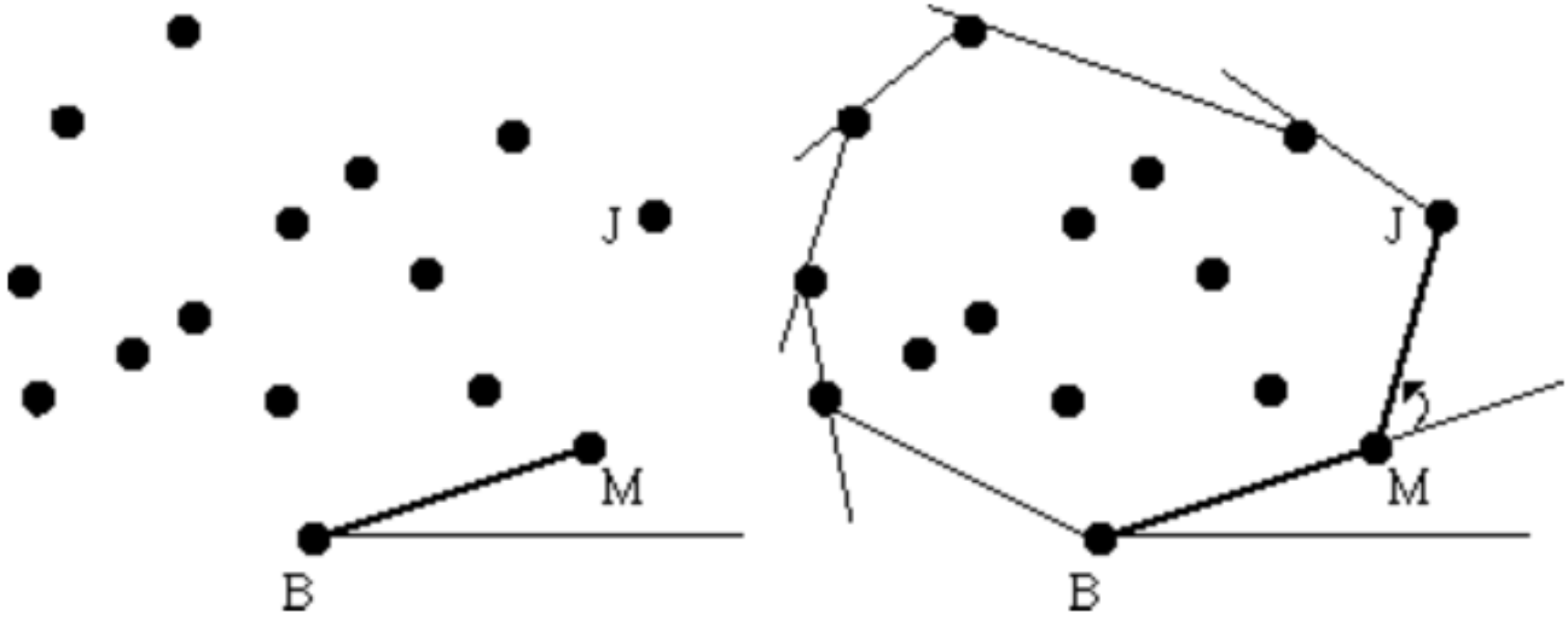
Finding the Convex Hull

- A convex hull of n given points is defined as the smallest convex polygon containing them all



- Computing Convex hull is closed related to sorting. (Why?)

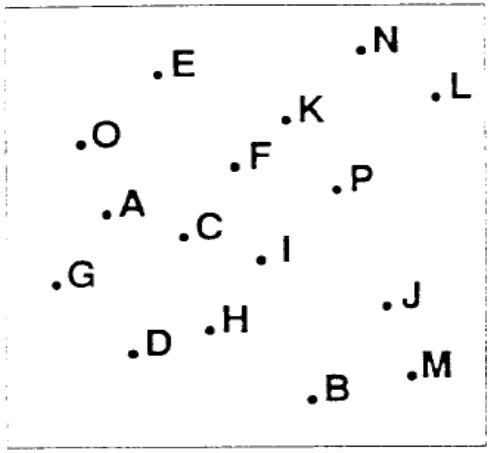
Idea

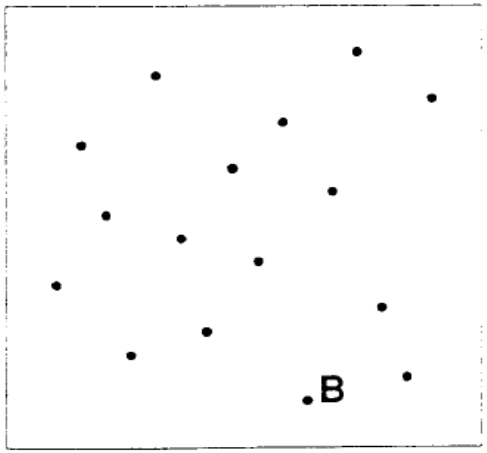


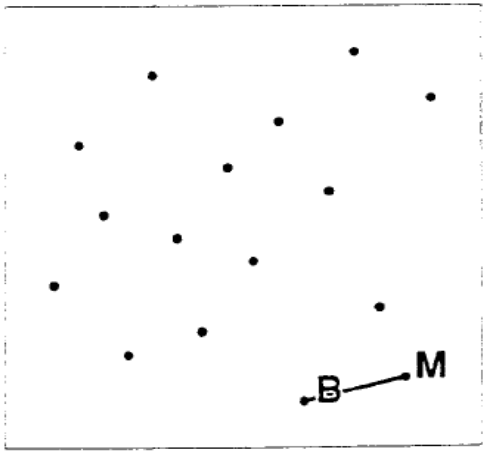
Method 1: Package Wrapping

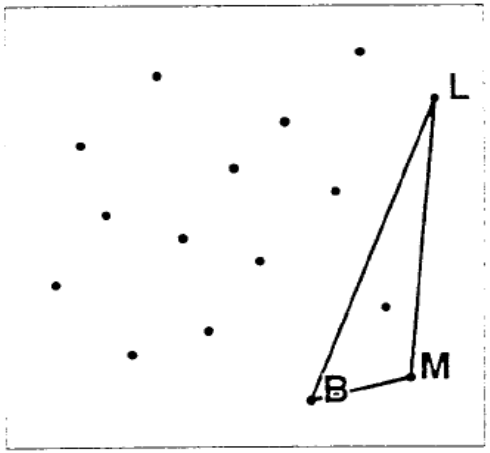
1. Find an anchor point (say, the one with the smallest y value).
2. Take a horizontal ray in the positive direction, and sweep it upward until hitting another point, this point must be on the hull.
3. Anchor at the newly found point and continue sweeping until hitting another point.
4. Repeat step 3 until the "package" is fully "wrapped"

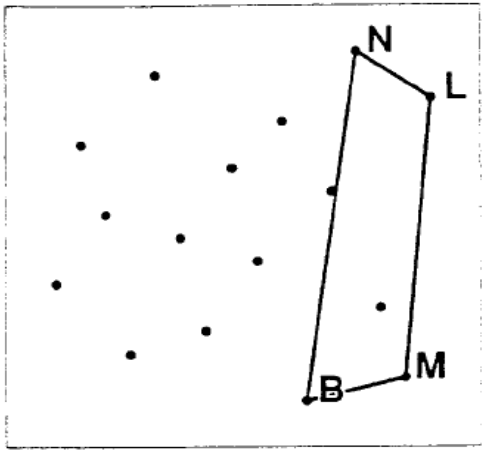
- Time complexity: If there are m vertices on the hull, then package wrapping technique requires about mn steps: $(n\text{-}1) + (n\text{-}2) + \ldots + (n\text{-}m) = mn - m(m+1)/2$.

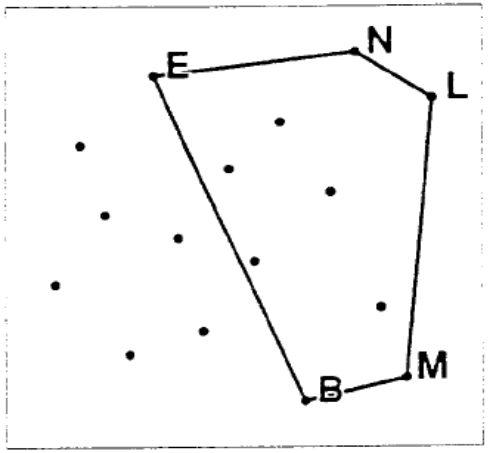

E
N
L
K
O
F
P
A
C
I
G
J
H
D
M
B

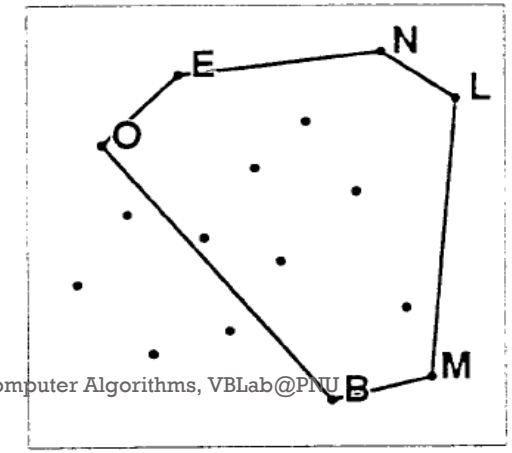

B

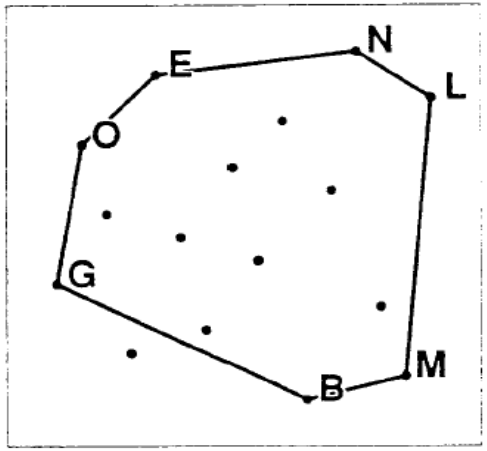

B
M

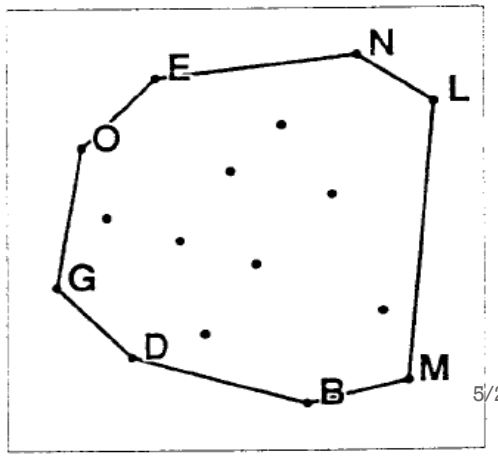

L
B
M


N
L
B
M


E
N
L
B
M

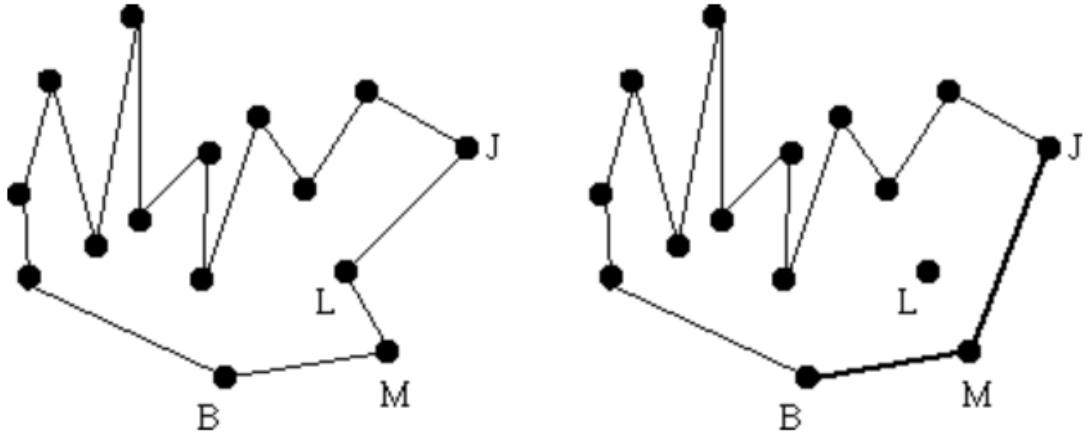

E
N
L
O
B
M


E
N
L
O
G
B
M

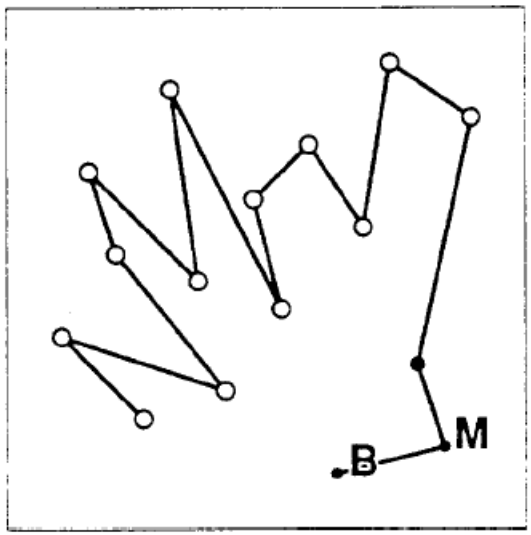

E
N
L
O
G
D
B
M

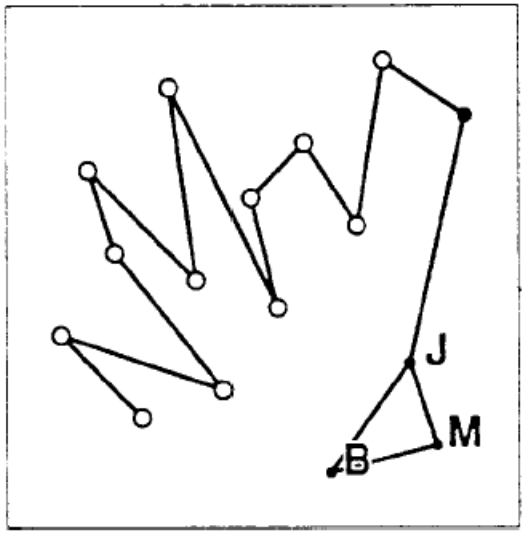
Method 2: The Graham Scan

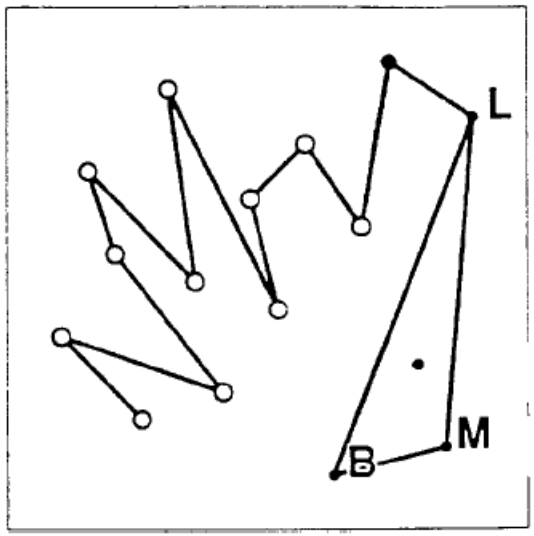
1. Construct a simple closed path from the given n points.
2. Select an extreme point, start a scan on all the points. If an a ngle ABC is found to be reflex, Eliminate point B from the co nvex hull.

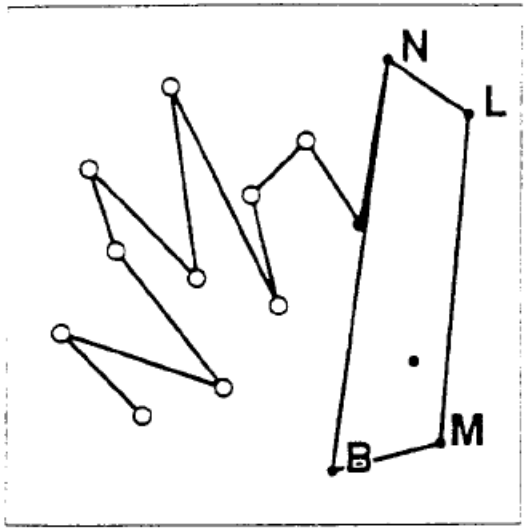


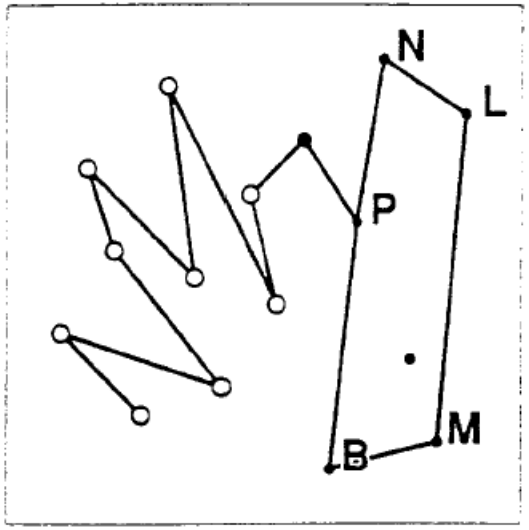
- Time complexity: O(n lg n). After the sort, the Graham scan is a linear-time process.

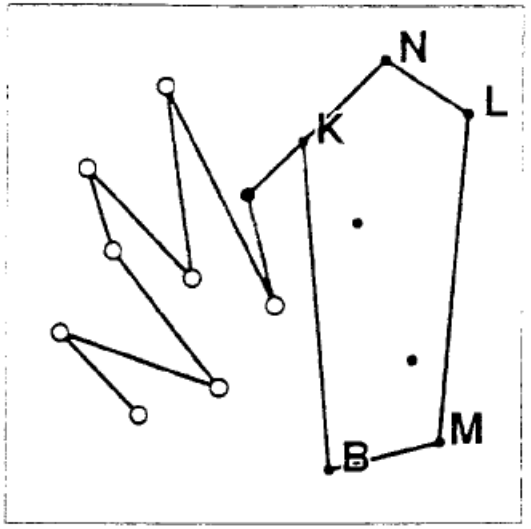

B
M

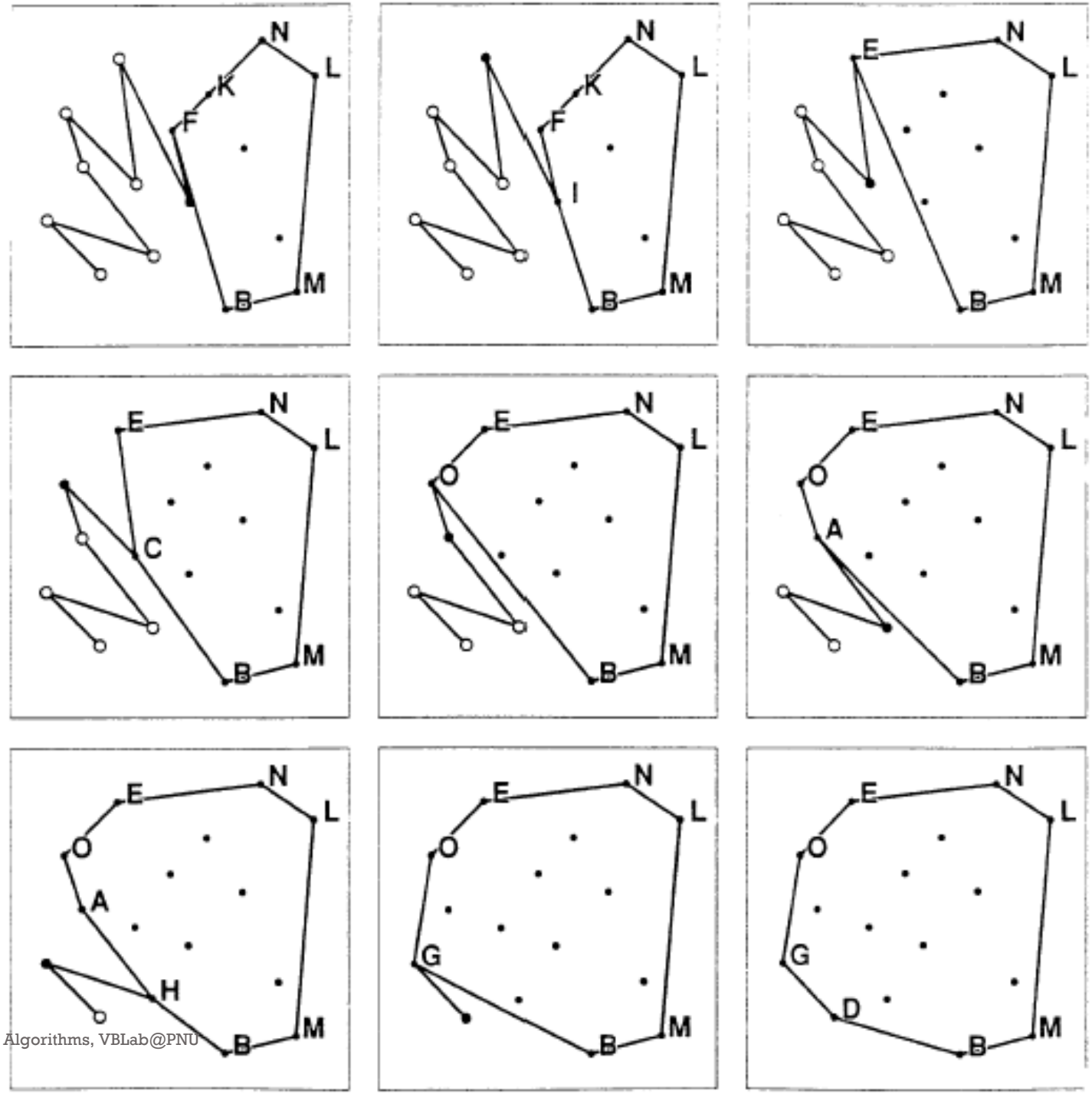

J
B
M


L
B
M


N
L
B
M


N
L
P
B
M


N
L
K
B
M


N
L
K
F
B
M
N
L
K
F
I
B
M
N
E
L
B
M
N
E
L
C
B
M
N
E
L
O
B
M
N
E
L
O
A
B
M
N
E
L
O
A
H
B
M
N
E
L
O
G
B
M
N
E
L
O
G
D
B
M

Closest Pair

Closest pair problem

- Given: N points in the plane
- Goal: Find a pair with smallest Euclidean distance between them
- Fundamental geometric primitive.
  - Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.
- Brute force.
  - Check all pairs of points p and q with $\theta(N2)$ distance calculations

Voronoi Diagrams

1854 Cholera Outbreak, Golden Square, London

- Life-or-death question:
  - Given a new cholera patient p, which water pump is closest to p's home?

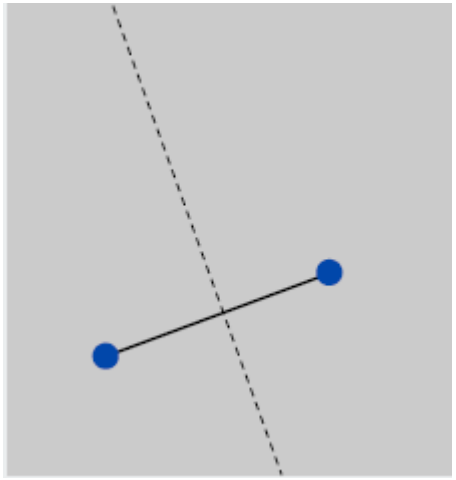
Nearest-neighbor problem

- Input.
  - N Euclidean points.
- Nearest neighbor problem.
  - Given a query point p, which one of original N points is closest to p?

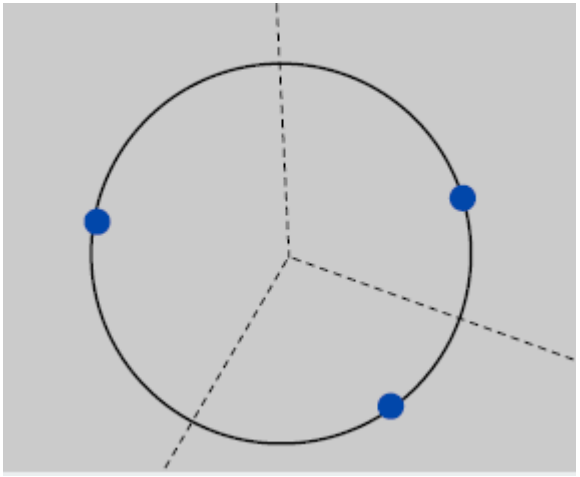
algorithm	preprocessing	Query
Brute	1	N
Goal	N log N	log N

Voronoi Diagram

- Voronoi region. Set of all points closest to a given point.
- Voronoi diagram. Planar subdivision delineating Voronoi regions.
- Fact. Voronoi edges are perpendicular bisector segments.



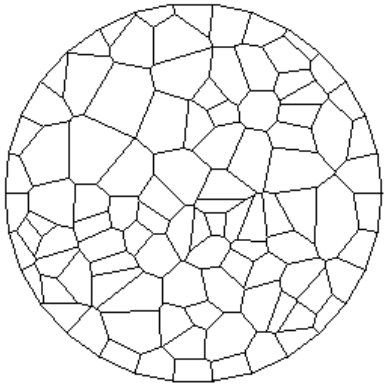
Voronoi of 2 points
(perpendicular bisector)

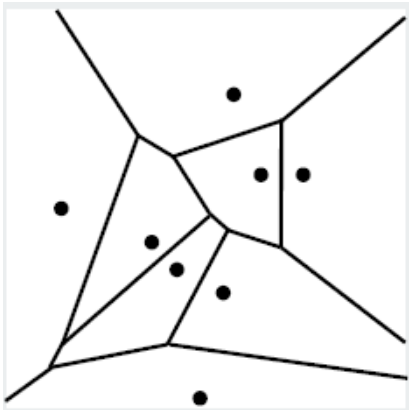


Voronoi of 3 points
(passes through circumcenter)

Voronoi Diagram

- Voronoi region. Set of all points closest to a given point.
- Voronoi diagram. Planar subdivision delineating Voronoi regions.
- Fact. Voronoi edges are perpendicular bisector segments.





Quintessential nearest neighbor data structure.

Voronoi Diagram : Applications

- Toxic waste dump problem. N homes in a region. Where to locate nuclear power plant so that it is far away from any home as possible?

looking for largest empty circle
(center must lie on Voronoi diagram)

- Path planning. Circular robot must navigate through environment with N obstacle points. How to minimize risk of bumping into a obstacle?

robot should stay on Voronoi diagram of obstacles

Reference: J. O'Rourke. Computational Geometry.

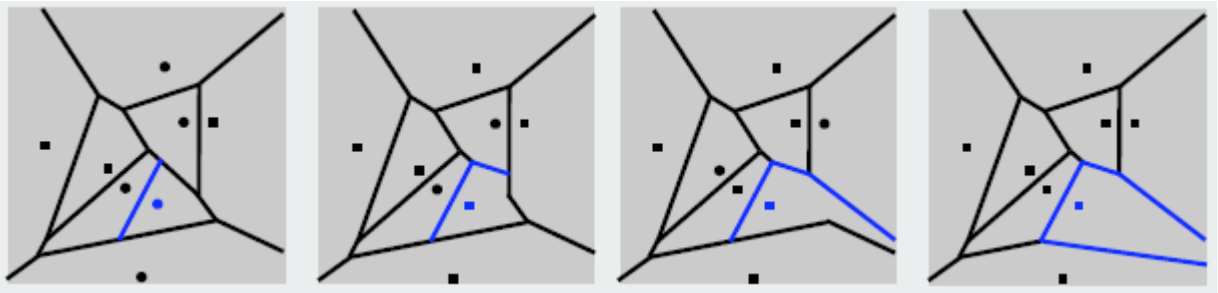
Voronoi Diagram : More Applications

- Anthropology. Identify influence of clans and chiefdoms on geographic regions.
- Astronomy. Identify clusters of stars and clusters of galaxies.
- Biology, Ecology, Forestry. Model and analyze plant competition.
- Cartography. Piece together satellite photographs into large "mosaic" maps.
- Crystallography. Study Wigner-Setiz regions of metallic sodium.
- Data visualization. Nearest neighbor interpolation of 2D data.
- Finite elements. Generating finite element meshes which avoid small angles.
- Fluid dynamics. Vortex methods for inviscid incompressible 2D fluid flow.
- Geology. Estimation of ore reserves in a deposit using info from bore holes.
- Geo-scientific modeling. Reconstruct 3D geometric figures from points.
- Marketing. Model market of US metro area at individual retail store level.
- Metallurgy. Modeling "grain growth" in metal films.
- Physiology. Analysis of capillary distribution in cross-sections of muscle tissue.
- Robotics. Path planning for robot to minimize risk of collision.
- Typography. Character recognition, beveled and carved lettering.
- Zoology. Model and analyze the territories of animals.

References: http://voronoi.com, http://www.ics.uci.edu/~eppstein/geom.html

Adding a Point to Voronoi Diagram

- Challenge. Compute Voronoi.
- Basis for incremental algorithms: region containing point gives points to check to compute new Voronoi region boundaries.



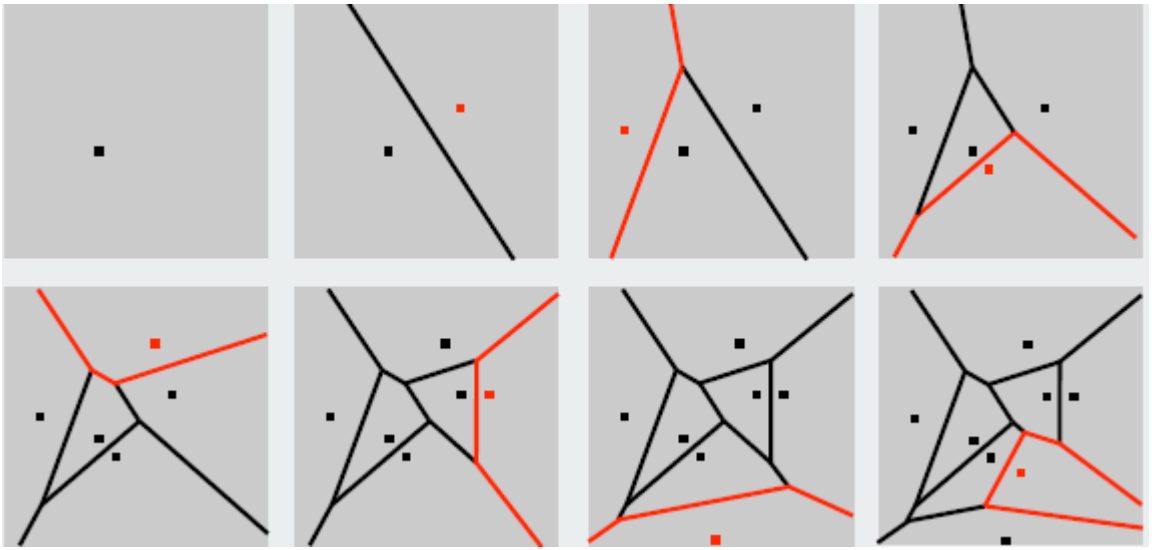
- How to represent the Voronoi diagram?
- Use multilist associating each point with its Voronoi neighbors

- How to find region containing point?
- Use Voronoi itself (possible, but not easy!)

Randomized Incremental Voronoi Algorithm

- Add points (in random order).
  - Find region containing point.
  - Update neighbor regions, create region for new point.

using Voronoi itself

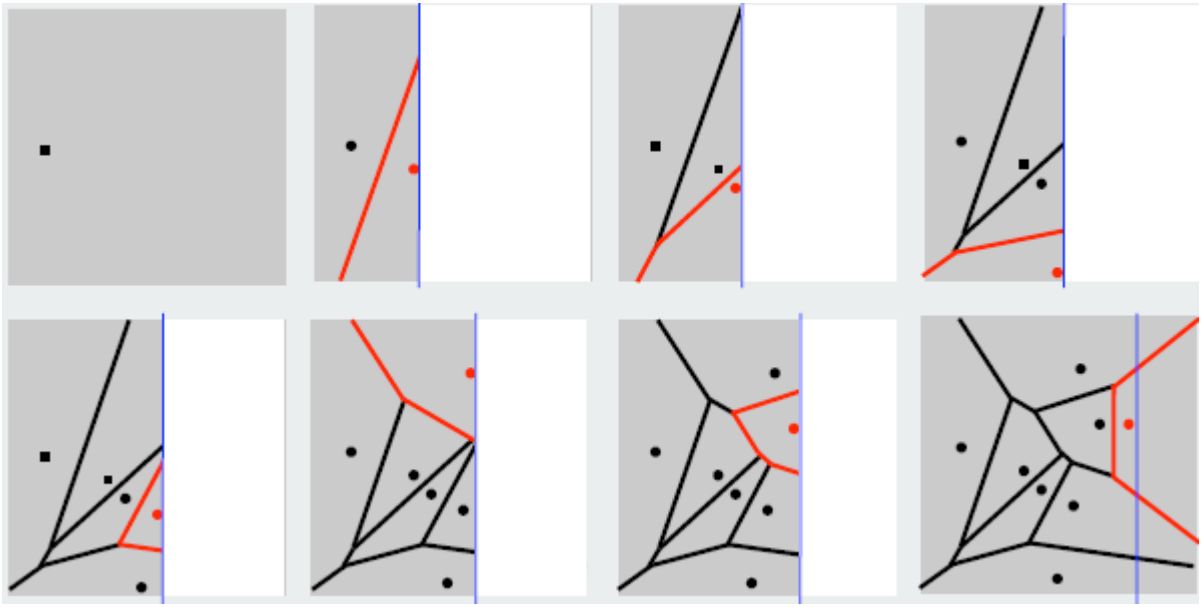


  - Running time: O(N log N) on average.

Not an elementary algortihm

Sweep-line Voronoi algorithm

- Presort points on x-coordinate
- Eliminates point location problem



Fortune's Algorithm

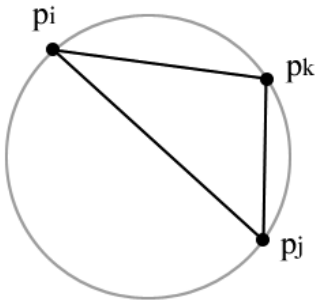
- Industrial-strength Voronoi implementation
  - Sweep-line algorithm
  - O(N log N) time
  - properly handles degeneracies

algorithm	preprocessing	Query
Brute	1	N
Goal	N log N	log N

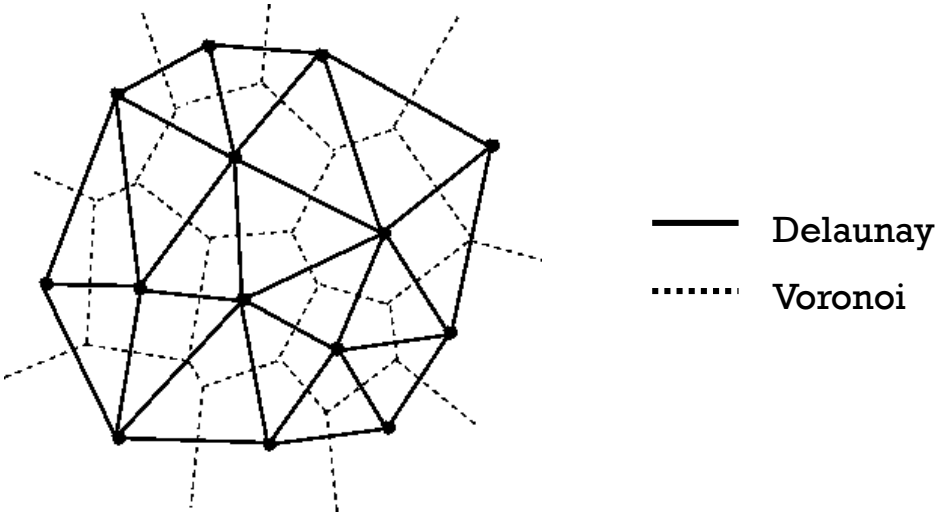
https://en.wikipedia.org/wiki/Fortune%27s_algorithm

Delaunay Triangulation

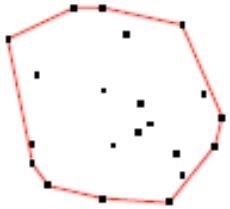
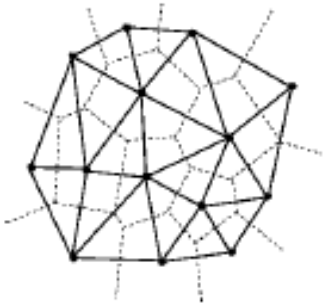
- Delaunay triangulation. Triangulation of N points such that no point is inside circumcircle of any other triangle.



- Fact 0. It exists and is unique (assuming no degeneracy).
- Fact 1. Dual of Voronoi (connect adjacent points in Voronoi
- Fact 2. No edges cross ⇒ O(N) edges.
- Fact 3. Maximizes the minimum angle for all triangular ele
- Fact 4. Boundary of Delaunay triangulation is convex hull.
- Fact 5. Shortest Delaunay edge connects closest pair of points.



Geometric algorithms summary: Algorithms of the day

		brute	ingenuity
convex hull		N^2	N log N
closest pair		N^2	N log N
Voronoi/Delauney		N^4	N log N