# Geometric Algorithms

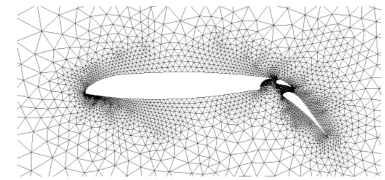
Reference: Chapters 24-25, Algorithms in C, 2nd Edition, Robert Sedgewick.



## Geometric Algorithms

Applications.

- Data mining.
- VLSI design.
- Computer vision.
- Mathematical models.
- Astronomical simulation.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).
- Models of physical world (maps, architecture, medical imaging).



airflow around an aircraft wing

Reference: http://www.ics.uci.edu/~eppstein/geom.html

History.

- Ancient mathematical foundations.
- Most geometric algorithms less than 25 years old.



## Geometric Primitives

Point: two numbers (x, y).
Line: two numbers a and b [ax + by = 1] ← any line not through origin
Line segment: four numbers $(x_1, y_1)$, $(x_2, y_2)$.
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 $p_1$, $p_2$, $p_3$, is $p_1$-$p_2$-$p_3$ a counterclockwise turn?

Other geometric shapes.

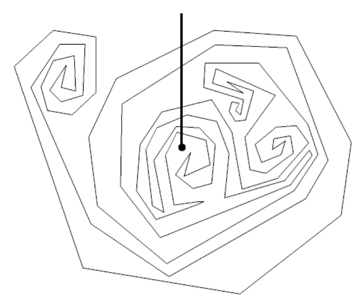
- Triangle, rectangle, circle, sphere, . . .
- 3D and higher dimensions sometimes more complicated.



## Inside, Outside

Jordan curve theorem. Any continuous simple closed curve cuts the plane in exactly two pieces: the inside and the outside.

Is a point inside a simple polygon?



Reference: http://www.ics.uci.edu/~eppstein/geom.html

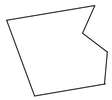
Application. Draw a colored polygon on the screen.

## Warning: Intuition May Mislead

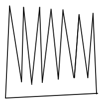
Warning: intuition may be misleading.

- Humans have spatial intuition in 2D and 3D.

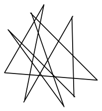
Is a given polygon simple?



| | | | | | |
|---|---|---|---|---|---|
| 1 | 6 | 5 | 8 | 7 | 2 |
| 7 | 8 | 6 | 4 | 2 | 1 |



| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 1 | 2 | 18 | 4 | 18 | 4 | 19 | 4 | 19 | 4 | 20 | 3 | 20 | 3 | 20 |



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 3 | 7 | 2 | 8 | 8 | 3 | 4 |
| 6 | 5 | 15 | 1 | 11 | 3 | 14 | 2 | 16 |

we think of this

algorithm sees this



## Warning: Intuition May Mislead

Warning: intuition may be misleading.

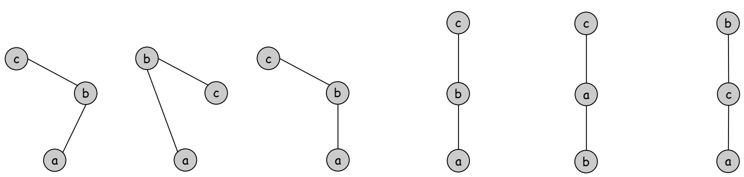
- Humans have spatial intuition in 2D and 3D.
- Computers do not.
- Neither has good intuition in higher dimensions!



## 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.




Yes

No

Yes
(∞ slope)

???
(collinear)

???
(collinear)

???
(collinear)

Lesson. Geometric primitives are tricky to implement.

- Dealing with degenerate cases.
- Coping with floating point precision.



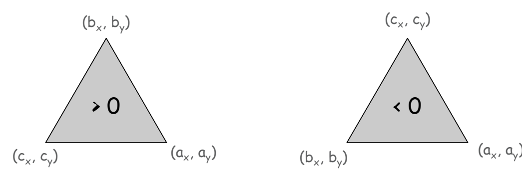
## Implementing CCW

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.
- Avoids floating point precision when coordinates are integral.



## Immutable Point ADT

```
public final class Point {
   public final int x;
   public final int y;

   public Point(int x, int y) { this.x = x; this.y = y; }

   public double distanceTo(Point q) {
      Point p = this;
      return Math.hypot(p.x - q.x, p.y - q.y);
   }

   public static int ccw(Point a, Point b, Point c) {
      double area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
      if      (area2 < 0) return -1;
      else if (area2 > 0) return +1;
      else                return  0;
   }

   public static boolean collinear(Point a, Point b, Point c) {
      return ccw(a, b, c) == 0;
   }
}
```



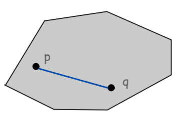
# Convex Hull



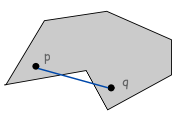
## Convex Hull

A set of points is convex if for any two points p and q, the line segment $\overline{pq}$ is completely in the set.

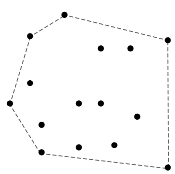
Convex hull. Smallest convex set containing the points.




convex




not convex



convex hull

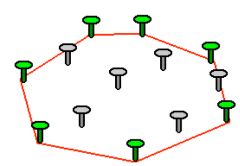
Properties.

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



## Convex Hull

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



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

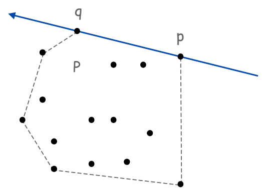
Parameters.

- N = # points.
- M = # points on the hull.

## Brute Force

Observation 1. Edges of convex hull connect pairs of points in P.

Observation 2. Edge pq is on convex hull if all other points are counterclockwise of pq.



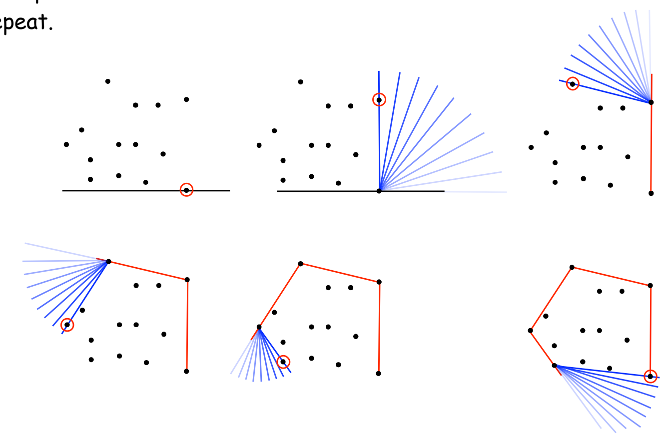

$O(N^3)$ algorithm. For all pairs of points p, q, check whether pq is an edge of convex hull.



## Package Wrap

Package wrap.

- Start with point with smallest y-coordinate.
- Rotate sweep line around current point in ccw direction.
- First point hit is on the hull.
- Repeat.

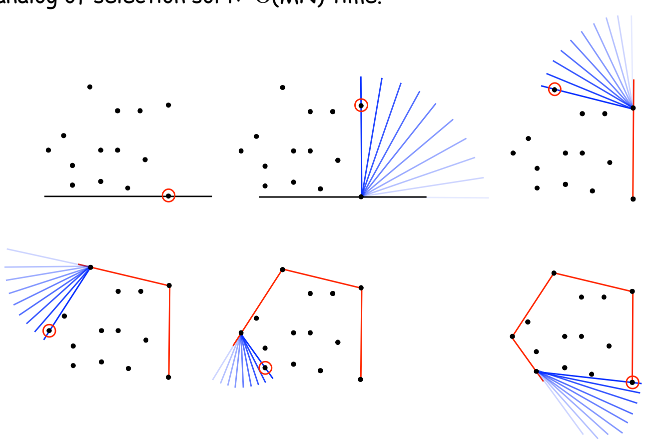




## Package Wrap

Implementation.

- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- 2D analog of selection sort: $\Theta(MN)$ time.

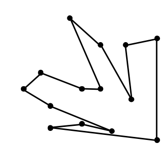


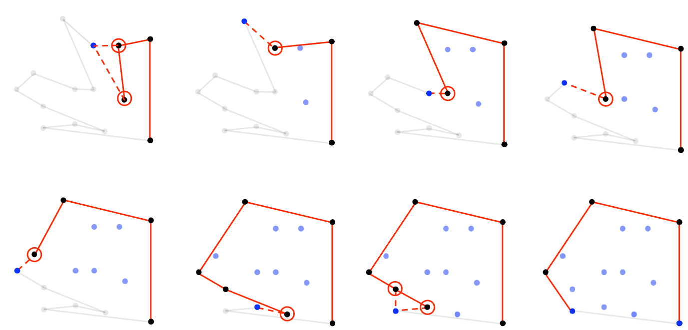


## Graham Scan: Example

Graham scan.

- Choose point p with smallest y-coordinate.
- Sort points by polar angle with p to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.



## Graham Scan: Example

### Implementation.

- Input: `p[1], p[2], ..., p[N]` are points.
- Output: `M` and rearrangement so that `p[1], ..., p[M]` is convex hull.
- Total cost: O(N log N) for sort and O(N) for rest.

why?

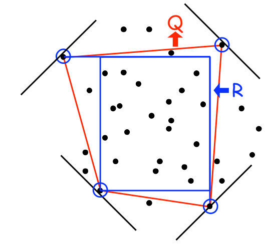
```
// preprocess so that p[1] has smallest y-coordinate
// sort by angle with p[1]

points[0] = points[N];  // sentinel
int M = 2;
for (int i = 3; i <= N; i++) {
   while (Point.ccw(p[M], p[M-1], p[i]) >= 0) {
      M--;                // back up to include i on hull
   }
   M++;
   swap(points, M, i);  // add i to putative hull
}
```



## Quick Elimination

### Quick elimination.

- Choose a quadrilateral Q or rectangle R with 4 points as corners.
- If point is inside, can eliminate.
  - 4 CCW tests for quadrilateral
  - 4 comparisons for rectangle

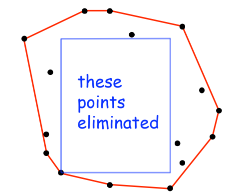



### Three-phase algorithm

- Pass through all points to compute R.
- Eliminate points inside R.
- Find convex hull of remaining points.

### Impact.

- In practice, can eliminate almost all points in O(N) time.
- Improves overall performance.






## Convex Hull Algorithms Costs Summary

Asymptotic cost to find M-point hull in N-point set

| Algorithm | Running Time |
|---|---|
| Package wrap | N M |
| Graham scan | N log N |
| Quickhull | N log N |
| Mergehull | N log N |
| Sweep line | N log N |
| Quick elimination | N * |
| Best in theory | N log M ← output sensitive running time |

* assumes "reasonable" point distribution



## How Many Points on the Hull?

### Parameters.

- N = # points.
- M = # points on the hull.

### How many points on hull?

- Worst case: N.
- Average case: difficult problems in stochastic geometry.

### Uniform.

- On a circle: N.
- In a disc: $N^{1/3}$.
- In a convex polygon with O(1) edges: log N.

## Lower Bounds

Models of computation.

- Comparison based: compare coordinates.
  (impossible to compute convex hull in this model of computation)

$$less(a,\ b) \ = \ (a_x < b_x) \ \| \ ((a_x == b_x) \ \&\& \ (a_y < b_y))$$

- Quadratic decision tree model: compute any quadratic function of the coordinates and compare against 0.

$$ccw(a,\ b,\ c) \ = \ a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y$$

Theorem. [Andy Yao 1981] In quadratic decision tree model, any convex hull algorithm requires Ω(N log N) operations.

even if hull points are not required to be output in counterclockwise order

higher degree polynomial tests don't help either [Ben-Or 1983]



# Closest Pair of Points



## Closest Pair of Points

Closest pair. Given N points in the plane, 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.

fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points p and q with $\Theta(N^2)$ comparisons.

1-D version. O(N log N) easy if points are on a line.

Assumption. No two points have same x coordinate.

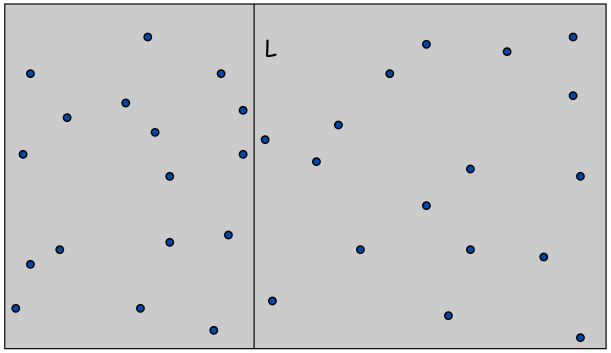
to make presentation cleaner



## Closest Pair of Points

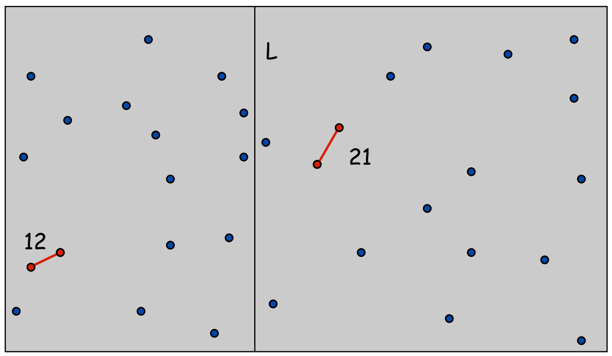
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}$N points on each side.



## Closest Pair of Points

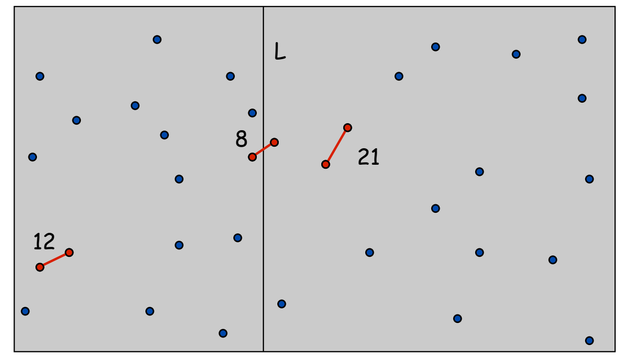
Algorithm.
- Divide: draw vertical line L so that roughly $\frac{1}{2}N$ points on each side.
- Conquer: find closest pair in each side recursively.






## Closest Pair of Points

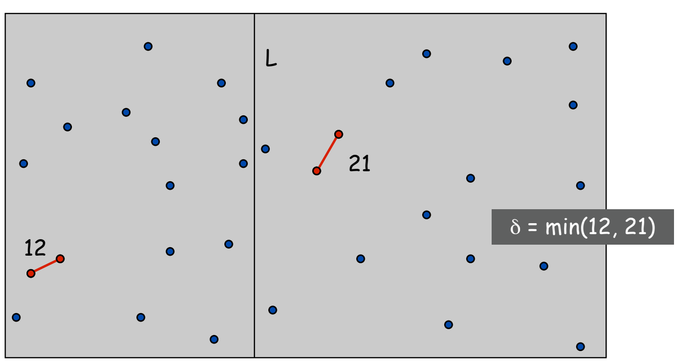
Algorithm.
- Divide: draw vertical line L so that roughly $\frac{1}{2}N$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. ← seems like $\Theta(N^2)$
- Return best of 3 solutions.






## Closest Pair of Points

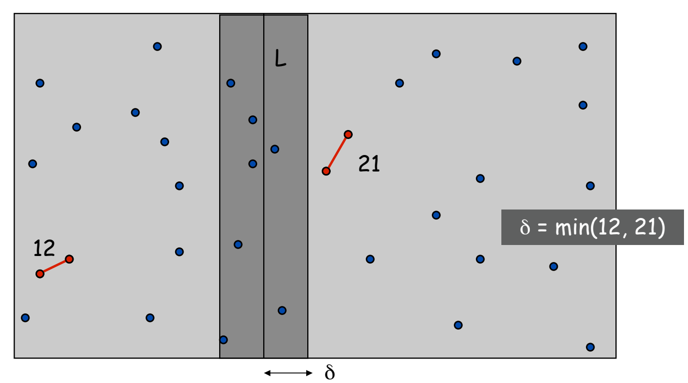
Find closest pair with one point in each side, assuming that distance < $\delta$.






## Closest Pair of Points

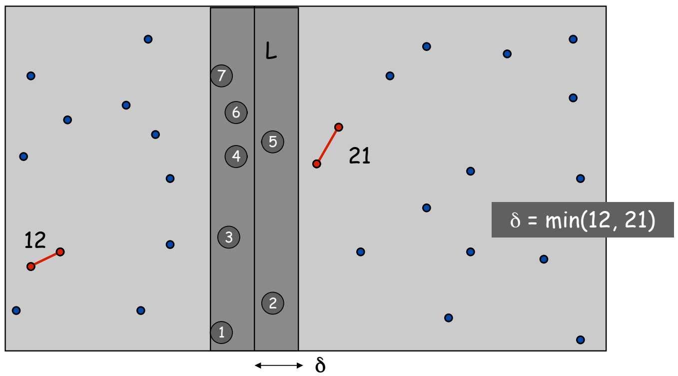
Find closest pair with one point in each side, assuming that distance < $\delta$.
- Observation: only need to consider points within $\delta$ of line L.



## Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line L.
- Sort points in $2\delta$-strip by their y coordinate.

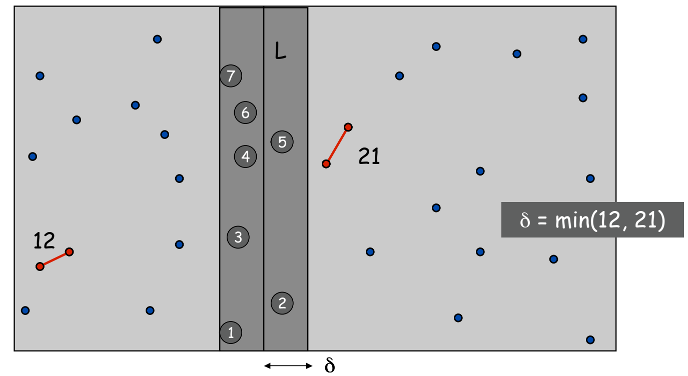





## Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line L.
- Sort points in $2\delta$-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!






## Closest Pair of Points

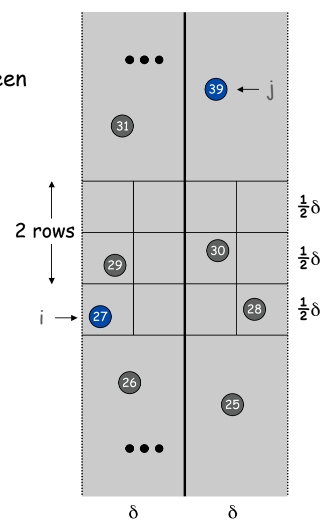
Def. Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest y-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Still true if we replace 12 with 7.






## Closest Pair Algorithm

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points          O(N log N)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                                  2T(N / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L       O(N)

   Sort remaining points by y-coordinate.                        O(N log N)

   Scan points in y-order and compare distance between           O(N)
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

## Closest Pair of Points: Analysis

Running time.

$$T(N) \le 2T(N/2) + O(N \log N) \Rightarrow T(N) = O(N \log^2 N)$$

Q. Can we achieve O(N log N)?

A. Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

$$T(N) \le 2T(N/2) + O(N) \Rightarrow T(N) = O(N \log N)$$



# Nearest Neighbor



## 1854 Cholera Outbreak, Golden Square, London

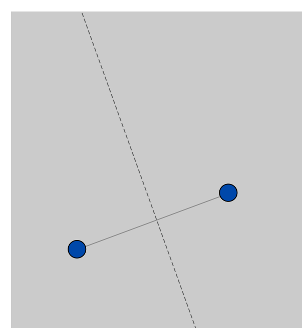

Reference: http://content.answers.com/main/content/wp/en/c/c7/Snow-cholera-map.jpg



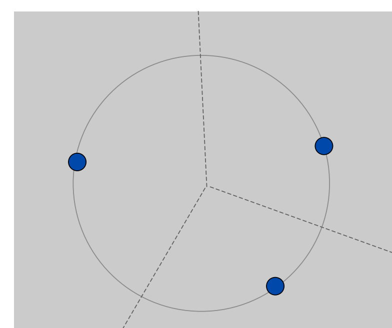
## Nearest Neighbor

Input. N Euclidean points.

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



Voronoi of 2 points
(perpendicular bisector)



Voronoi of 3 points
(passes through circumcenter)

## Nearest Neighbor

Input. N Euclidean points.

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

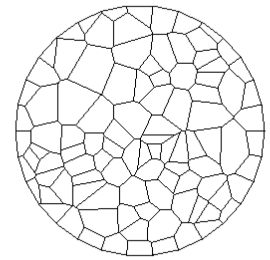
Brute force. O(N) time per query.

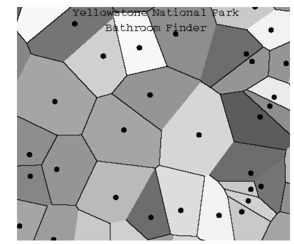
Goal. O(N log N) preprocessing, O(log N) per query.



## Voronoi Diagram / Dirichlet Tesselation

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.



## Applications of Voronoi Diagrams

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



## Applications

Crystallography. N crystal seeds grow at a uniform rate. What is appearance of crystal upon termination of growth?

Facility location. 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?

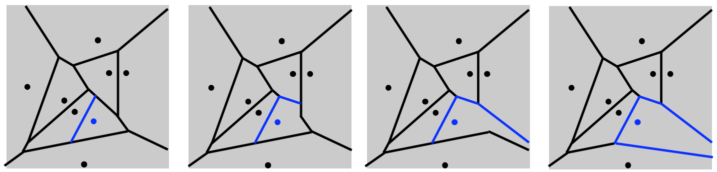
↖ robot should stay on Voronoi diagram of obstacles

Reference: J. O'Rourke. Computational Geometry.

## 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.



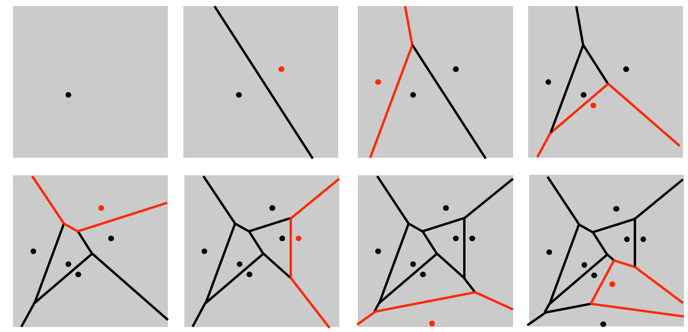
How to represent the Voronoi diagram? Use multilist associating each point with its Voronoi neighbors.



## Randomized Incremental Voronoi Algorithm

Add points (in random order).

- Find region containing point. ← use Voronoi itself
- Update neighbor regions, create region for new point.



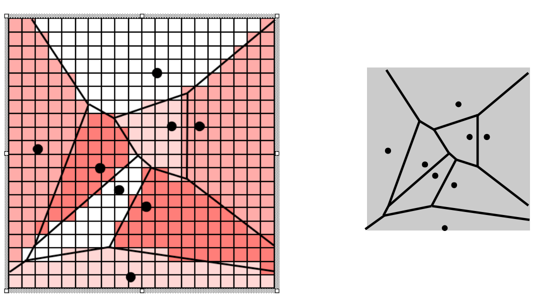
- Running time: O(N log N) on average.



## Discretized Voronoi Diagram

Discretized Voronoi.

- Approach 1: provide approximate answer (to within grid size).
- Approach 2: keep list of points to check in grid squares.
- Computation not difficult (move outward from points).





## Discretized Voronoi: Java Implementation

InteractiveDraw. Version of `StdDraw` that supports user interaction.
DrawListener. Interface to support `InteractiveDraw` callbacks.

```
public class Voronoi implements DrawListener {
   private int SIZE = 512;
   private Point[][] nearest = new Point[SIZE][SIZE];
   private InteractiveDraw draw;
   public Voronoi() {
      draw = new InteractiveDraw(SIZE, SIZE);
      draw.setScale(0, 0, SIZE, SIZE);
      draw.addListener(this);
      draw.show();
   }                               ↖ send callbacks to Voronoi

   public void keyTyped(char c) { }
   public void mouseDragged (double x, double y) { }
   public void mouseReleased(double x, double y) { }
```

## Discretized Voronoi: Java Implementation

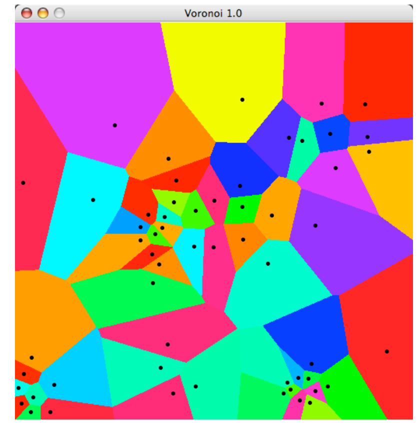
```
public void mousePressed(double x, double y) {
   Point p = new Point(x, y);
   draw.setColorRandom();
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         Point q = new Point(i, j);
         if ((nearest[i][j] == null) ||
             (q.distanceTo(p) < q.distanceTo(nearest[i][j]))) {
            nearest[i][j] = p;
            draw.moveTo(i, j);
            draw.spot();
         }
      }
   }
   draw.setColor(StdDraw.BLACK);
   draw.moveTo(x, y);
   draw.spot(4);
   draw.show();
}
```

user clicks (x, y)

check every other point q to see if p became its nearest neighbor



## Voronoi Diagram

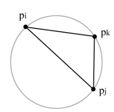




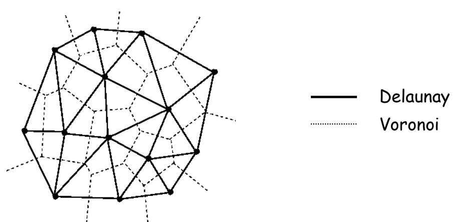

## Delaunay Triangulation

Input: N Euclidean points.

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




Fact 1. Dual of Voronoi (connect adjacent points in Voronoi diagram).
Fact 2. No edges cross (planar) ⇒ O(N) edges.
Fact 3. Maximizes the minimum angle for all triangular elements.
Fact 4. Boundary of Delaunay triangulation is convex hull.
Fact 5. Closest pair in Delaunay graph is closest pair.






## Some Geometric Algorithms

Asymptotic time to solve a 2D problem with N points

| Problem | Brute Force | Cleverness |
|---|---|---|
| convex hull | $N^2$ | N log N |
| closest pair | $N^2$ | N log N |
| nearest neighbor | N | log N |
| polygon triangulation | $N^2$ | N log N |
| furthest pair | $N^2$ | N log N |