**Geometric Primitives**

- **Point**: two numbers \((x, y)\).
- **Line**: two numbers \(a\) and \(b\) \([ax + by = 1]\) \(\quad \rightarrow \) 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 \(p_1p_2p_3\) a counterclockwise turn?

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

**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).


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


Application. Draw a colored polygon on the screen.

**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?**

Warning: Intuition May Mislead

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

Is a given polygon simple?

### Implementing CCW

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

#### Lesson
- Geometric primitives are tricky to implement.
- Dealing with degenerate cases.
- Coping with floating point precision.

Determinant gives twice area of triangle.

\[
2 \times \text{Area}(a, b, c) = \begin{vmatrix}
\begin{array}{ccc}
  a_x & a_y & 1 \\
  b_x & b_y & 1 \\
  c_x & c_y & 1
\end{array}
\end{vmatrix} = (b_y - a_y)(c_z - a_z) - (b_z - a_z)(c_y - a_y)
\]

- 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.
Convex Hull

A set of points is convex if for any two points p and q, the line segment \(pq\) is completely in the set.

**Convex hull.** Smallest convex set containing the points.

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

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

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.

**Implementation.**
- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- 2D analog of selection sort: \(O(MN)\) 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.
Convex Hull Algorithms Costs Summary

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package wrap</td>
<td>N M</td>
</tr>
<tr>
<td>Graham scan</td>
<td>N log N</td>
</tr>
<tr>
<td>Quickhull</td>
<td>N log N</td>
</tr>
<tr>
<td>Mergehull</td>
<td>N log N</td>
</tr>
<tr>
<td>Sweep line</td>
<td>N log N</td>
</tr>
<tr>
<td>Quick elimination</td>
<td>N*</td>
</tr>
<tr>
<td>Best in theory</td>
<td>N log M</td>
</tr>
</tbody>
</table>

* assumes "reasonable" point distribution

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.

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)
  \[ \text{less}(a, b) = (a_x < b_x) \lor ((a_x = b_x) \land (a_y < b_y)) \]
- Quadratic decision tree model: compute any quadratic function of
  the coordinates and compare against 0.
  \[ \text{ccw}(a, b, c) = a_xb_y - a_yb_x + a_yc_x - a_xc_y + b_yc_x - c_yb_x \]

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

**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

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

Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

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

**Claim.** If |i - j| ≥ 12, then the distance between \( s_i \) and \( s_j \) is at least δ.

**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 ≥ 2(\( \frac{1}{2} \delta \)).

**Fact.** Still true if we replace 12 with 7.

**Closest Pair Algorithm**

```c
Closest-Pair(p_1, ..., p_n) {
    Compute separation line L such that half the points are on one side and half on the other side.
    \( \delta_1 = \text{Closest-Pair(left half)} \)
    \( \delta_2 = \text{Closest-Pair(right half)} \)
    \( \delta = \min(\delta_1, \delta_2) \)
    Delete all points further than \( \delta \) from separation line L.
    Sort remaining points by y-coordinate.
    Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
    return \( \delta \).
}
```

O(N log N) 2T(N / 2) O(N)
O(N log N) O(N)
O(N)
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)

1854 Cholera Outbreak, Golden Square, London

**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.

---

**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-Seitz 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:** [voronoi.com](http://voronoi.com), [http://www.ics.uci.edu/~eppstein/geom.html](http://www.ics.uci.edu/~eppstein/geom.html)

---

**Voronoi Diagram / Dirichlet Tessellation**

**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**

**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?

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

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.

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).

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.

**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) { }
```
**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.

```java
public void mousePressed(double x, double y) {
    Point p = new Point(x, y);  // user clicks (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.setColor();
            }
        }
    }
    draw.setColor(StdDraw.BLACK);
    draw.moveTo(x, y);
    draw.setColor();
    draw.show();
}
```

**Asymptotic time to solve a 2D problem with N points**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Brute Force</th>
<th>Cleverness</th>
</tr>
</thead>
<tbody>
<tr>
<td>convex hull</td>
<td>N^2</td>
<td>N log N</td>
</tr>
<tr>
<td>closest pair</td>
<td>N^2</td>
<td>N log N</td>
</tr>
<tr>
<td>nearest neighbor</td>
<td>N</td>
<td>log N</td>
</tr>
<tr>
<td>polygon triangulation</td>
<td>N^2</td>
<td>N log N</td>
</tr>
<tr>
<td>furthest pair</td>
<td>N^2</td>
<td>N log N</td>
</tr>
</tbody>
</table>