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).


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 \[ax + by = 1\]
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 p_1, p_2, p_3, is p_1-p_2-p_3 a counterclockwise turn?

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

Intuition

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

Is a given polygon simple?

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

Is a given polygon simple?
Polygon Inside, Outside

**Jordan curve theorem.** [Veblen 1905] Any continuous simple closed curve cuts the plane in exactly two pieces: the inside and the outside.

**Is a point inside a simple polygon?**

![Image](https://www.cs.uiuc.edu/~appels/gemns.html)

**Application.** Draw a filled polygon on the screen.

---

Does line segment intersect ray?

**CCW.** Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Analog of comparisons in sorting.
- Idea: compare slopes.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>???</td>
</tr>
<tr>
<td>(slope)</td>
<td>(collinear)</td>
<td>(collinear)</td>
</tr>
</tbody>
</table>

**Lesson.** Geometric primitives are tricky to implement.

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

---

Polygon Inside, Outside: Crossing Number

**Determinant gives twice area of triangle.**

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

- If \(\text{area} > 0\) then a-b-c is counterclockwise.
  - If \(\text{area} < 0\), then a-b-c is clockwise.
  - If \(\text{area} = 0\), then a-b-c are collinear.

---

**Implementing CCW**

```java
public boolean contains(double x0, double y0) {
    int crossings = 0;
    for (int i = 0; i < N; i++) {
        if (yi < y0) {
            crossings += 1;
        } else if (yi > y0) {
            crossings -= 1;
        }
    }
    return (crossings % 2 != 0); // Check for intersection
}
```
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.

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

**Properties.**

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

---

**Immutable Point ADT**

```java
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) {
        return Math.hypot(this.x - q.x, this.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;
    }
}
```
**Brute Force**

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

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

\[ O(N^3) \text{ algorithm.} \] For all points \( p \) and \( q \) in \( P \), check whether \( pq \) is an edge of convex hull.

\[ \text{each check requires } O(N) \text{ ccw calculations, where } N \text{ is the number of points in } P \]

**Package Wrap (Jarvis March)**

Implementation.
- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- \( \Theta(N) \) per iteration.

Parameters.
- \( N \) = number of points.
- \( h \) = number of points on the hull.

Package wrap running time. \( \Theta(Nh) \) per iteration.

How many points on the hull?
- Worst case: \( h = N \).
- Average case: difficult problems in stochastic geometry.
  - in a disc: \( h = \sqrt[3]{N} \).
  - in a convex polygon with \( O(1) \) edges: \( h = \log N \).
**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.

```
// 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-1], p[M], p[i]) <= 0) { //
        M--;
    }
    swap(points, M, i);
} // discard points that would create clockwise turn
M++;
```

**Running time.** $O(N \log N)$ for sort and $O(N)$ for rest.

**Convex Hull Algorithms Costs Summary**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package wrap</td>
<td>$Nh$</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>$N \log h$</td>
</tr>
</tbody>
</table>

*output sensitive running time

asymptotic cost to find $h$-point hull in $N$-point set

1 assumes “reasonable” point distribution
Convex Hull: Lower Bound

Models of computation.
- **Comparison based:** compare coordinates.
  (impossible to compute convex hull in this model of computation)
  \[
  (a \prec b) \lor ((a = b) \land (a \prec c) \land (c \prec b))
  \]
- **Quadratic decision tree model:** compute any quadratic function of the coordinates and compare against 0.
  \[
  (a \cdot b \cdot y - a \cdot y \cdot b + a \cdot y \cdot c + b \cdot a \cdot c \cdot y - c \cdot a \cdot b) < 0
  \]

**Theorem.** [Andy Yao, 1981] In quadratic decision tree model, any convex hull algorithm requires \(\Omega(N \log N)\) ops. 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.
  fast closest pair inspired fast algorithms for these problems

**Brute force.** Check all pairs of points \(p\) and \(q\) with \(\Theta(N^2)\) distance calculations.

**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.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side.
- Return best of 3 solutions.
**Closest Pair Algorithm**

Closest-Pair($p_1, \ldots, p_n$) {
    Compute separation line $L$ such that half the points are on one side and half on the other side.
    $\delta_1 =$ Closest-Pair(left half)
    $\delta_2 =$ 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 $\delta$.
}

**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!

![Diagram showing closest pair of points](image)

$\delta = \min(12, 21)$

**Closest Pair of Points: Analysis**

**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.

**Running time.**

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

**Upper bound.** Can be improved to $O(N \log N)$.

**Lower bound.** In quadratic decision tree model, any algorithm for closest pair requires $\Omega(N \log N)$ steps.
Nearest Neighbor

**Input.** N Euclidean points.

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocess</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>Goal</td>
<td>N log N</td>
<td>log N</td>
</tr>
</tbody>
</table>

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 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: 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-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.


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?

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


Scientific Rediscoveries

<table>
<thead>
<tr>
<th>Year</th>
<th>Discoverer</th>
<th>Discipline</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1644</td>
<td>Descartes</td>
<td>Astronomy</td>
<td>&quot;Heavens&quot;</td>
</tr>
<tr>
<td>1850</td>
<td>Dirichlet</td>
<td>Math</td>
<td>Dirichlet tesselation</td>
</tr>
<tr>
<td>1908</td>
<td>Voroni</td>
<td>Math</td>
<td>Voroni diagram</td>
</tr>
<tr>
<td>1909</td>
<td>Boldyrev</td>
<td>Geology</td>
<td>area of influence polygons</td>
</tr>
<tr>
<td>1911</td>
<td>Thiessen</td>
<td>Meteorology</td>
<td>Thiessen polygons</td>
</tr>
<tr>
<td>1927</td>
<td>Niggi</td>
<td>Crystallography</td>
<td>domains of action</td>
</tr>
<tr>
<td>1933</td>
<td>Wigner-Seitz</td>
<td>Physics</td>
<td>Wigner-Seitz regions</td>
</tr>
<tr>
<td>1958</td>
<td>Frank-Casper</td>
<td>Physics</td>
<td>atom domains</td>
</tr>
<tr>
<td>1965</td>
<td>Brown</td>
<td>Ecology</td>
<td>area of potentially available</td>
</tr>
<tr>
<td>1966</td>
<td>Mead</td>
<td>Ecology</td>
<td>plant polygons</td>
</tr>
<tr>
<td>1985</td>
<td>Hoofd et al</td>
<td>Anatomy</td>
<td>capillary domains</td>
</tr>
</tbody>
</table>

Reference: Kenneth E. Hoff III
**Fortune's Algorithm**

Sweep-line algorithm can be implemented in $O(N \log N)$ time.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocess</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute</td>
<td>1</td>
<td>$N$</td>
</tr>
<tr>
<td>Fortune</td>
<td>$N \log N$</td>
<td>$\log N$</td>
</tr>
</tbody>
</table>

Hoff's algorithm. Align apex of a right circular cone with sites.
- Minimum envelope of cone intersections projected onto plane is the Voronoi diagram.
- View cones in different colors $\Rightarrow$ render Voronoi.

**Discretized Voronoi**

Solve nearest neighbor problem on an $N$-by-$N$ grid.

**Brute force.** For each grid cell, maintain closest point. When adding a new point to Voronoi, update $N^2$ cells.

**Hoff's Algorithm**

Draw cones using standard graphics hardware!

**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 diagram).
**Fact 2.** No edges cross → $O(N)$ edges.
**Fact 3.** Maximizes the minimum angle for all triangular elements.
**Fact 4.** Boundary of Delaunay triangulation is convex hull.
**Fact 5.** Shortest Delaunay edge connects closest pair of points.

---

**Summary.** Many fundamental geometric problems require ingenuity to solve large instances.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Brute</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>furthest pair</td>
<td>$N^2$</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Delaunay triangulation</td>
<td>$N^4$</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>polygon triangulation</td>
<td>$N^2$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

Asymptotic time to solve a 2D problem with $N$ points.