Geometric Algorithms

- primitive operations
- convex hull
- closest pair
- voronoi diagram

References:
Algorithms in C (2nd edition), Chapters 24-25
http://www.cs.princeton.edu/introalgsds/71primitives
http://www.cs.princeton.edu/introalgsds/72hull
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.
primitive operations
- convex hull
- closest pair
- voronoi diagram
**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?

we think of this
algorithm sees this

no crossings
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?**

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

http://www.ics.uci.edu/~eppstein/geom.html
public boolean contains(double x0, double y0)
{
    int crossings = 0;
    for (int i = 0; i < N; i++)
    {
        double slope = (y[i+1] - y[i]) / (x[i+1] - x[i]);
        boolean cond1 = (x[i] <= x0) && (x0 < x[i+1]);
        boolean cond2 = (x[i+1] <= x0) && (x0 < x[i]);
        boolean above = (y0 < slope * (x0 - x[i]) + y[i]);
        if ((cond1 || cond2) && above) crossings++;
    }
    return (crossings % 2 != 0);
}
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.

![Diagram of CCW](image)

**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.

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

- 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.
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 else (area2 < 0) return -1;      else if (area2 > 0) return +1;      else if (area2 > 0  return  0;   }

    public static boolean collinear(Point a, Point b, Point c)
    {      return ccw(a, b, c) == 0;   }  
}
Sample ccw client: Line intersection

Intersect: Given two line segments, do they intersect?
• Idea 1: find intersection point using algebra and check.
• Idea 2: check if the endpoints of one line segment are on different "sides" of the other line segment.
• 4 ccw computations.

```java
public static boolean intersect(Line l1, Line l2) {
    int test1, test2;
    test1 = Point.ccw(l1.p1, l1.p2, l2.p1) * Point.ccw(l1.p1, l1.p2, l2.p2);
    test2 = Point.ccw(l2.p1, l2.p2, l1.p1) * Point.ccw(l2.p1, l2.p2, l1.p2);
    return (test1 <= 0) && (test2 <= 0);
}
```
primitive operations
convex hull
closest pair
voronoi diagram
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
Brute-force algorithm

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

**Observation 2.**
$p-q$ is on convex hull if all other points are counterclockwise of $\overrightarrow{pq}$.

**$O(N^3)$ algorithm.**
For all pairs of points $p$ and $q$ in $P$
- compute $\text{ccw}(p, q, x)$ for all other $x$ in $P$
- $p-q$ is on hull if all values positive
Package Wrap (Jarvis March)

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 (Jarvis March)

**Implementation.**
- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- $\Theta(N)$ per iteration.
How Many Points on the Hull?

Parameters.
• $N =$ number of points.
• $h =$ number of points on the hull.

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

How many points on hull?
• Worst case: $h = N$.
• Average case: difficult problems in stochastic geometry.
  • in a disc: $h = N^{1/3}$.
  • 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.
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.

```c
// 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--;
    M++;
    swap(points, M, i);
}
```

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

**why?**
Quick Elimination

Quick elimination.
- Choose a quadrilateral Q or rectangle R with 4 points as corners.
- Any point inside cannot be on hull
  - 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.

In practice
- can eliminate almost all points in linear time.
**Convex Hull Algorithms Costs Summary**

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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>growth of running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package wrap</td>
<td>$N \times h$</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>$N^\dagger$</td>
</tr>
<tr>
<td>Best in theory</td>
<td>$N \log h$</td>
</tr>
</tbody>
</table>

$^\dagger$ assumes "reasonable" point distribution

(output sensitive)
Convex Hull: Lower Bound

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

\[(a.x < b.x) \text{ || } ((a.x == b.x) \&\& (a.y < b.y))\]

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

\[(a.x*b.y - a.y*b.x + a.y*c.x - a.x*c.y + b.x*c.y - c.x*b.y) < 0\]

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

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

even if hull points are not required to be output in counterclockwise order
primitive operations
convex hull
closest pair
voronoi diagram
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 $\Theta(N^2)$ distance calculations.

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

Degeneracies complicate solutions.
  [ assumption for lecture: no two points have same x coordinate]
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.
- Return best of 3 solutions.

seems like $\Theta(N^2)$
Closest Pair of Points

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

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

\[ \delta = \min(12, 21) \]
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.

\[ \delta = \min(12, 21) \]
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!

\( \delta = \min(12, 21) \)
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 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 \( \delta \).
}
Closest Pair of Points: Analysis

Algorithm gives upper bound on running time

Recurrence

\[ T(N) \leq 2T(N/2) + O(N \log N) \]

Solution

\[ T(N) = O(N (\log N)^2) \]

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.
primitive operations
convex hull
closest pair
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?

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

<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 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 an obstacle?

-robot should stay on Voronoi diagram of obstacles

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.

<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>Voronoi</td>
<td>Math</td>
<td>Voronoi 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>Niggli</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
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.

• Running time: $O(N \log N)$ on average.

Not an elementary algorithm
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
- properly handles floating-point computations

<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>

Try it yourself!

http://www.diku.dk/hjemmesider/studerende/duff/Fortune/

Interface between numeric and combinatorial computing
- exact calculations impossible (using floating point)
- exact calculations required!
- one solution: randomly jiggle the points

best animation on the web student Java project “lost” the source decompiled source available
Fortune’s algorithm in action

http://www.diku.dk/hjemmesider/studerende/duff/Fortune/
Fortune’s algorithm in action
Fortune’s algorithm in action
Fortune’s algorithm in action
Fortune’s algorithm in action
Geometric-algorithm challenge

Problem: Draw a Voronoi diagram
Goals: lecture slide, book diagram

How difficult?
1) any COS126 student could do it
2) need to be a typical diligent COS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible
**Geometric-algorithm challenge**

**Problem:** Draw a Voronoi diagram
*Goals:* lecture slide, book diagram

**How difficult?**

- ✓ 1) any COS126 student could do it
- 2) need to be a typical diligent COS226 student
- 3) hire an expert
- 4) intractable
- 5) no one knows
- 6) impossible

*surprise!*
Discretized Voronoi diagram

Observation: to draw a Voronoi diagram, only need an approximation

Ex: Assign a color to each pixel corresponding to its nearest neighbor

An effective approximate solution to the nearest neighbor problem

<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>
<tr>
<td>Discretized</td>
<td>(N) (P)</td>
<td>1</td>
</tr>
</tbody>
</table>

complicated alg (stay tuned)
Discretized Voronoi: Java Implementation

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

```java
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();
    }

    public void keyTyped(char c) {}  
    public void mouseDragged(double x, double y) {}  
    public void mouseReleased(double x, double y) {}  
    public void mousePressed  
    {  /* See next slide  */  }
}
```

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();
}
Voronoi alternative 2: Hoff's algorithm

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.

Implementation. Draw cones using standard graphics hardware!

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 $\Rightarrow O(N)$ 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.
Euclidean MST

Euclidean MST. Given N points in the plane, find MST connecting them.
- Distances between point pairs are Euclidean distances.

Brute force. Compute \( N^2 / 2 \) distances and run Prim’s algorithm.

Ingenuity.
- MST is subgraph of Delauney triangulation
- Delauney has \( O(N) \) edges
- Compute Delauney, then use Prim or Kruskal to get MST in \( O(N \log N) \)!
Ingenuity in algorithm design can enable solution of large instances for numerous fundamental geometric problems.

<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>Voronoi</td>
<td>?</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Delaunay triangulation</td>
<td>$N^4$</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Euclidean MST</td>
<td>$N^2$</td>
<td>$N \log N$</td>
</tr>
</tbody>
</table>

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

Note: 3D and higher dimensions test limits of our ingenuity
## Geometric algorithms summary: Algorithms of the day

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Brute</th>
<th>Ingenuity</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>Voronoi/Delauney</td>
<td>$N^4$</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Euclidean MST</td>
<td>$N^2$</td>
<td>$N \log N$</td>
</tr>
</tbody>
</table>