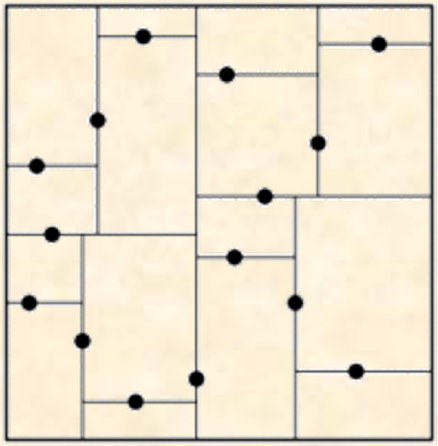
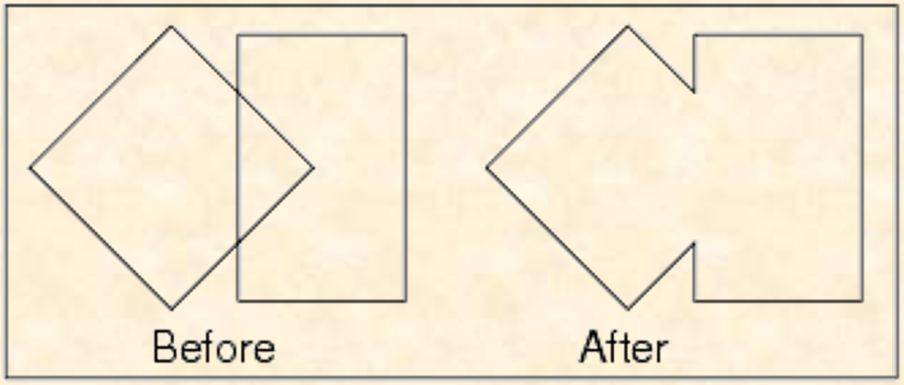
Lecture 25
Computational Geometry





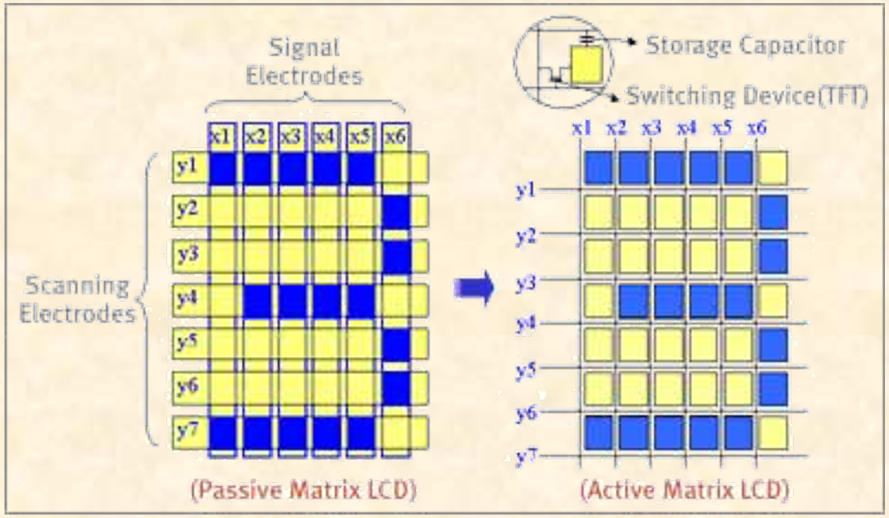
EECS 281: Data Structures & Algorithms

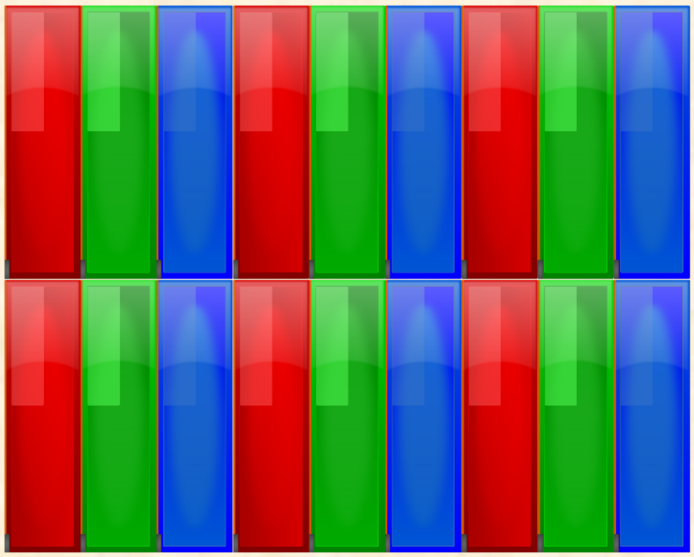
Computational Geometry Overview

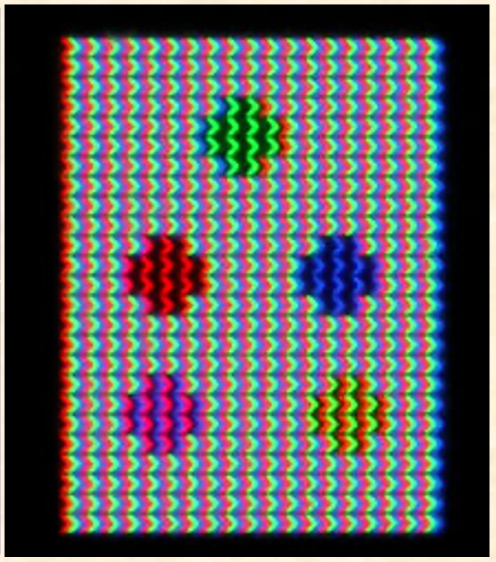
Data Structures & Algorithms

Raster (Bitmap) Graphics

Images on screen are represented by pixel arrays



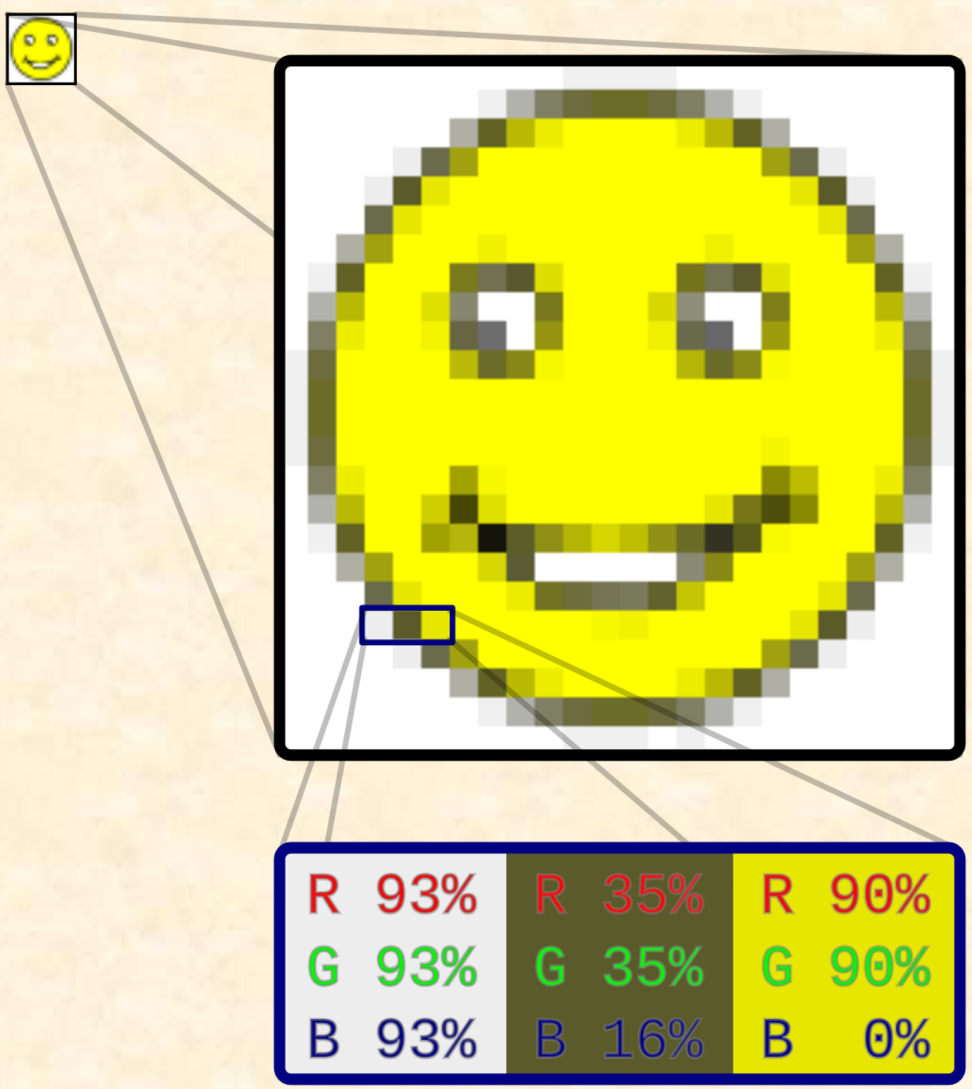




Same idea works for storing images in files

- RGB: 3 colors per pixel, 1 byte per color
- An 1920×1080 image would require ~6.22MB
- Much larger files for high-res camera images

Vector Graphics Scale Better



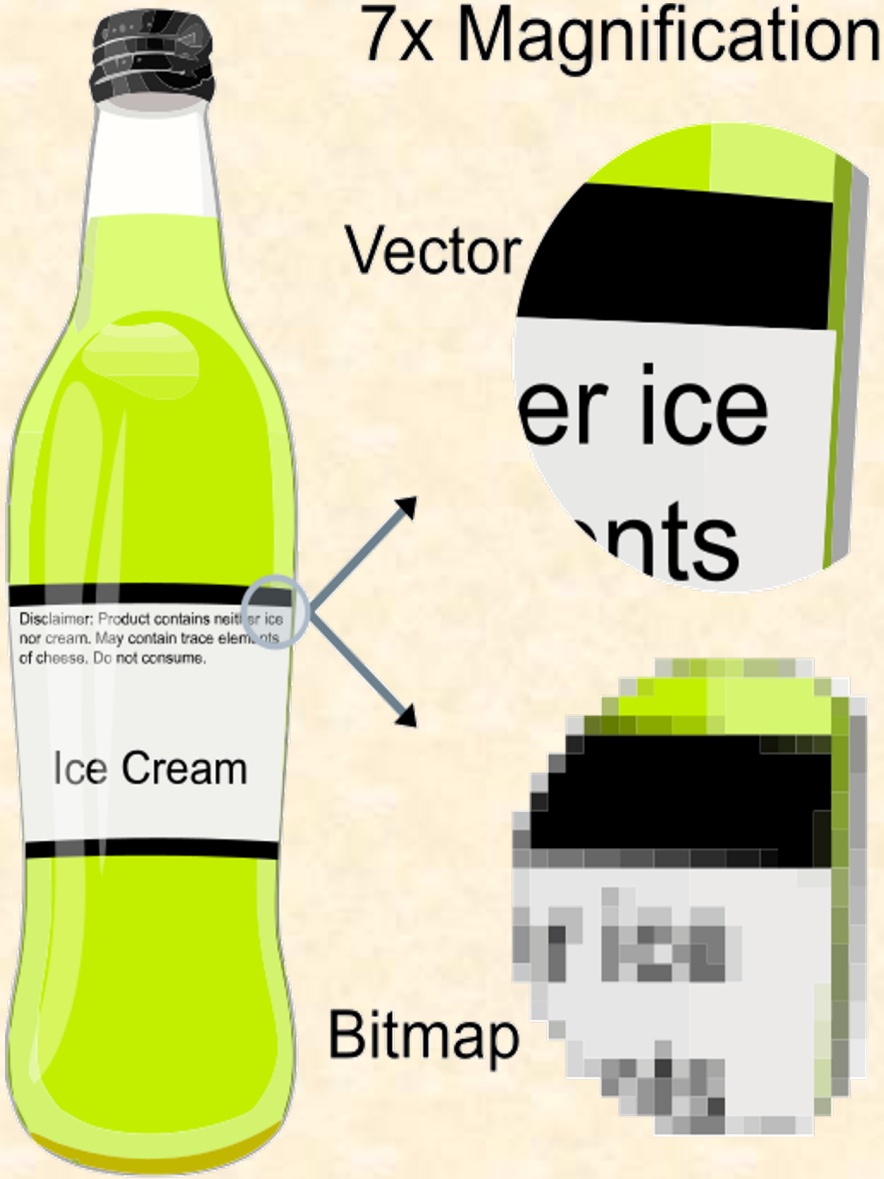


Image Compression

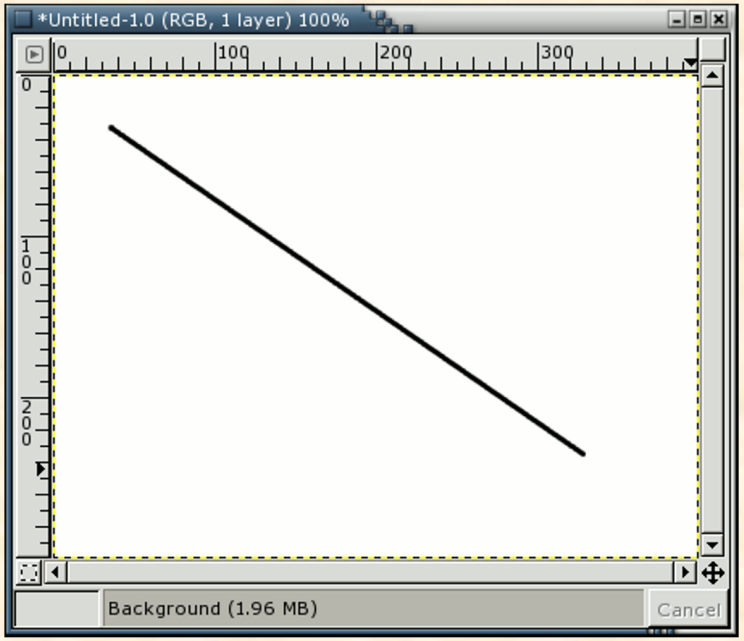
- Lossy vs. lossless compression

Example of Lossy Compression

Original Lena Image (12KB size)

Lena Image, Compressed (85% less information, 1.8KB)

Lena Image, Highly Compressed (96% less information, 0.56KB)



- Flat background and sparse line graphics compress well without losses
- Vector graphics compress even better

Basic Geometry Objects

Points, segments and lines in 2D and 3D

2D point: (x,y)

2D segment: two distinct points: $\{(x_0, y_0), (x_1, y_1)\}$

2D line: slope + intercept: (k, c) in $y = k\,x + c$

A segment defines a line

- Given two unequal points, solve for k, c

$$y_0 = kx_0 + c \qquad k = (y_0 - y_1)/(x_0 - x_1)$$
$$y_1 = kx_1 + c \qquad c = y_0 - kx_0$$

- Different segments may define the same line
- Must use floats or doubles, not ints

Geometry Transforms

Idea: modify all points & segments the same way

- Shifting/displacing point (x, y) by a vector (dx, dy)

  $(x, y) \rightarrow (x + dx, y + dy)$

- Scaling/stretching in the x dimension by $C > 0$

  $(x, y) \rightarrow (C\,x, y)$

- Mirroring in the x dimension

  $(x, y) \rightarrow (-x, y)$

- Rotation by angle θ

"Rotation Matrix"

$$(x, y) \rightarrow \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

Basic Geometry Questions

- Given two lines
  - If they intersect, find their intersection point
- Given three points
  - Do they lie on the same line? Find their triangle area?
  - Are they ordered clockwise or counter-clockwise?
- Given point *P* and line *L*
  - Does *P* lie on L? Find the point on *L* closest to *P*
- Given *N* points (1D, 2D, 3D)
  - Find the smallest rectilinear bounding box of *N* points
  - Find a closest pair of points
  - Find a closest point among *N* to a given point *A*

Basic Geometry Questions

Two very different kinds of question

1. Deal with several points, lines, segments at a time w/o considering algorithmic complexity
   - Often solved by simple formulas or simple logic

2. Deal with collections of *N* points, lines, segments;
   need efficient algorithms in terms of *N*
   - Often use simple formulas many times

Floating-Point Comparisons ==

General Rule 1: When comparing floating-point values for equality, use approximate comparisons.

Instead of using

```
if (a == b)
```

use

```
if (fabs(a - b) < epsilon)
```

for a suitable epsilon, for example

```
constexpr float epsilon = 0.0001;
```

Floating-Point Comparisons <,>

General Rule 2: When comparing floating-point values for inequality, use approximate comparisons.

Instead of using

```
if (a < b)
```

use

```
if (a < b + epsilon)
if (a < b - epsilon)
```

Computational Geometry Overview

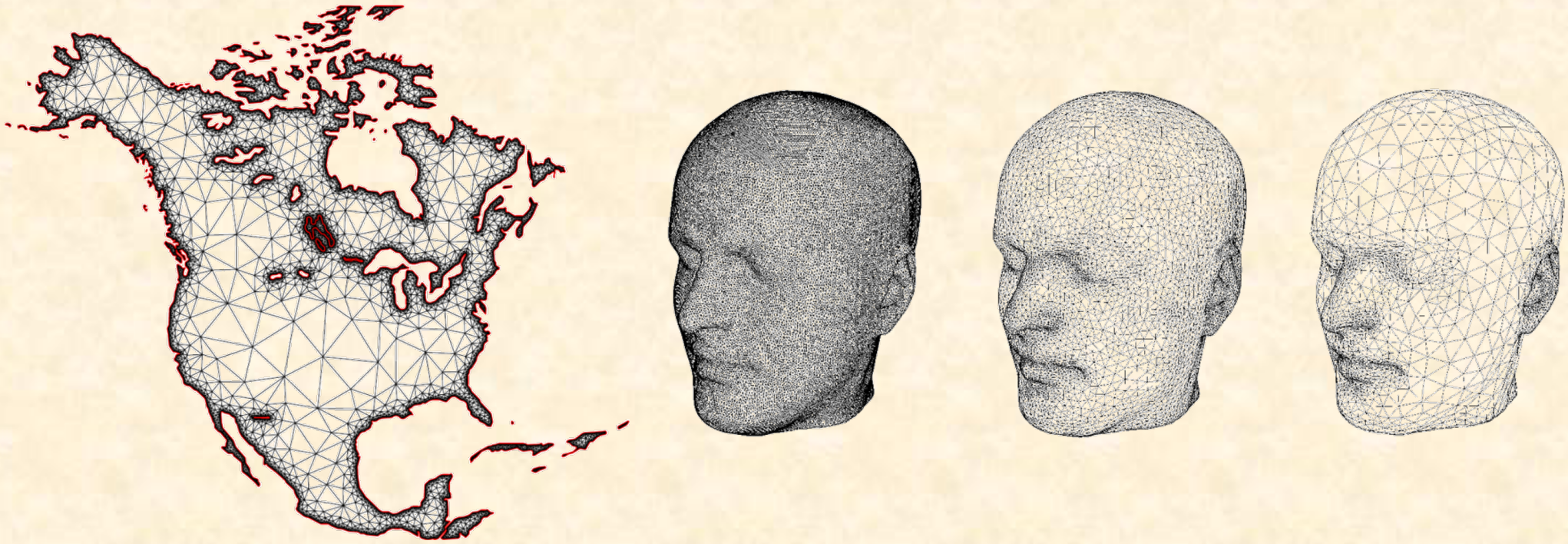
Data Structures & Algorithms

Composite Geometry Objects

Pointsets and polygons

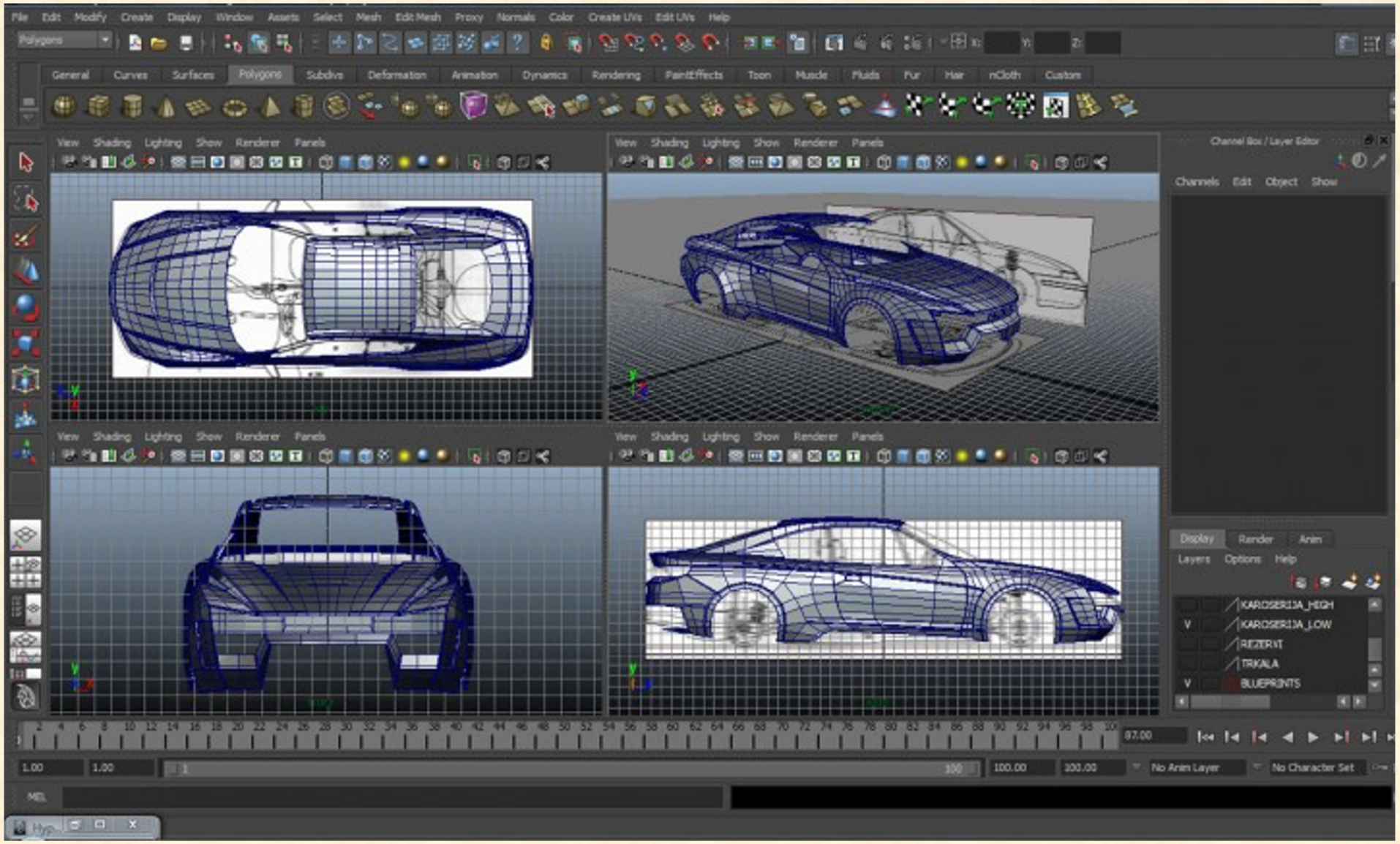
Composite Geometry Objects

2D and 3D meshes



Stored in files: points and segments

Application: Solid Modeling CAD



Points and Lines

Data Structures & Algorithms

Intersection of Two Lines (1)

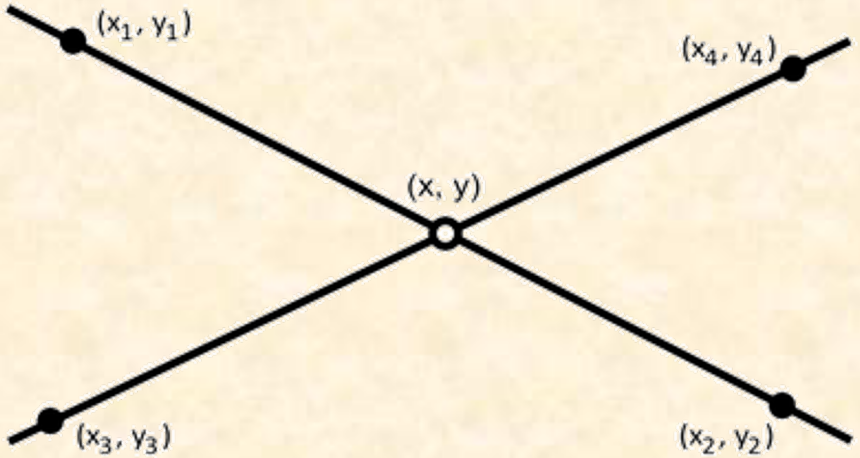
Two lines given by their slope-intercept equations

$y = ax + b$ and $y = cx + d$

- Check if the slopes are equal ($a == c$)
  - case: $b == d$ (same line)
  - case: $b != d$ (parallel lines, no intersection)
- If slopes are unequal ($a != c$), find the single intersection point (x_0, y_0)
  - $y_0 = ax_0 + b = cx_0 + d$
  - $x_0 = (d - b) / (c - a)$
  - $y_0 = ax_0 + b$

Intersection of Two Lines (2)

Two lines given by pairs of points



Same idea as before,
but more involved math

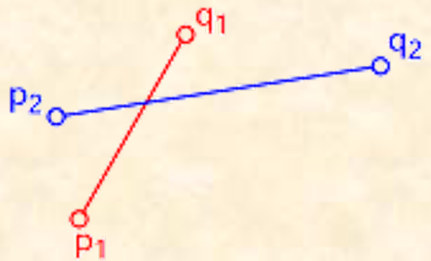
From Wikipedia:

$$(P_x, P_y) = \left(\frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_1 - x_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

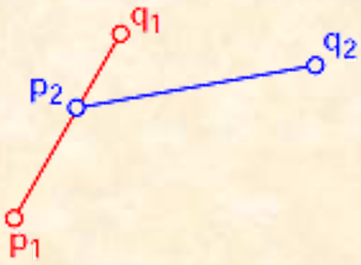
Zero denominators → parallel lines

Segment Intersection Test (1)

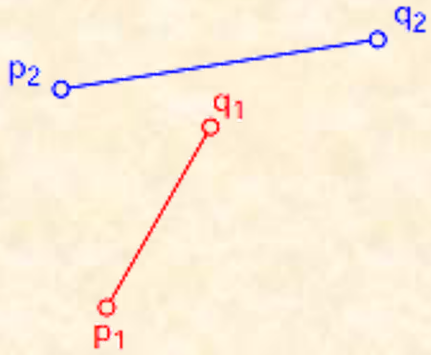
Do two segments intersect?



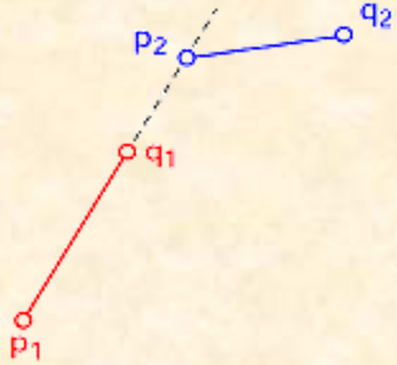
Example 1: Orientations of (p1, q1, p2) and (p1, q1, q2) are different. Orientations of (p2, q2, p1) and (p2, q2, q1) are also different



Example 2: Orientations of (p1, q1, p2) and (p1, q1, q2) are different. Orientations of (p2, q2, p1) and (p2, q2, q1) are also different



Example 3: Orientations of (p1, q1, p2) and (p1, q1, q2) are different. Orientations of (p2, q2, p1) and (p2, q2, q1) are **same**



Example 4: Orientations of (p1, q1, p2) and (p1, q1, q2) are different. Orientations of (p2, q2, p1) and (p2, q2, q1) are **same**

Segment Intersection Test (2)

1. Find two lines formed by the segments
2. If the lines are parallel (same slope)
   - But distinct (intercepts are ≠), return false
   - And coincide (equal intercepts), check if the segments' *X* and *Y* projections overlap
3. Else, find the intersection point
   - Check if it lies inside both segments (by checking this for *X* and *Y* coordinates)

Collinear Points

Q: Do points P_1, P_2, P_3 lie on the same line?

A: Check if the lines formed by segments $a = [P_1, P_2]$ and $b = [P_2, P_3]$ are the same

1. Find slope-intercept $\{k_a, c_a\}$ values from the line segment $[P_1, P_2]$
2. Find slope-intercept $\{k_b, c_b\}$ values from the line segment $[P_2, P_3]$
3. Collinear if $k_a == k_b$ and $c_a == c_b$

Distance from Point to a Line

- Given a point (x_0, y_0), find the shortest distance to a line $ax + by + c = 0$

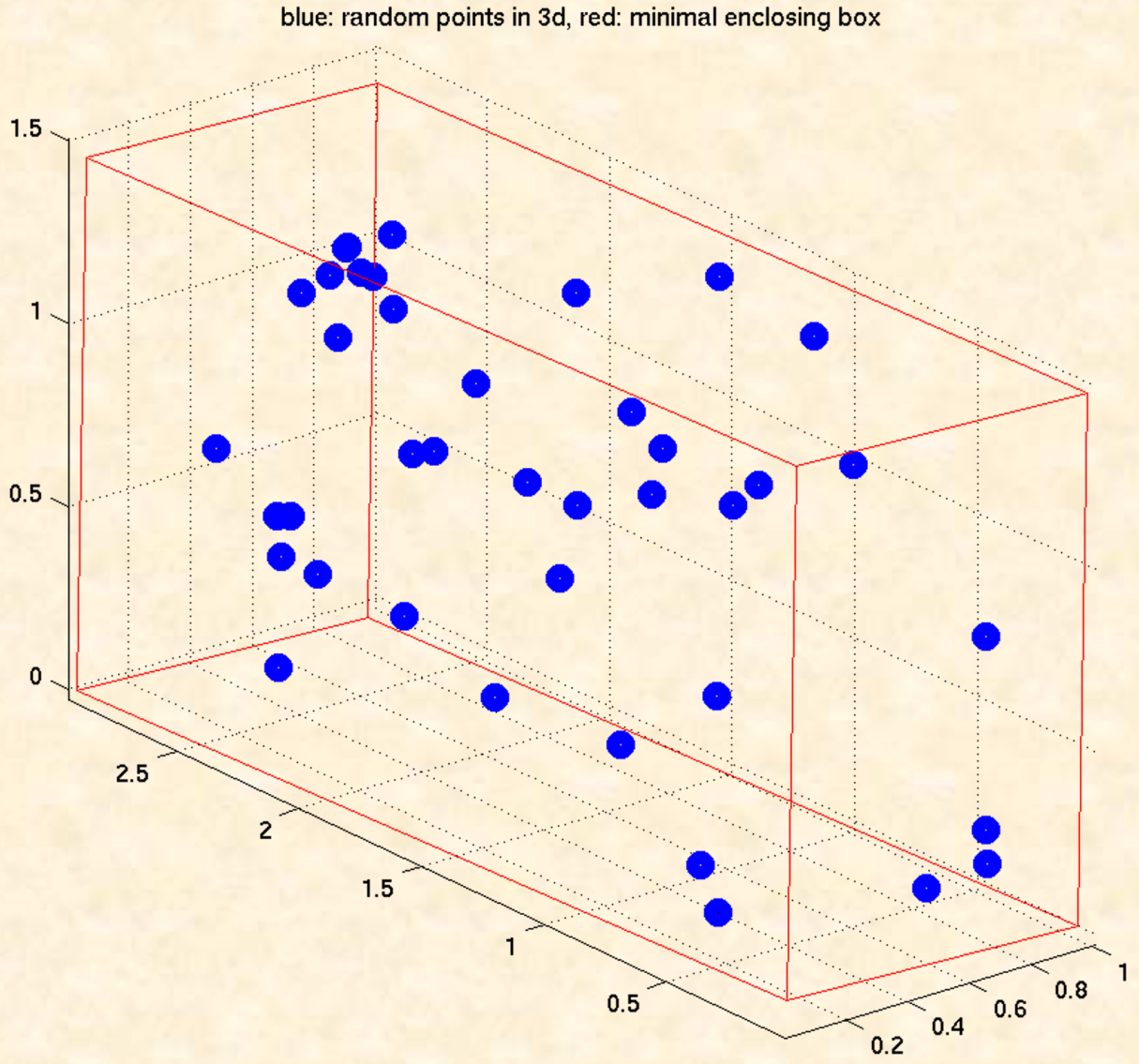
$$\operatorname{distance}(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}.$$

- Find the coordinates of the point on the line closest to (x_0, y_0)

$$x = \frac{b(bx_0 - ay_0) - ac}{a^2 + b^2} \text{ and } y = \frac{a(-bx_0 + ay_0) - bc}{a^2 + b^2}.$$

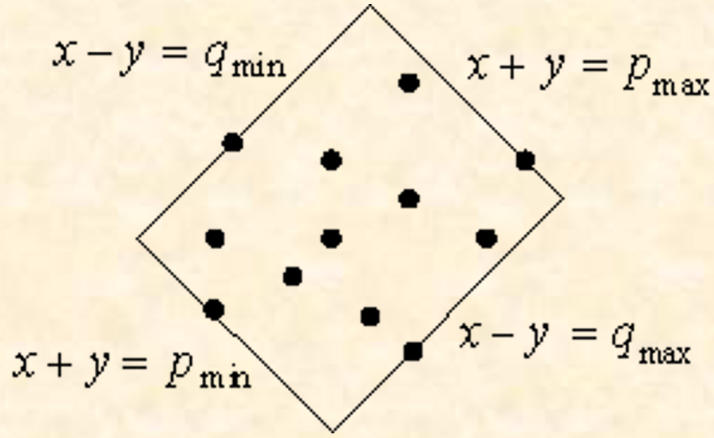
*Source: Wikipedia

Min Bounding Box of a Pointset



Min Bounding Box of a Pointset

- Finding the minimum size BB is easy, IF you stay parallel to *x*/*y* axes
  - For each coordinate, find min and max, ***O*(*N*)**
- Usually a smaller BB exists without edges parallel to *x*/*y* axes



Possible in ***O*(*N*)** time, but non-trivial algorithm

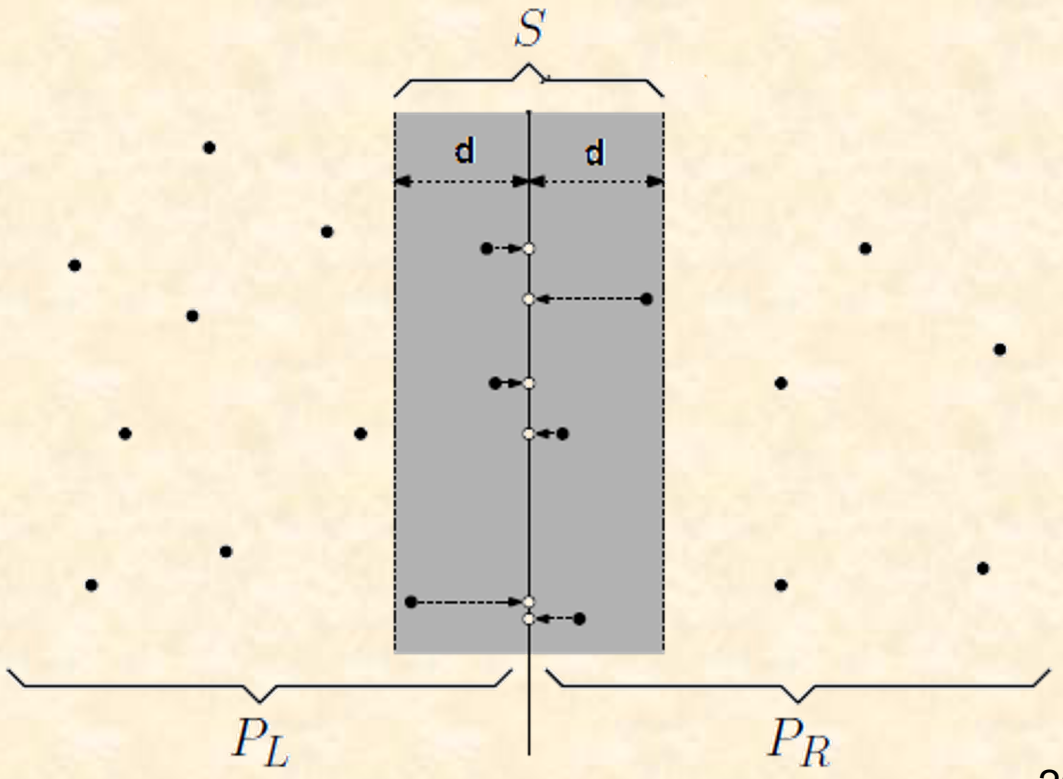
Points and Lines

Data Structures & Algorithms

Closest Pair of Points (1)

- Given *N* points, find two with min distance
  - Direct $O(N^2)$-time algorithm
- In 1D, $O(N \log N)$ time via sorting
- In 2D, use Divide & Conquer:
  - Left side
  - Right side
  - Middle strip

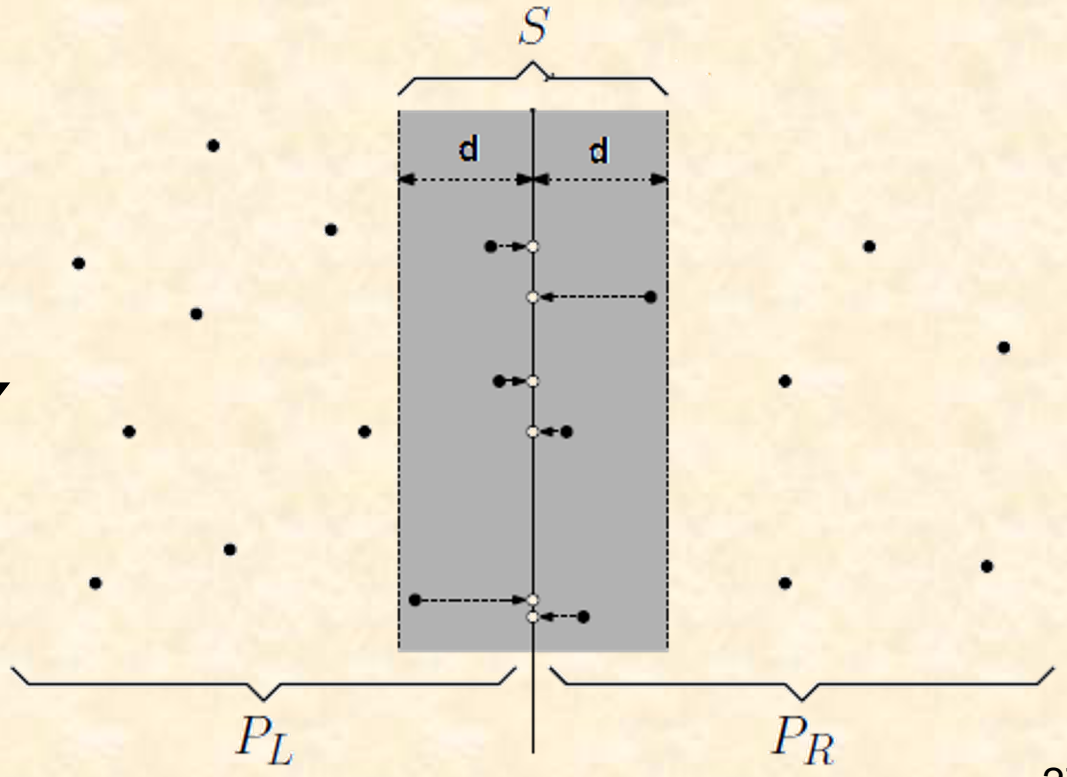
(alternate directions)



Closest Pair of Points (2)

Partitioning can be done using *X*-median

1. Recurse on the left half, find min dist
2. Recurse on the right half, find min dist
3. Additionally form a middle strip of width min-dist
4. Sort the strip by *Y*
5. Compare each pt to 7 points after it



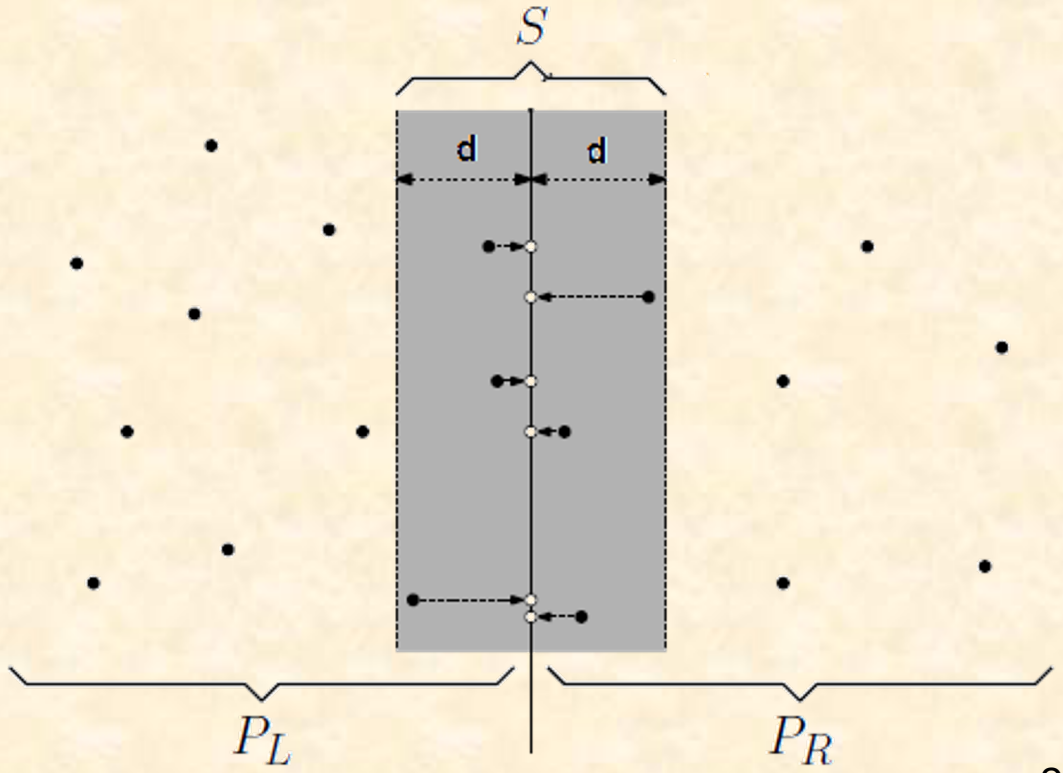
Closest Pair of Points (3)

Recursion is similar to that in Quick sort, but with two linear terms – for partition and for processing the middle strip

***O*(*N* log *N*)** time

If sorting is used to partition, we get

***O*(*N* log^2 *N*)** time



Finding Closest Point from *N*

- Given *N* points, we'd like to serve the following type of queries
  - For point *P*, find a closest point among *N* pts ("the closest" would be misleading in general)
  - To go beyond the obvious $O(N^2)$-time solution, build a data structure (a search index)
- This problem has many applications in databases, geospatial mapping, Internet systems, computer-aided design etc

Polygons

Data Structures & Algorithms

Clockwise Test & Triangle Area

- Given three non-colinear points a, b, c
  - Determine if they turn *counterclockwise* (left) or *clockwise* (right)
  - Find the area of the triangle they form
- Two problems solved by the same formula

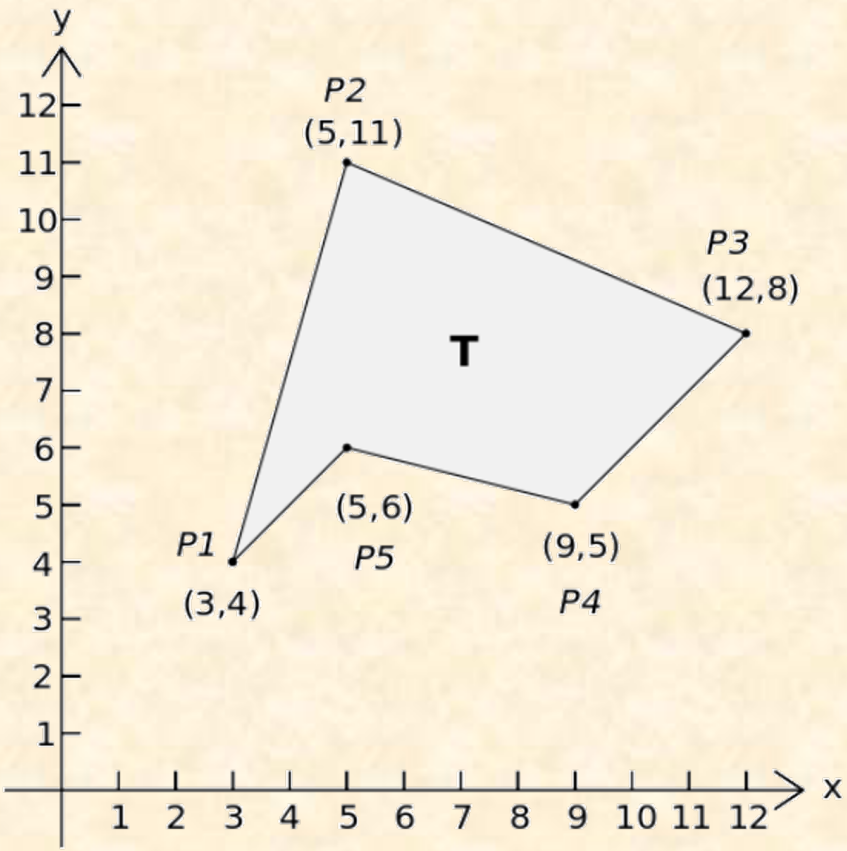
$$\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y$$

$$= (b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)$$

"signed triangle area" is < 0 for "clockwise"

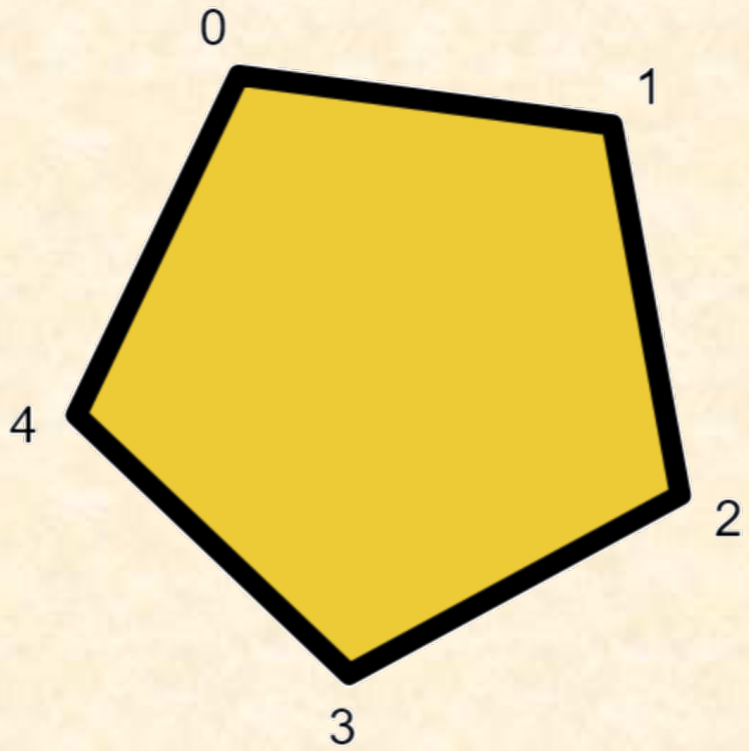
Area of a Polygon

- a polygon is given by n points (x_i, y_i)
- no segments are allowed to intersect

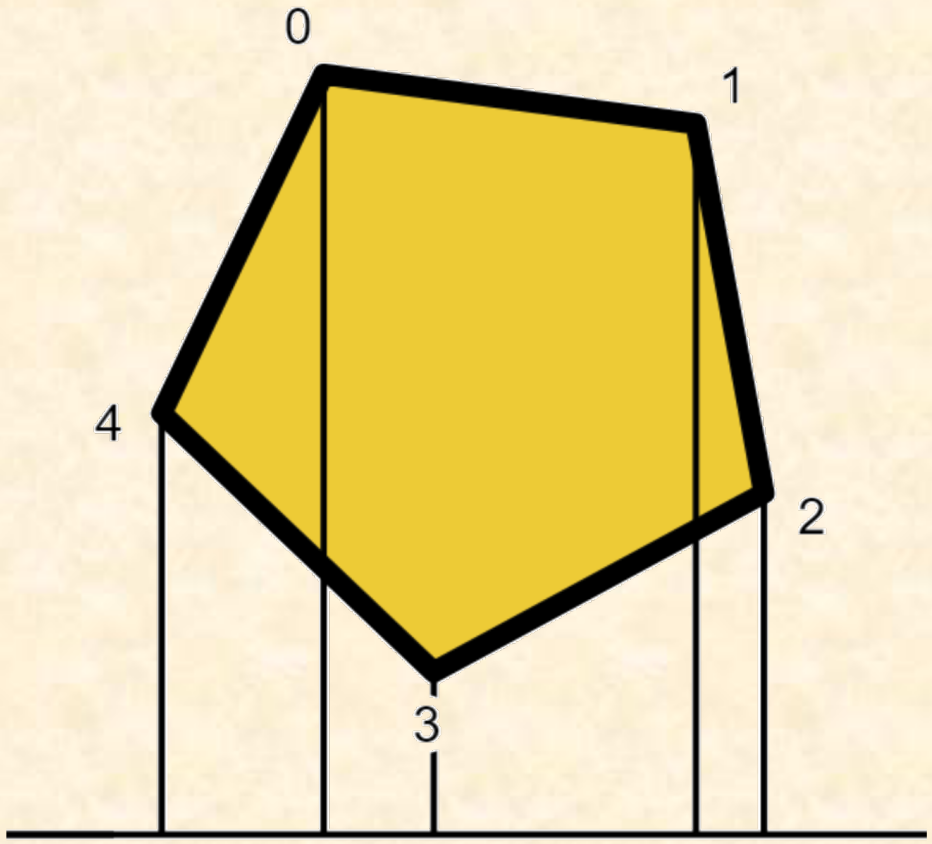


Area of a Polygon

A polygon with 5 points



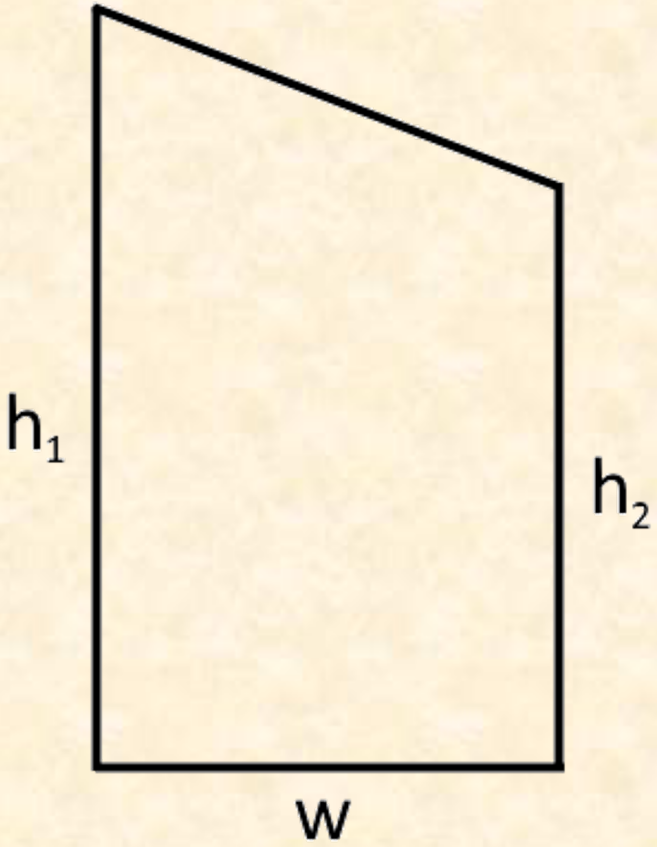
Area of a Polygon



Break the polygon into pieces, by drawing vertical lines from each point down to the x-axis.

We're left with nearly-trapezoidal regions.

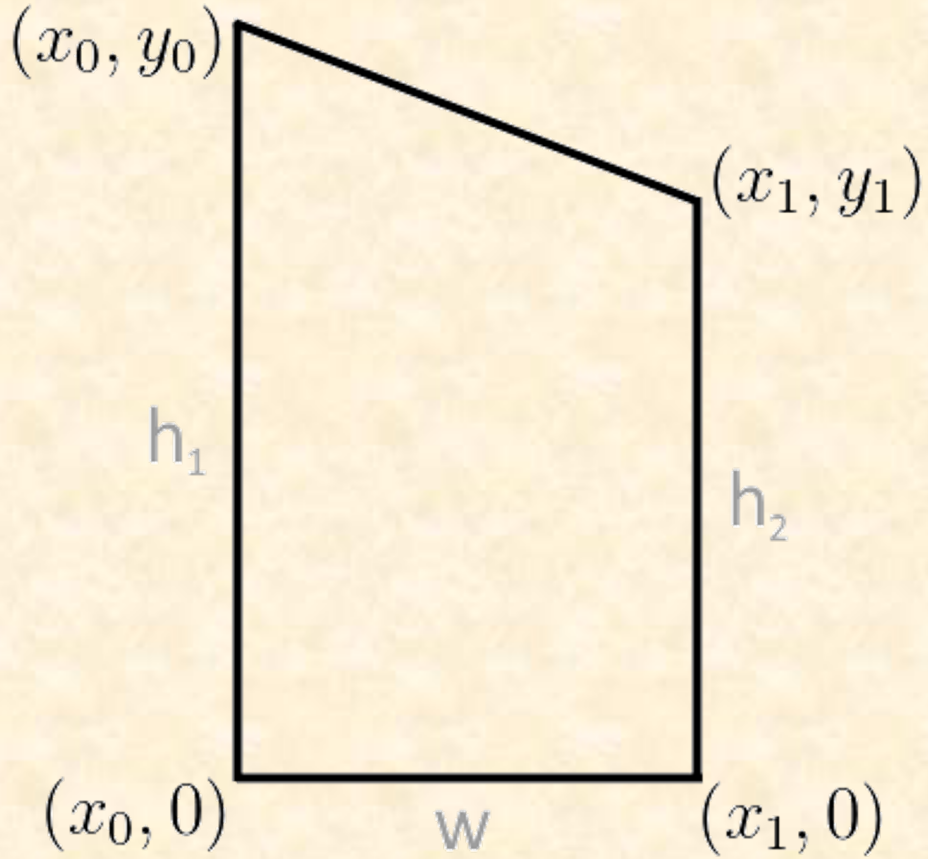
Area of a *Trapezoid*



Area of a trapezoid with right angles:

$$A = \frac{w(h_1 + h_2)}{2}$$

Area of a *Trapezoid*



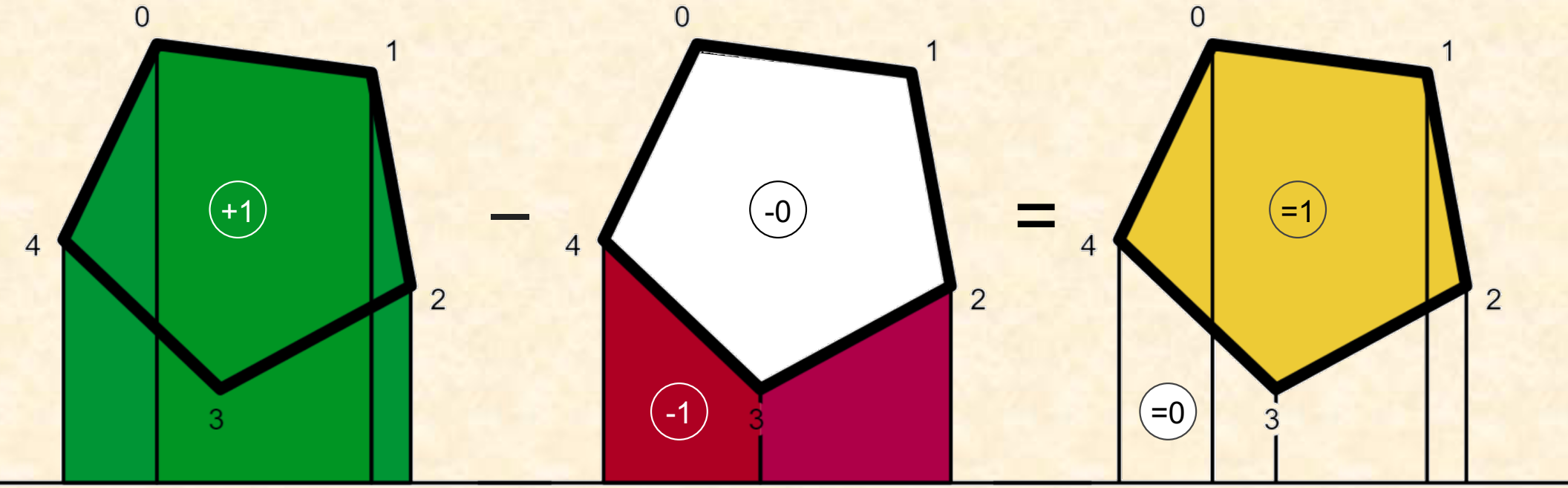
Area of a trapezoid with right angles:

$$A = \frac{(x_1 - x_0)(y_o + y_1)}{2}$$

Area of a Polygon

Remove the red shape from the green shape to get the yellow shape

- Each trapezoid has a top edge in the polygon
- Each edge is the top of a trapezoid (red or green)



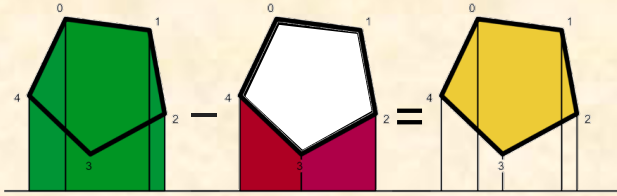
Area of a Polygon

- Unsigned Area
  - Green trapezoids have $x_i < x_{i+1}$

$$A = \frac{(y_i + y_{i+1})(x_{i+1} - x_i)}{2}$$

  - Red trapezoids have $x_i > x_{i+1}$

$$A = \frac{(y_i + y_{i+1})(x_i - x_{i+1})}{2}$$

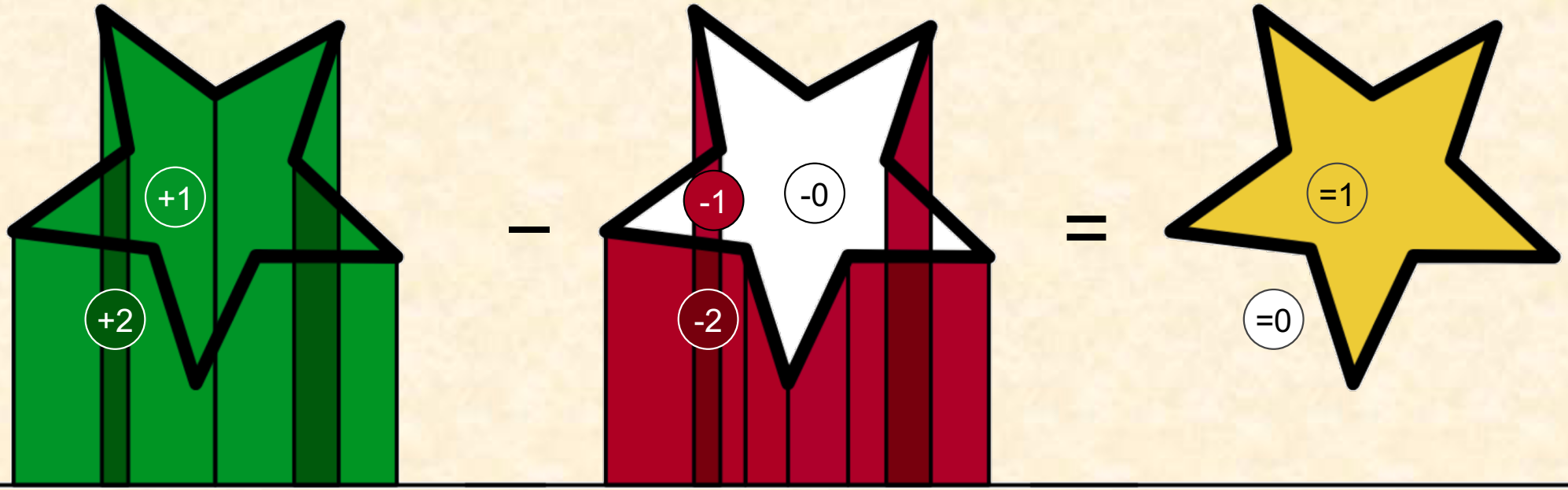


- Signed Area
  - Green trapezoid have $A > 0$
  - Red trapezoids have $A < 0$

$$A = \frac{(y_i + y_{i+1})(x_{i+1} - x_i)}{2}$$

Area of a Polygon

- For non-convex shapes, the positive/negative regions can overlap.
- Some regions are double-counted (+ or -)
- It still works!



Area of a Polygon

- From Wikipedia: the Shoelace formula

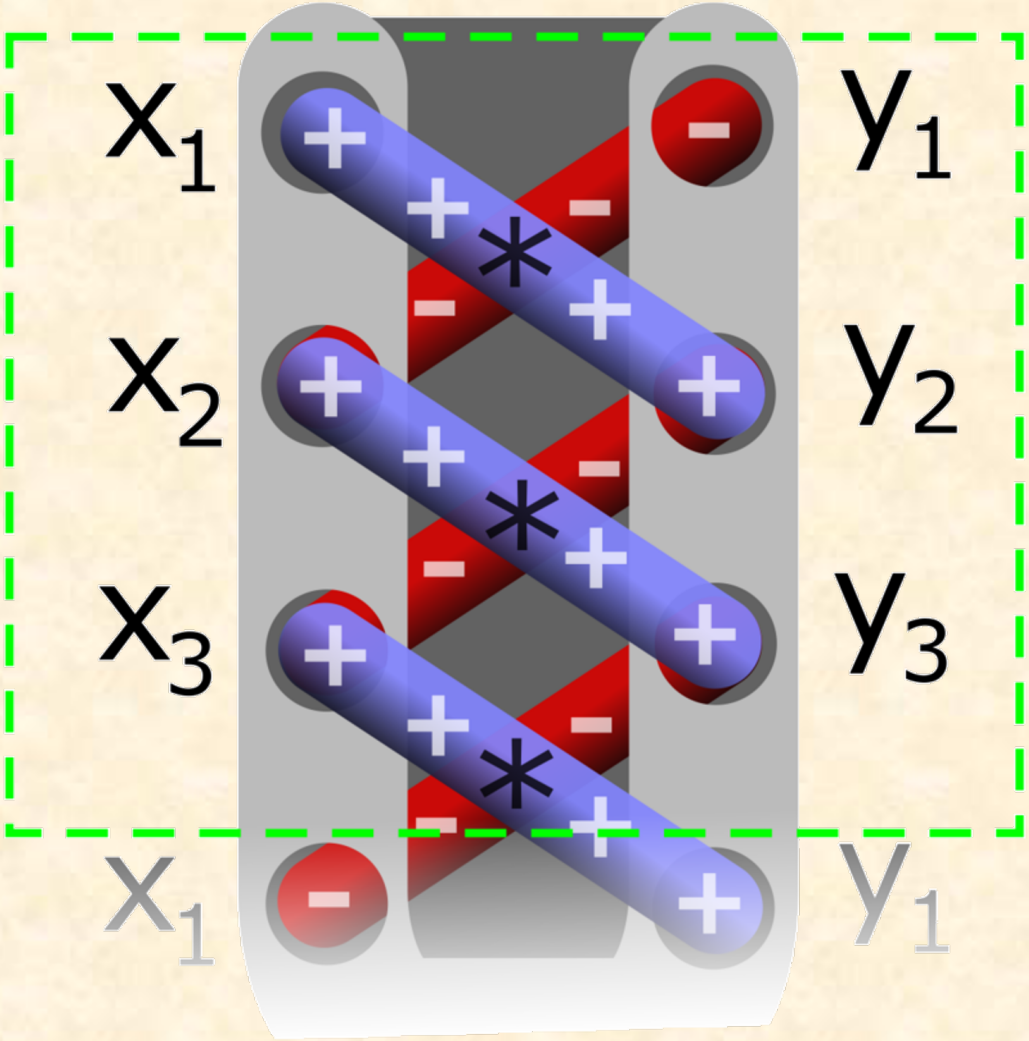
$$A = \frac{1}{2}\left(\sum_{i=0}^{n-1} x_i y_{i+1} - \sum_{i=0}^{n-1} x_{i+1} y_i\right)$$

The points are (x_0, y_0), (x_1, y_1), (x_2, y_2), …, (x_{n-1}, y_{n-1}), with $(x_n, y_n) = (x_0, y_0)$ for convenience

* will be negative if the points are listed in counterclockwise order

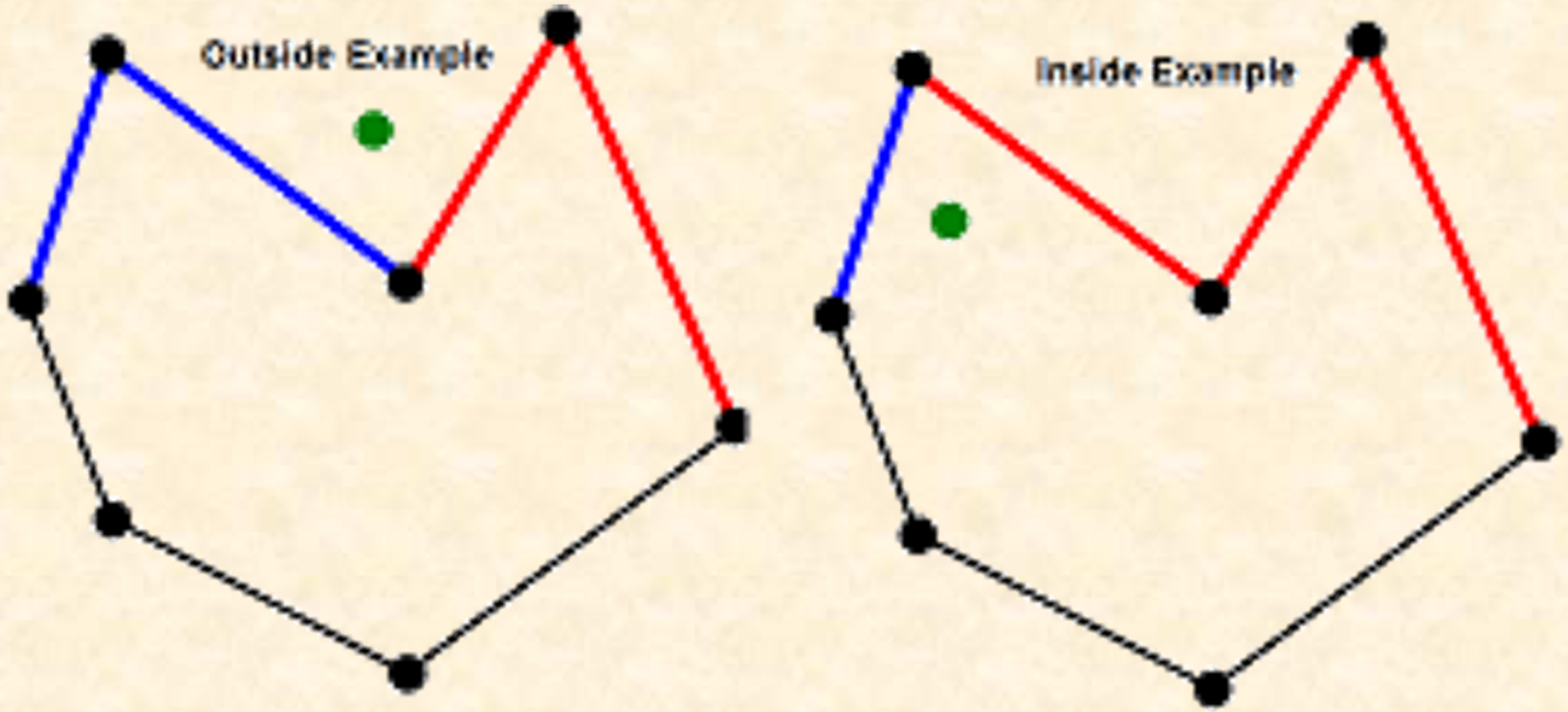
- Punchline: computing `AreaOf(polygon)` in linear time (in the number of points)

The Shoelace Formula



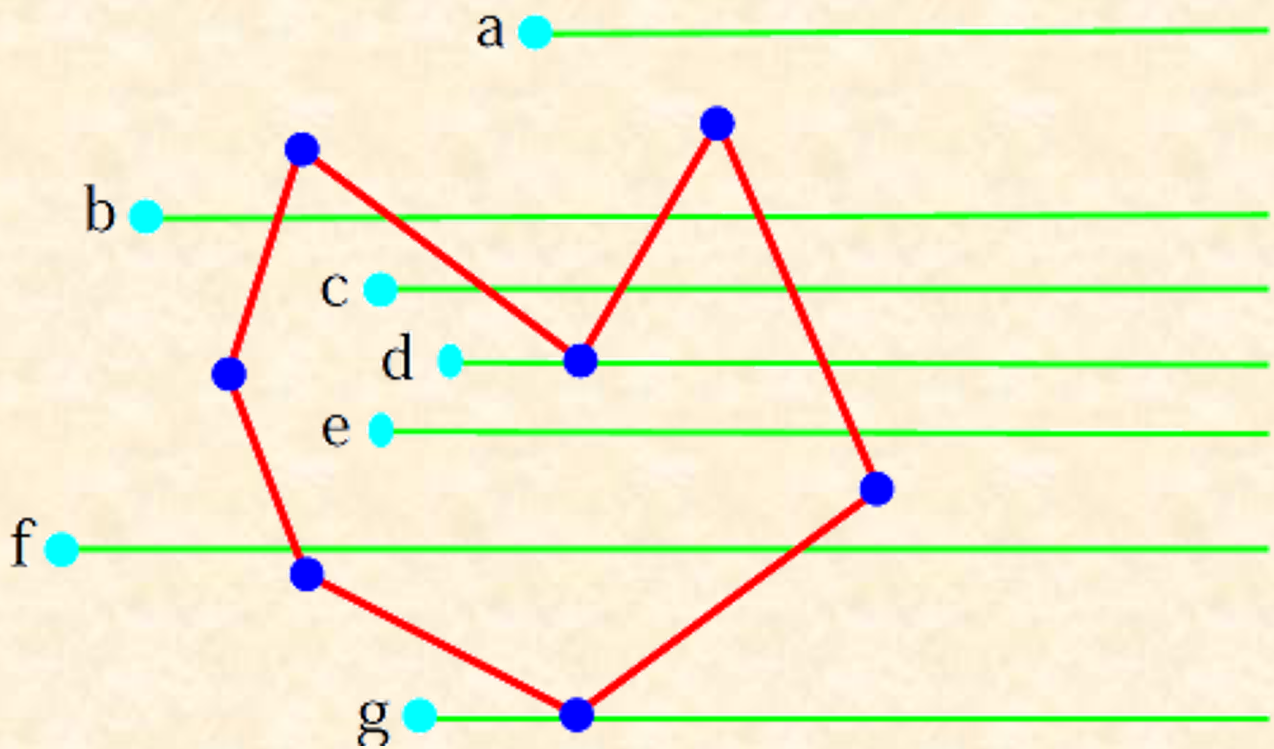
Point Inside a Polygon (1)

Given a polygon (defined by *N* points) and a point *P,* determine if P is inside the polygon.



Point Inside a Polygon (2)

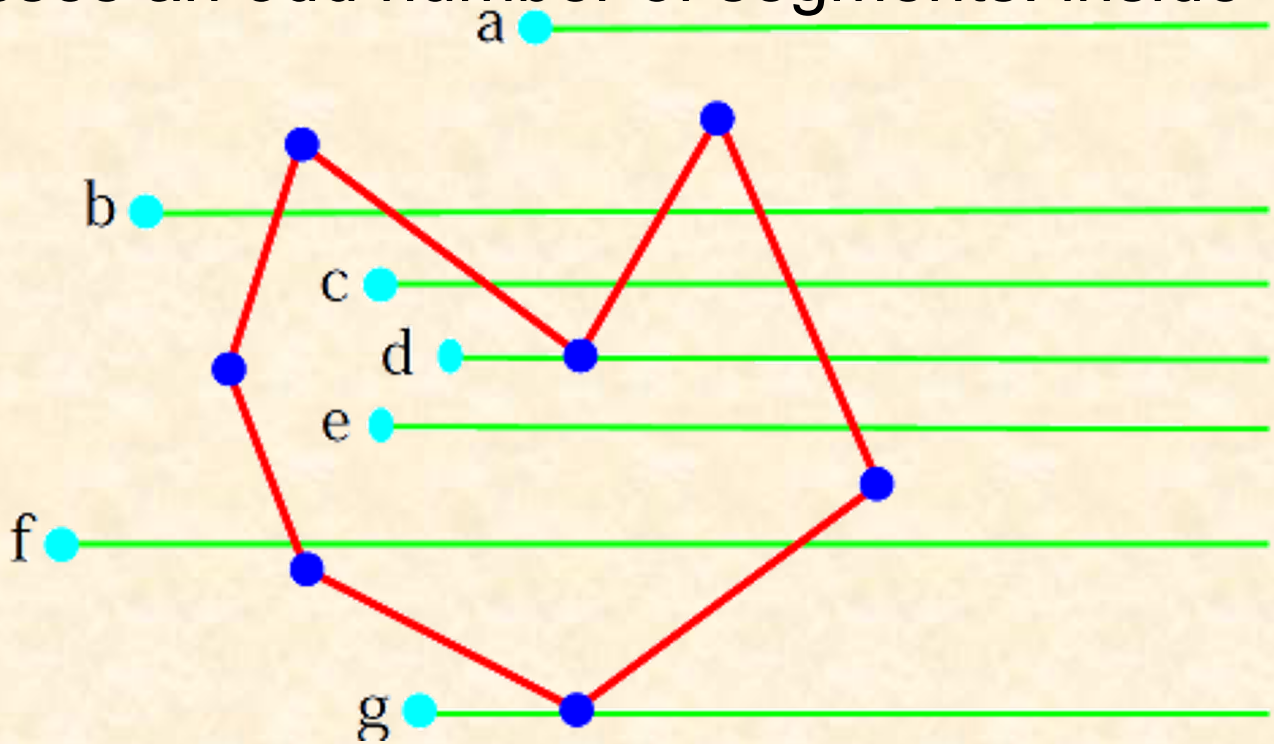
Given a polygon (defined by *N* points) and a point *P,* determine if P is inside the polygon.



Point Inside Polygon (3)

Project a horizontal line through the point

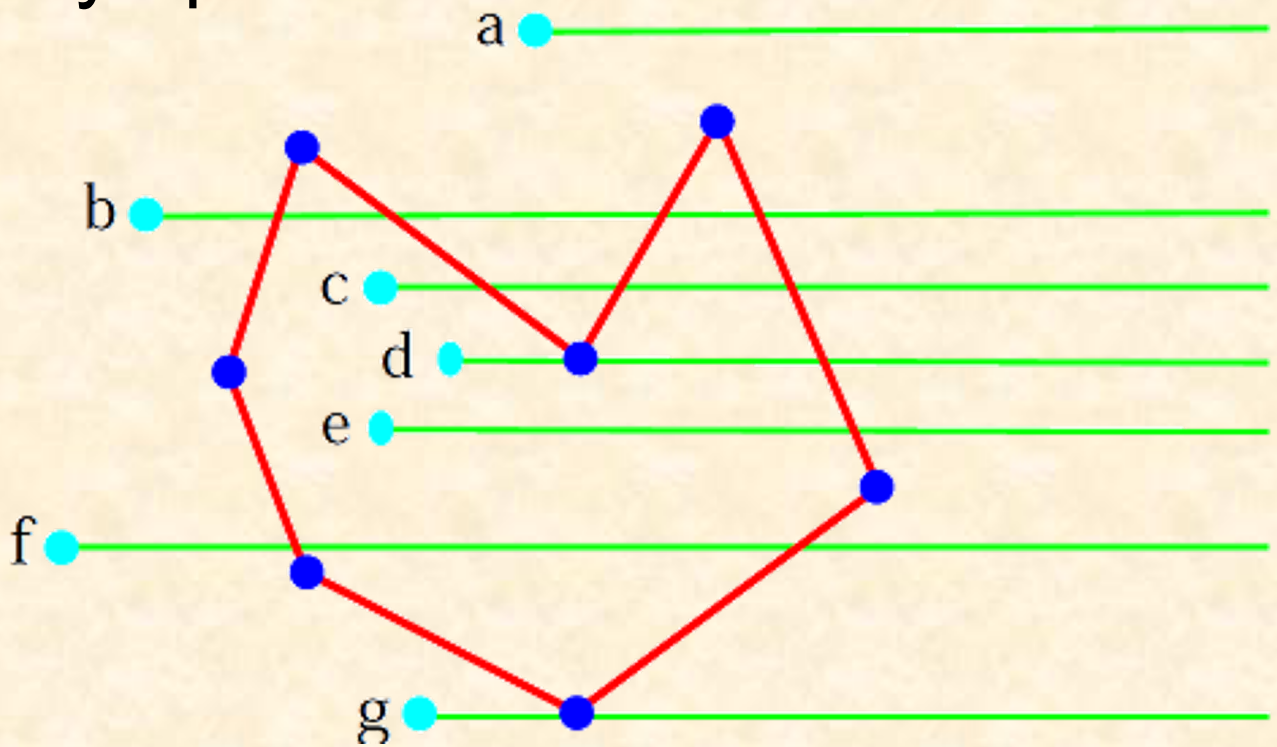
- Crosses an even number of segments: outside
- Crosses an odd number of segments: inside



*Reminder: all comparisons use epsilon!

Point Inside Polygon (4)

The special cases when lines cross polygon's vertices can be handled by "wiggling" the points up/down by epsilon.



*Reminder: all comparisons use epsilon!

Polygons

Data Structures & Algorithms

Additional Topics

Data Structures & Algorithms

Computing Min & Max at Once

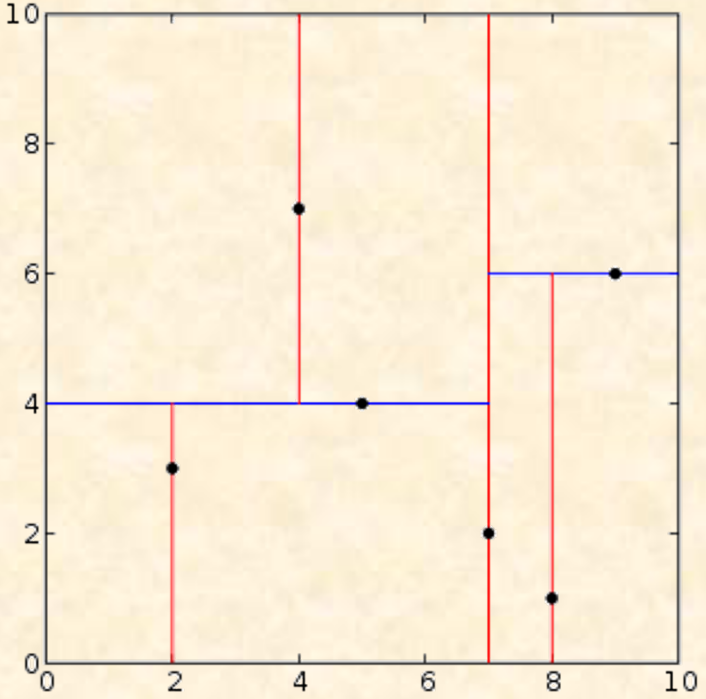
- Computing min of *N* values takes *N* - 1 comparisons
- Computing max of *N* values takes *N* - 1 comparisons
- However, both values can be computed using 1 + ceil (3 * (*N* – 2) / 2) comparisons
  - For each pair of input values, compare the larger of the two with running max and the smaller of the two with running min

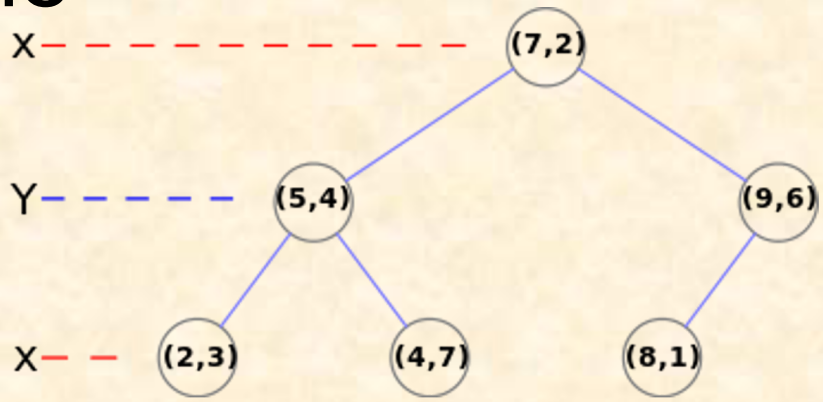
k-d (*k*-dimensional) trees (1)

- A binary tree to represent coordinate data
- The data structure works in *k* dimensions, but we will use *k*=2 for illustration
- Construction
  - Pick one coordinate and partition all points using the chosen value (linear time)
  - Then recurse to each partition, but use a different coordinate next
  - Build a tree

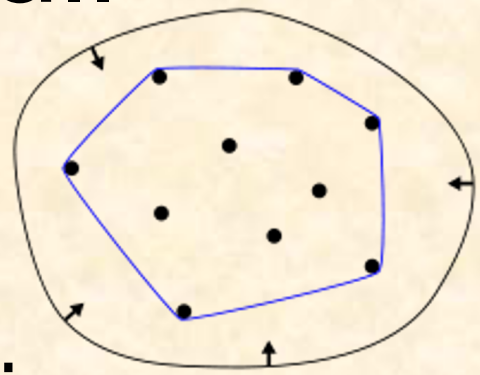
k-d trees (2)

- Each tree node represents a vertical or horizontal cut
  - One point recorded @node
- Construction process takes O(*N* log *N*) time
- Search takes O(log *N*) time
  - Similar to binary search, but alternates dimensions





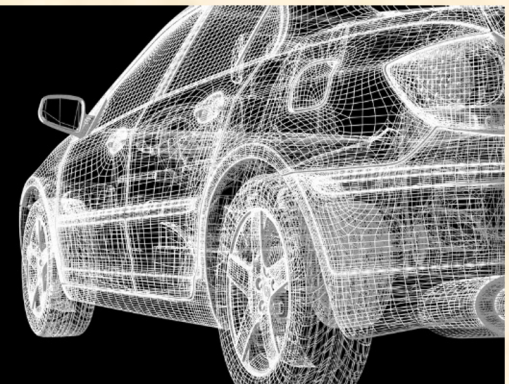
Optional: Convex Hull

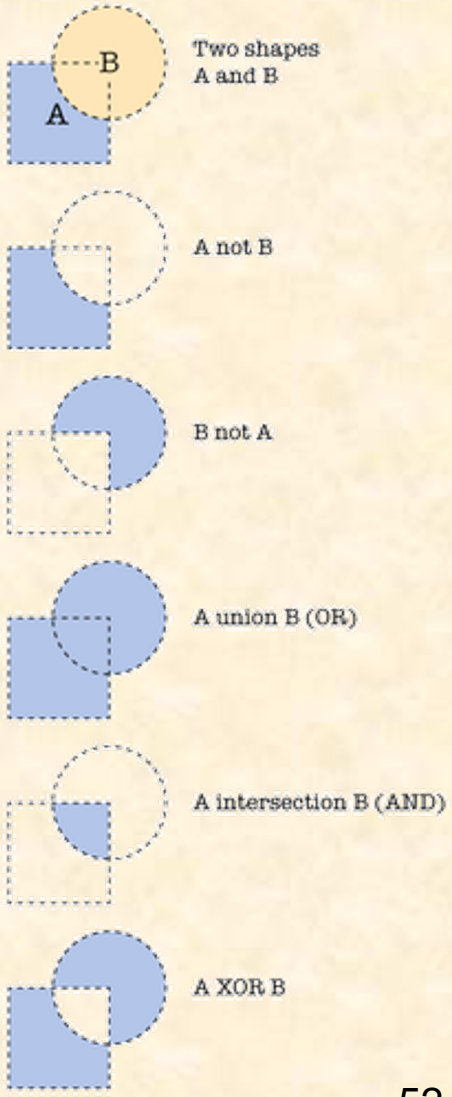


- Find the smallest polygon enclosing the given pointset
- Intuition: if points are nails stuck in a board, then a rubber band stretched around the nails assumes the shape of the convex hull, **in O(1) time!**
- Several algorithms find convex hull of *N* points in 2D in O(*N* log *N*) time
  - Can be accomplished with a *sweeping line*

Optional: Polygonal Operations

- Boolean ops on polygonal shapes:
  - Union, intersection, set difference
- Polygonal shapes can represent mechanical parts in software for solid-modelling CAD
  - These techniques can be extended to 3D and also to smooth shapes





Additional Topics

Data Structures & Algorithms

Summary

- Geometric objects
  - Basic: points, segments, lines
  - Compound: pointsets, polygons, meshes
- Transforms: shifts, scaling, mirroring, rotation
- Geometric tests for points, segments, lines
- Efficient geometric algorithms
  - Closest pair of points, closest-point queries
  - Area of polygons, point-in-polygon test, etc
- Floating-point comparisons must use epsilon

Interview Question

- Given *n* points, find a line through two of the given points that divides the remaining points into two halves, in $O(n \log n)$ time
- Assumptions to make it slightly easier:
  - There are an even number of points
  - No three points are collinear
- Hint: draw it out!