TDDD95 Algorithmic Problem Solving
Le 11 – Computational Geometry

Herman Appelgren
Dept of Computer and Information Science
Linköping University

Outline

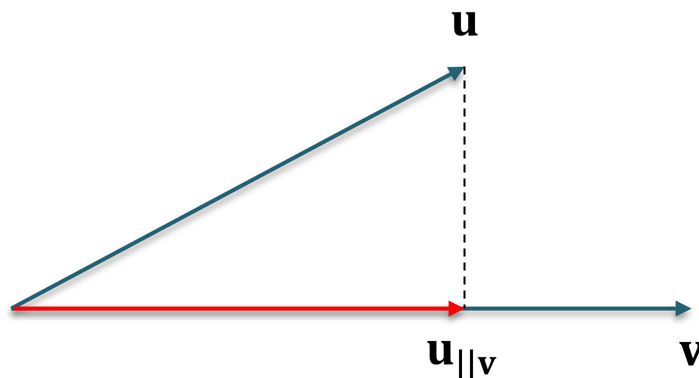
- ~~Exercise 11: Search~~
  - ~~A: Yet Satisfiability Again!~~
  - ~~B: Gokigen Naname~~
  - ~~B: Square Fields (hard)~~
  - ~~C: Maximum Loot~~
- Computational Geometry
  - 2D Geometry Toolbox
  - 2D Lines (Lab 4.3-4.4)
  - Polygons (Lab 4.1-4.2)
  - Convex Hull (Lab 4.7)
  - Closest Point Pair (Lab 4.5-4.6)
- Linear Recurrences (Lab 4.9)

2D Geometry Toolbox

- Scalar Product
  - $\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}||\mathbf{v}|\cos(\theta)$ where θ is the angle between **u** and **v**.
  - **u** and **v** are *orthogonal* iff $\mathbf{u} \cdot \mathbf{v} = 0$.
  - If $\mathbf{u} \cdot \mathbf{v} > 0$ the angle between them is less than 90 degrees, and vice versa.
  - Calculated as the sum of the products of each coordinate, i.e. $u_x v_x + u_y v_y$ in 2 dimensions.
- Cross Product
  - The cross product $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ is the unique vector that satisfies
    - $|\mathbf{w}| = |\mathbf{u}||\mathbf{v}|\sin(\theta)$
    - Is orthogonal to both **u** and **v** (i.e. $\mathbf{w} \cdot \mathbf{u} = \mathbf{w} \cdot \mathbf{v} = 0$).
    - **u**, **v** and **w** satisfies the right-hand rule.
  - The last two properties are the most important in our context.
  - Typically only defined in 3D, where $\mathbf{w} = \left(\mathrm{u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x}\right)^T$.
  - We extend the definition to 2D, where it is the *scalar* $\mathrm{u_x v_y - u_y v_x}$, i.e. the z-coordinate of $\left(\mathrm{u_x, u_y, 0}\right) \times (\mathrm{v_x, v_y, 0})$.

Projection

- The scalar product can be used to compute the *orthogonal projection* $\mathbf{u}_{||\mathbf{v}}$ of $\mathbf{u}$ onto $\mathbf{v}$.
  - The orthogonal projection is the shortest vector such that $\mathbf{u} - \mathbf{u}_{||\mathbf{v}}$ is orthogonal to $\mathbf{v}$.
  - Intuitively, it is the part of $\mathbf{u}$ that is parallel to $\mathbf{v}$.
  - The *projection formula* states that $\mathbf{u}_{||\mathbf{v}} = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{v}|^2}\mathbf{v} = \frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}\mathbf{v}$.

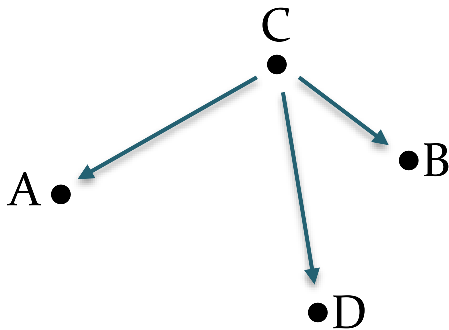


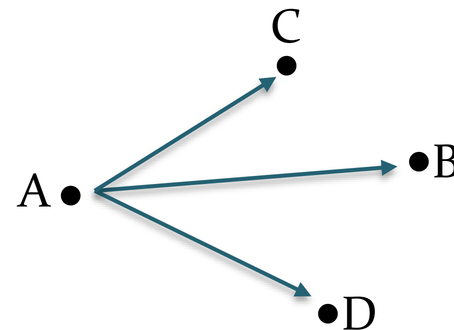
2D Geometry Toolbox

- Do $\mathbf{u}$ and $\mathbf{v}$ point in the same direction?
  - $\mathbf{u}$ and $\mathbf{v}$ are parallel iff $\mathbf{u} \times \mathbf{v} = 0$.
  - If additionally $\mathbf{u} \cdot \mathbf{v} > 0$, then they point in the same direction.
- Is the point C on the line between points A and B?
  - If $\mathbf{CA} \times \mathbf{CB} = 0$, then C is on the (infinite) line that intersects A and B.
  - If additionally $\mathbf{CA} \cdot \mathbf{CB} < 0$ then A and B are on opposite sides of C, so C is on the line segment between A and B.

2D Geometry Toolbox

- Does the line segment from A to B intersect the line segment from C to D?
  - If $\mathbf{CA} \times \mathbf{CD}$ and $\mathbf{CB} \times \mathbf{CD}$ have different signs, then A and B are on opposite sides of the line between C and D.
  - If $\mathbf{AC} \times \mathbf{AB}$ and $\mathbf{AD} \times \mathbf{AB}$ have different signs, then C and D are on opposite sides of the line between A and B.
  - If both are true, then the line segments intersect.



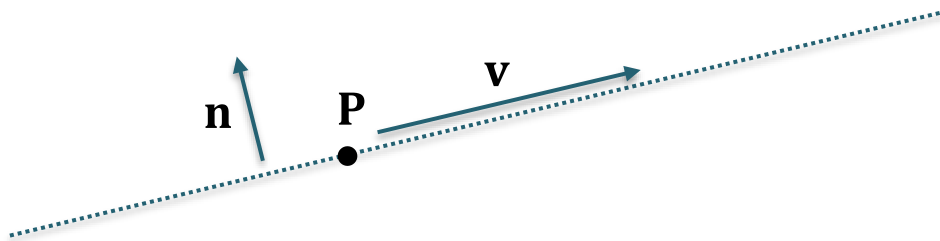


2D Geometry Toolbox

- Use scalar or cross products whenever possible!
  - Very fast to compute.
  - Very numerically stable, especially compared to trigonometric functions, square roots and similar nonlinear functions.
  - If the coordinates are integer-valued, so are the scalar and cross products.
  - Can be used to answer many basic geometric questions in 2D and 3D.

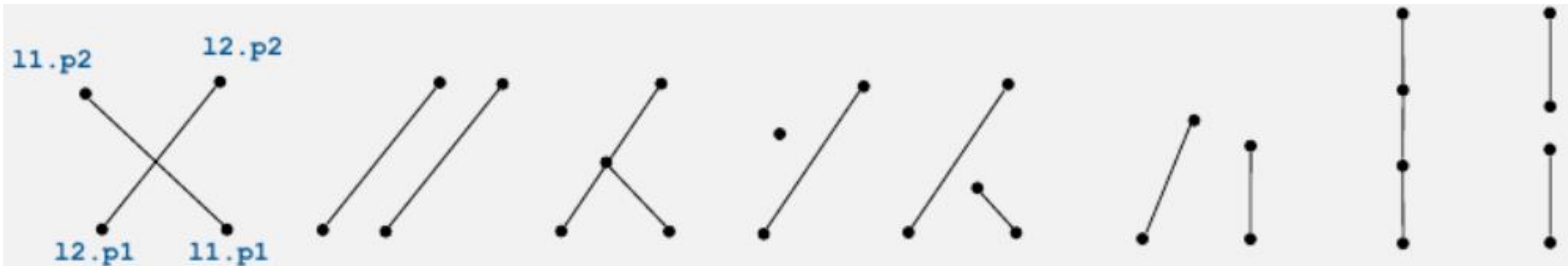
Line Representations

- Parametric representation
  - All points that can be written $\mathbf{P} + t\mathbf{v}$ where $\mathbf{P}$ is any point on the line, $\mathbf{v}$ is a *tangent vector* pointing in the line direction, and t is some scalar.
  - Convenient to construct, since a point and a tangent vector is often easy to obtain.
  - Useful when we operate on vectors.
- Normal form (2D only)
  - All points (x, y) that satisfy $ax + by - c = 0$ where a, b and c are scalar parameters.
  - A useful property is that the vector $\mathbf{n} = (a, b)$ is orthogonal to the line.
  - Provides a condition that is easy to check for a given point.
  - Useful when solving for intersections and similar.



2D Line Segment Intersection (Lab 4.3)

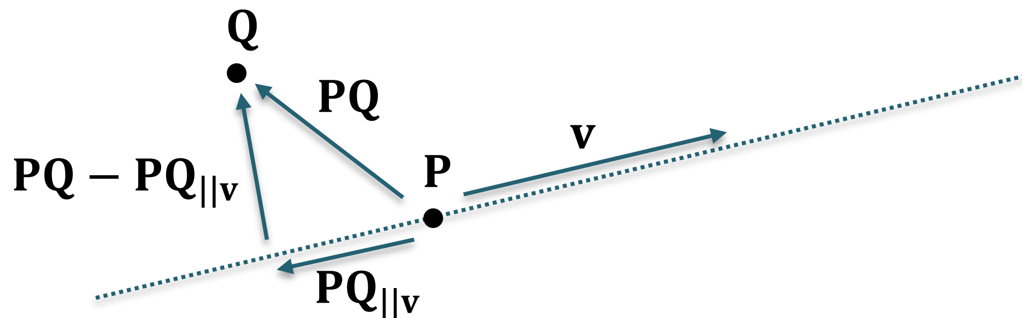
- Do the (infinite) lines L1 and L2 intersect?
  - If the lines are parallel (check using normal or tangent vectors), then they either have no intersection, or they coincide.
  - Find the intersection point (x, y) that satisfies the normal equation for both lines. In 2D, this is a 2x2 system of linear equations, and the solution can be hard-coded.
- Do the line segments L1 and L2 intersect?
  - As above, but check that the intersection is between the endpoints for both lines (see toolbox).
  - Note that unlike infinite lines, line segments may have a unique intersection even if they are parallel.



(image: Tommy Färnqvist)

2D Line Segment Distance (Lab 4.4)

- What is the distance between a point $\mathbf{Q}$ and the (infinite) line $\mathbf{P} + t\mathbf{v}$?
  - Compute the orthogonal projection $\mathbf{PQ}_{||\mathbf{v}}$.
  - The distance is the length of $\mathbf{PQ} - \mathbf{PQ}_{||\mathbf{v}}$.
- What is the closest distance between a point on the line segment L1 and a point on the line segment L2?
  - If there is an intersection, the distance is zero.
  - Otherwise, the closest distance must involve one of the endpoints. Check the distance between each endpoint and the other line segment, bearing in mind that the closest distance may be between two endpoints.

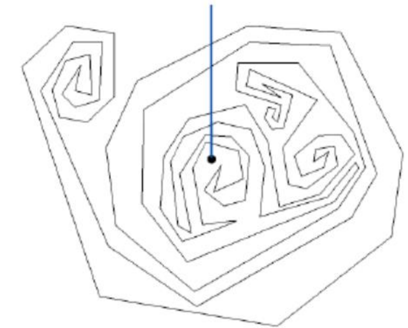


Polygons

- A polygon is a closed curve consisting of N line segments.
  - Often represented by the sequence of line segment end points $\boldsymbol{p_i}$.
  - If the line segments only intersect at the endpoints, then the polygon is *simple*.
  - A simple polygon has a well-defined inside, outside and area.
  - We assume simple polygons from here onward.
- Polygon Area (Lab 4.1)
  - Calculated as $A = \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{p_i} \times \boldsymbol{p_{i+1}}$ if we assume that $\boldsymbol{p_1} = \boldsymbol{p_{N+1}}$.
  - The polygon is given in counterclockwise order if $A > 0$, and vice versa.
  - The geometric area is given by $|A|$.

Point in Polygon (Lab 4.2)

- Is the point **Q** contained inside the polygon P?
- Assume that **Q** isn't on any of P's line segments.
  - Straight-forward to check using basic toolbox.
- Method 1: Ray Casting
  - **Q** is inside P iff an infinite ray from **Q** intersects with P an odd number of times.
  - Count number of intersections.
  - Make sure to handle intersections at end points.

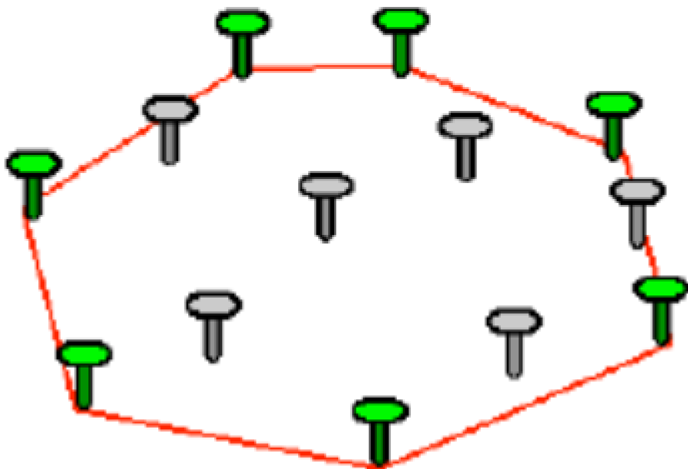


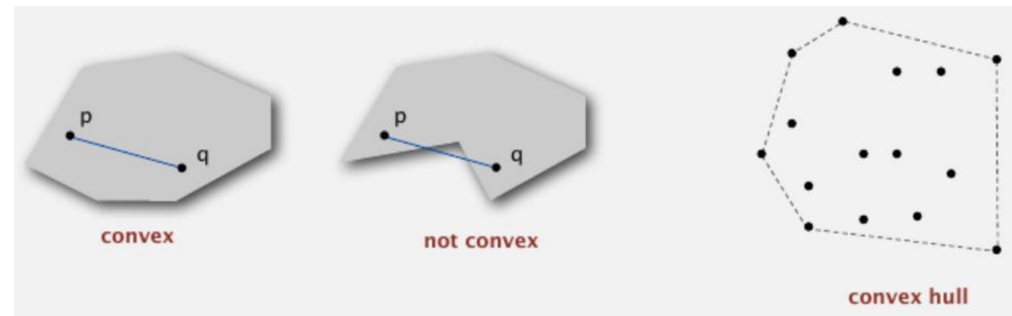
- Method 2: Winding Number
  - Imagine that you stand at **Q** and track the polygon's edges by turning.
  - If you are inside the polygon, you will in total turn 360 degrees, otherwise you will turn 0 degrees.
  - Might run into rounding errors, and is typically slower than the ray casting method due to requiring trigonometric functions.

(image: Tommy Färnqvist)

Convex Hull (Lab 4.7)

- **Definition:** A set of points S is *convex* if the line segment between any pair of points is contained in the set.
- **Definition:** The *convex hull* of S is the smallest convex set containing all points in S.
  - Equivalently: The shortest perimeter enclosing all points.
  - Equivalently: The smallest area convex polygon enclosing all points.
  - Intuition: Enclosing all points using a rubber band.
  - Important e.g. in optimization and as a geometry preprocessing step.

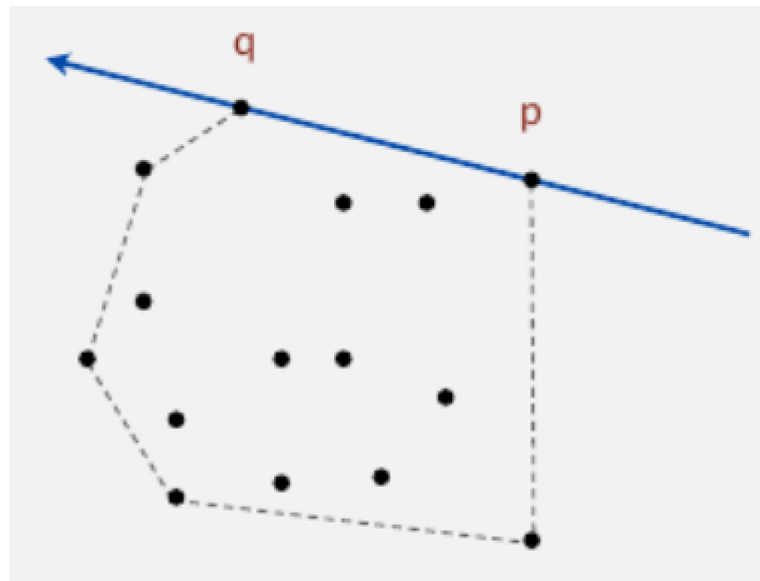




(image: Tommy Färnqvist)

Convex Hull – Naive Algorithm

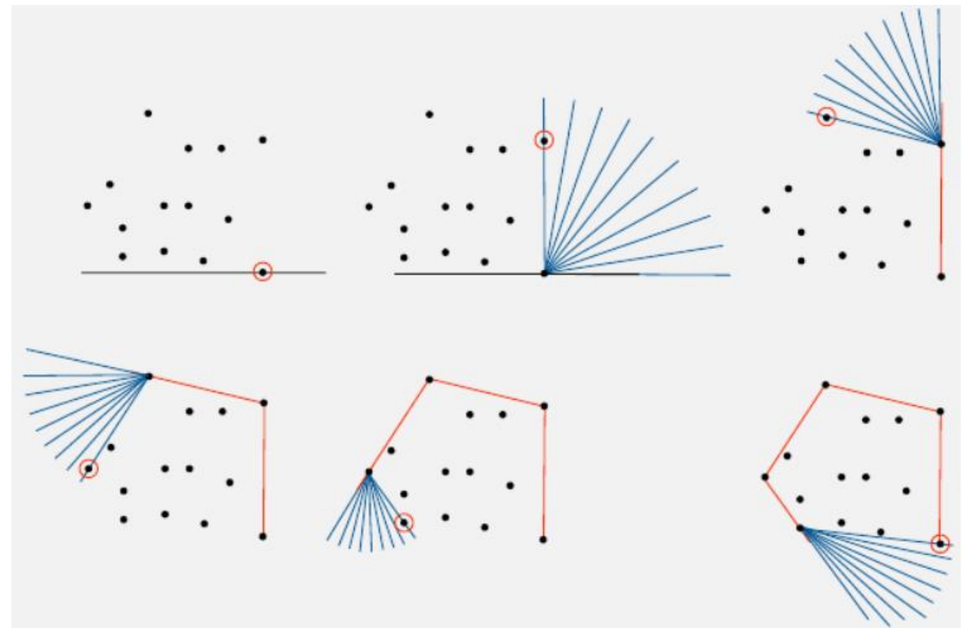
- **Observation:** If p and q are on the hull, then all other points are on the same side of the line through A and B (or on it).
  - Check this condition for all pairs of points.
- **Time complexity:** $O(N^2)$ pairs, $O(N)$ points to check for each pair. Total time complexity $O(N^3)$.



(image: Tommy Färnqvist)

Package Wrap Algorithm / Jarvis March

- **Observation:** The extreme point in any direction will always be on the hull.
  - Start with the point with the smallest y coordinate and an orientation directed in the positive x direction.
  - Find the point with the smallest counter clockwise change in orientation.
  - Repeat until you return to the start.
- **Time complexity:** $O(hN)$ where h is the number of points on the hull.
  - Worst case: $h = N$
  - When sampled uniformely from a disc: $O(N^{\frac{1}{3}})$
  - When sampled uniformely from a convex polygon of fixed size: $O(\log N)$

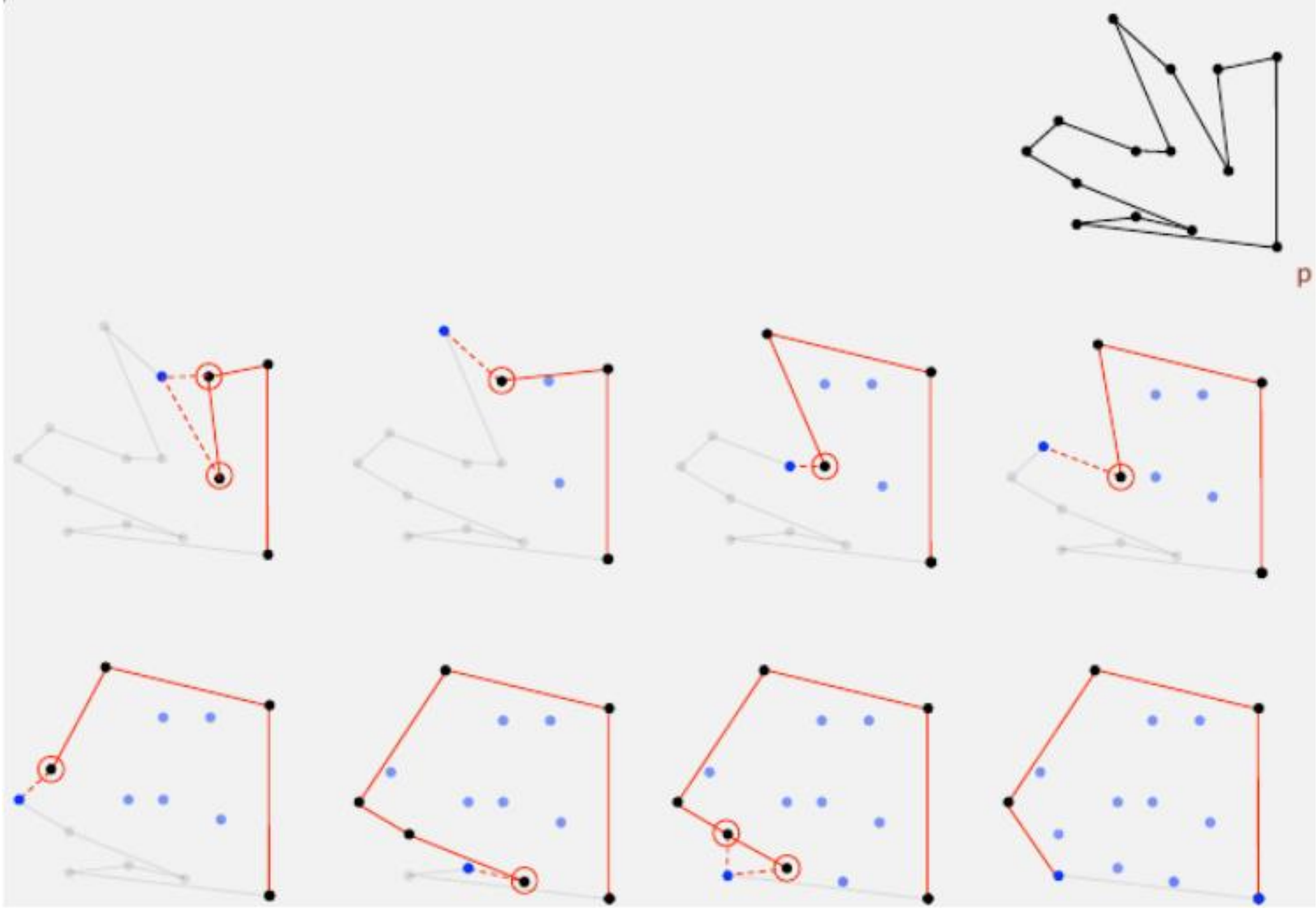


(image: Tommy Färnqvist)

Graham Scan

- **Observation:** All angles on the hull are in the same direction.
  - Again, start with the point **Q** with the smallest y coordinate.
  - Create a simple polygon by sorting by the slope from **Q** to each point.
  - Traverse the polygon and remove the previous point from the hull whenever the next point would create a turn in the wrong direction.
- **Time complexity:** Find **Q** in $O(N)$, sort in $O(N \log N)$, traverse in $O(N)$. In total $O(N \log N)$.
- Make sure to handle if points on the hull are colinear! If so, only the two extreme points should be included.

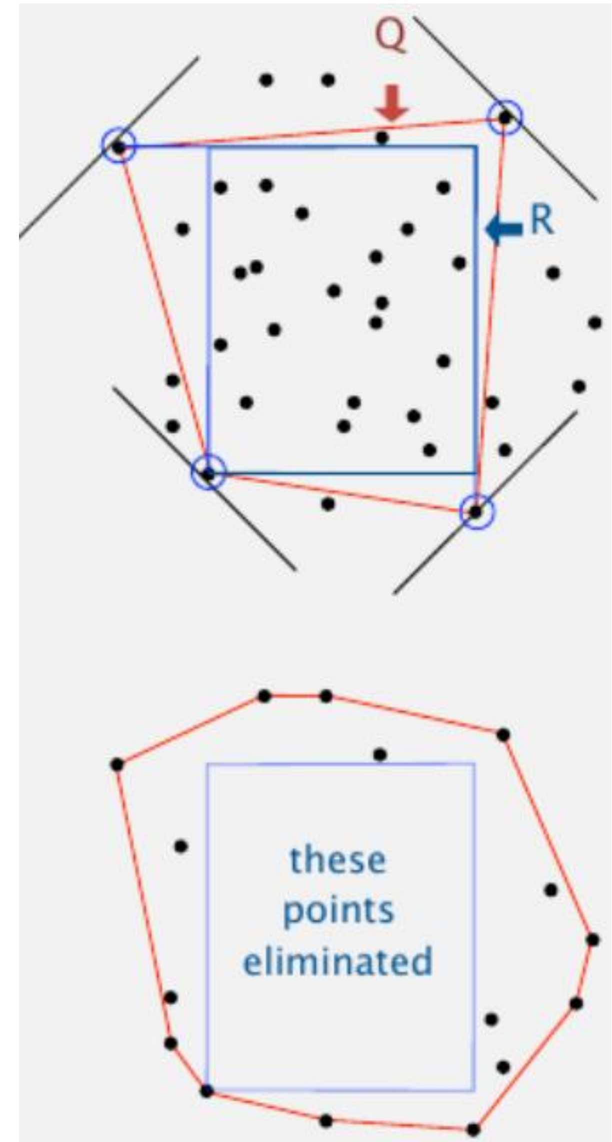
Graham Scan



(image: Tommy Färnqvist)

Convex Hull Optimization

- Optional optimization
  - Create a quadrilateral based on 4 random points in the set.
  - Any points contained in this quadrialateral cannot be on the convex hull.
  - In practice, this often eliminates most points in linear time.



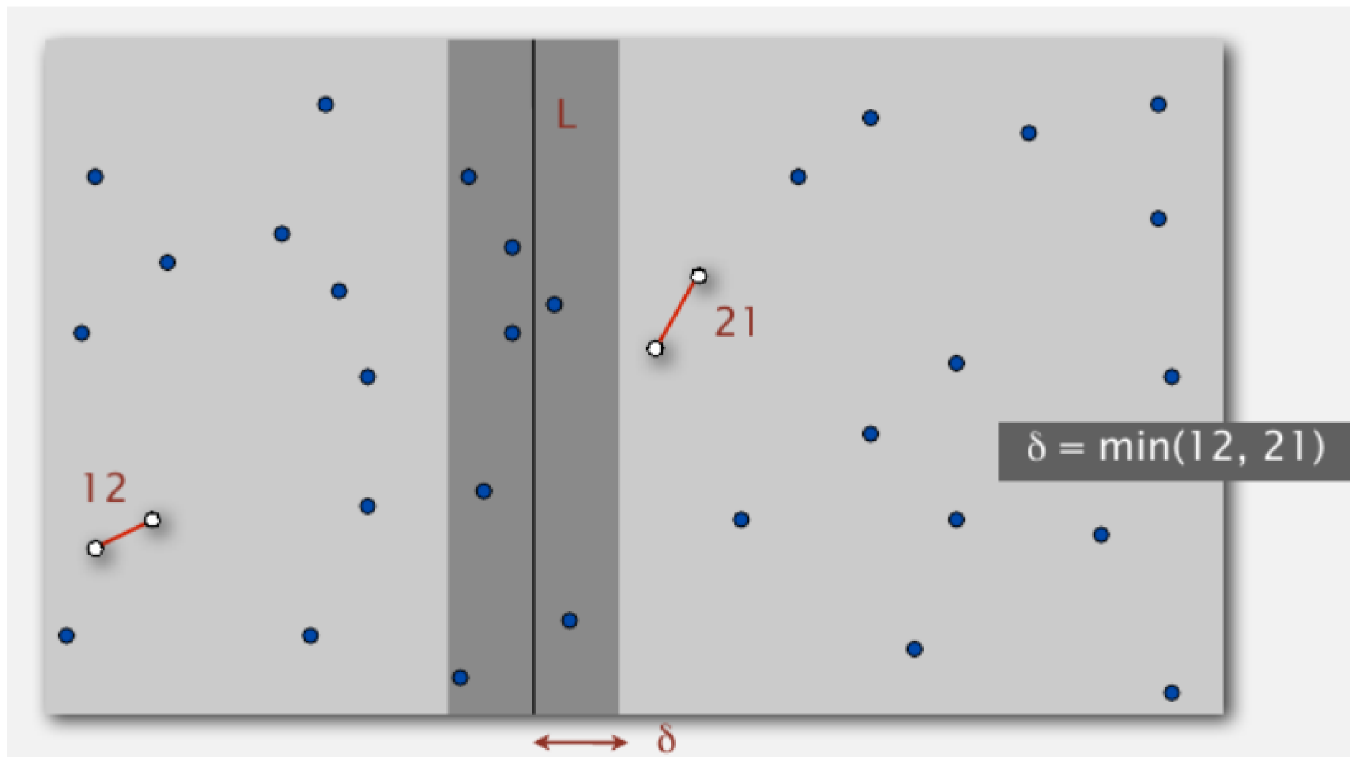
(image: Tommy Färnqvist)

Closest Pair (Lab 4.5-4.6)

- Given a set S of N 2D points, find the pair with the smallest distance between them.
  - Naive brute-force: Check all pairs in $O(N^2)$.
  - In 1D: Sort in $O(N \log N)$, then find the closest pair in $O(N)$.
- We use a Divide-and-Conquer approach to achieve $O(N \log N)$ in 2D as well!
  - First, sort the points by their x coordinate.
  - Divide: Create two approximately equal sets S1 and S2 using a vertical line.
  - Conquer: Find the closest pair in each set recursively.
    - Base case: 2 points.
  - Combine: Find the closest pair where one point is in S1 and one in S2.
  - All steps except the initial sorting can be done in linear time, so the time complexity is $O(N \log N)$.

Closest Pair (Lab 4.5)

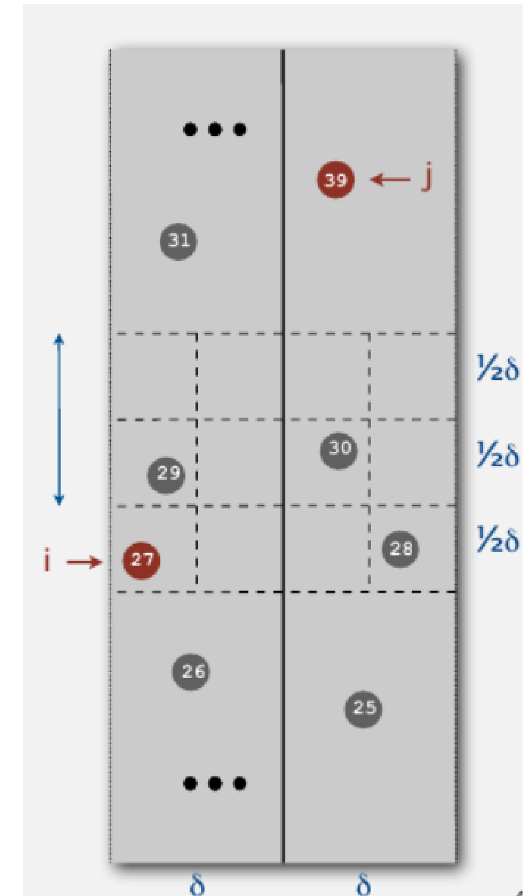
- The combination step is the tricky part.
  - Naively testing all pairs would put us back in $O(N^2)$.
  - Note that if δ is the closest distance found so far, we only have to consider points withing δ from the dividing line.
  - For uniformly distributed points, this is sufficient for $O(N)$.



(image: Tommy Färnqvist)

Closest Pair (Lab 4.6)

- To further improve, don't consider all point pairs within the band.
  - A pair closer than δ must be within 12 positions of each other when the band is sorted by y coordinate.
  - Results in $O(N \log N)$ in this step regardless of point distribution.
  - Can be improved to $O(N)$ by sorting once and reusing the result.
- Why 12 positions?
  - Note that there can be at most one point per $\frac{1}{2}\delta$ box, or they would already be closer than δ.
  - Two points separated by at most δ must be at most two rows apart.
  - Still true if we reduce 12 to 7.



(image: Tommy Färnqvist)

Linear Recurrences (Lab 4.9)

- A linear recurrence of order N is a sequence of numbers where each number is a linear combination of the previous N numbers and a constant: $x_k = a_0 + \sum_{i=1}^{N} a_i x_{k-i}$.
  - Example: $x_1 = 2$, $x_k = 1 + 2x_{k-1}$ is a linear recurrence of order 1.
  - Example: The fibonacci series $x_1 = 1$, $x_2 = 1$, $x_k = x_{k-1} + x_{k-2}$ is a linear recurrence of order 2.
- Given a linear recurrence, how can we compute x_k?
  - If we the have N initial numbers, we can compute any following x_k.
  - The naive algorithm runs in $O(Nk)$, which is much too slow for large k.

Linear Recurrences (Lab 4.9)

- **Observation:** We can rewrite the recurrence on matrix form.
  - Let s_k be the state vector $(1, x_k, x_{k-1}, \dots, x_{k-N+1})^T$.
  - The next state vector $s_{k+1} = As_k$ where

$$A = \begin{pmatrix} 1 & & & & \\ a_0 & a_1 & a_2 & \dots & a_N \\ & 1 & & & \\ & & \ddots & & \end{pmatrix}$$

  - Since matrix multiplication is linear, $s_{i+k} = A^k s_i$.
- We've transformed the problem to computing a large power of a matrix.
  - Can be done efficiently using $O(\log k)$ matrix multiplications.
  - See the live Number Theory slides on binary exponentiation (Apr. 13).

Outline

- Exercise 11: Search
  - A: Yet Satisfiability Again!
  - B: Gokigen Naname
  - B: Square Fields (hard)
  - C: Maximum Loot
- Computational Geometry
  - 2D Geometry Toolbox
  - 2D Lines (Lab 4.3-4.4)
  - Polygons (Lab 4.1-4.2)
  - Convex Hull (Lab 4.7)
  - Closest Point Pair (Lab 4.5-4.6)
- Linear Recurrences (Lab 4.9)