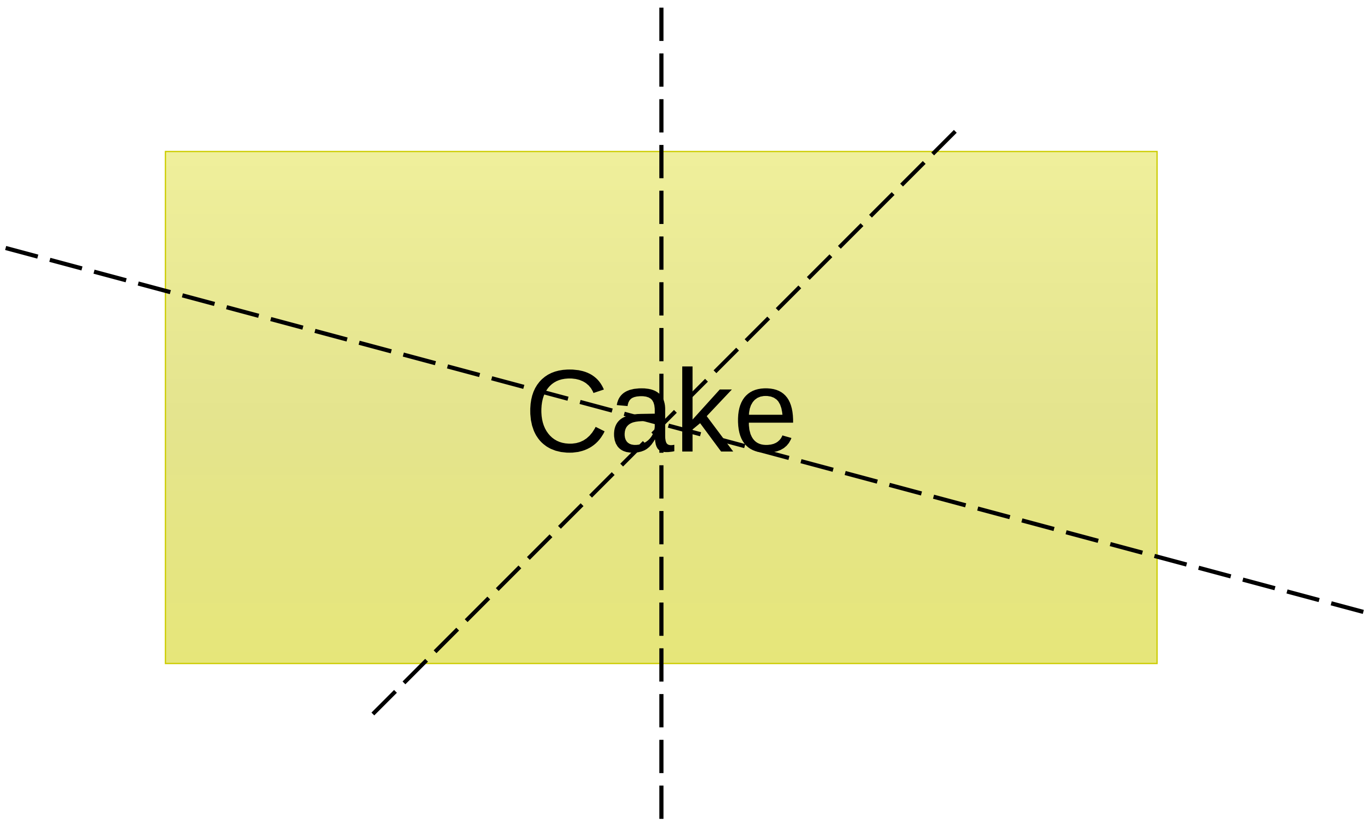
CS133
Computational Geometry

Intersection Problems

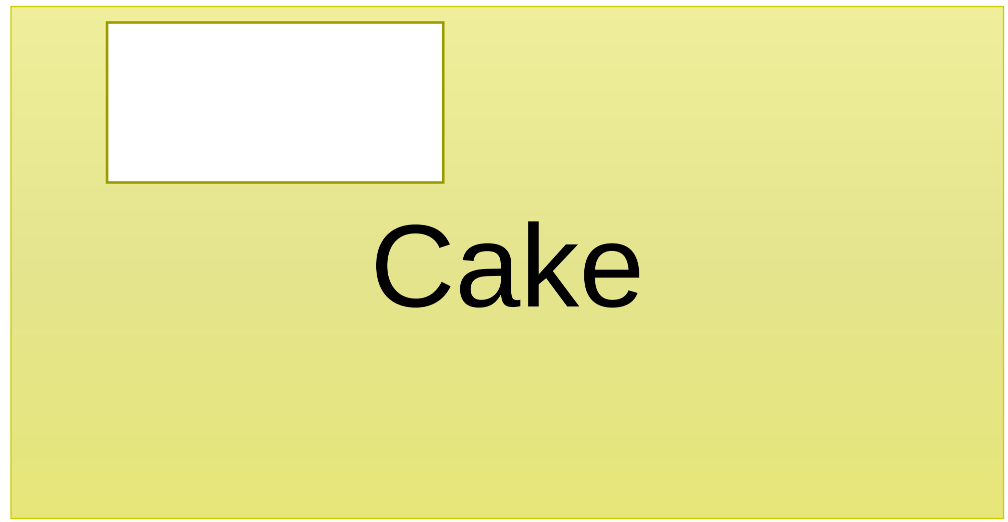
Riddle: Fair Cake-cutting

- Using only one straight-line cut, how to split the cake into two equal pieces (by area)?



Riddle: Fair cake-cutting

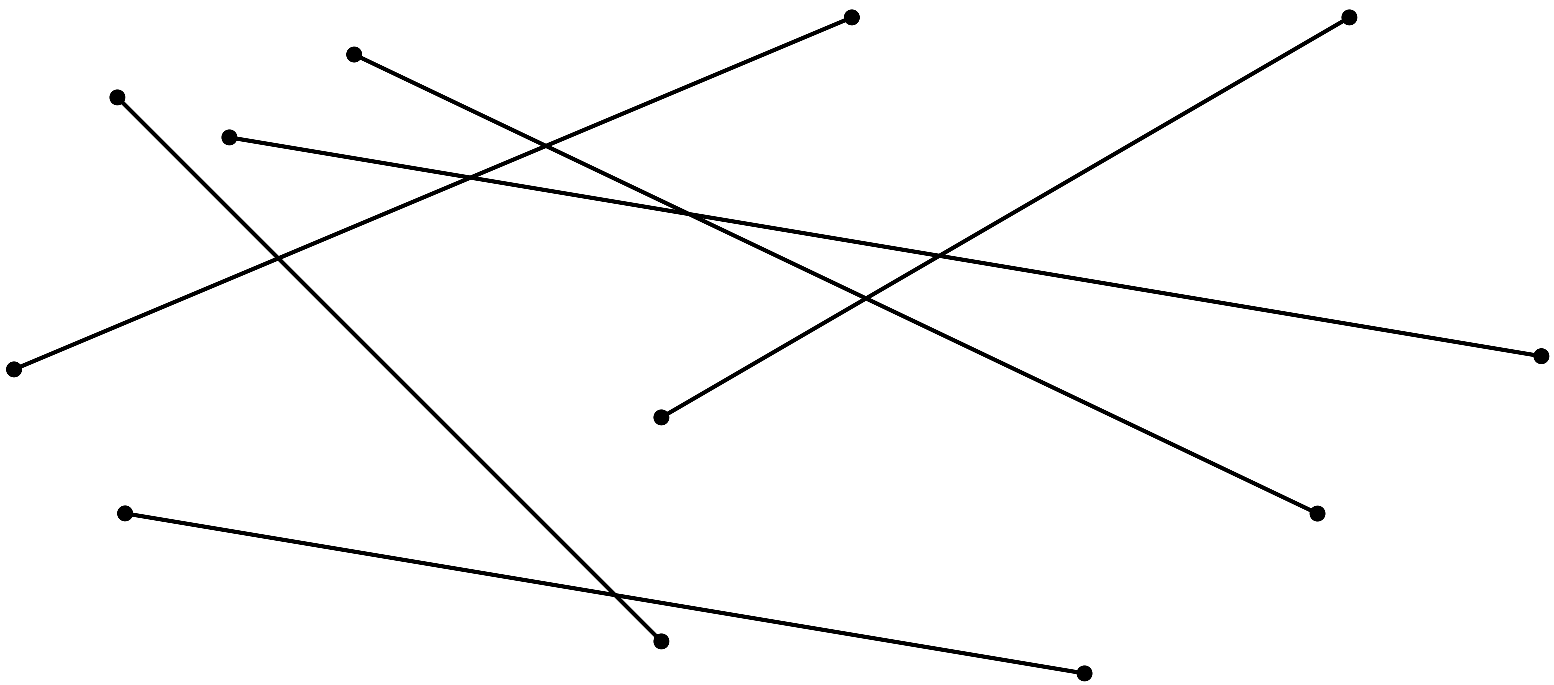
- Mixed cake!
- Still one cut



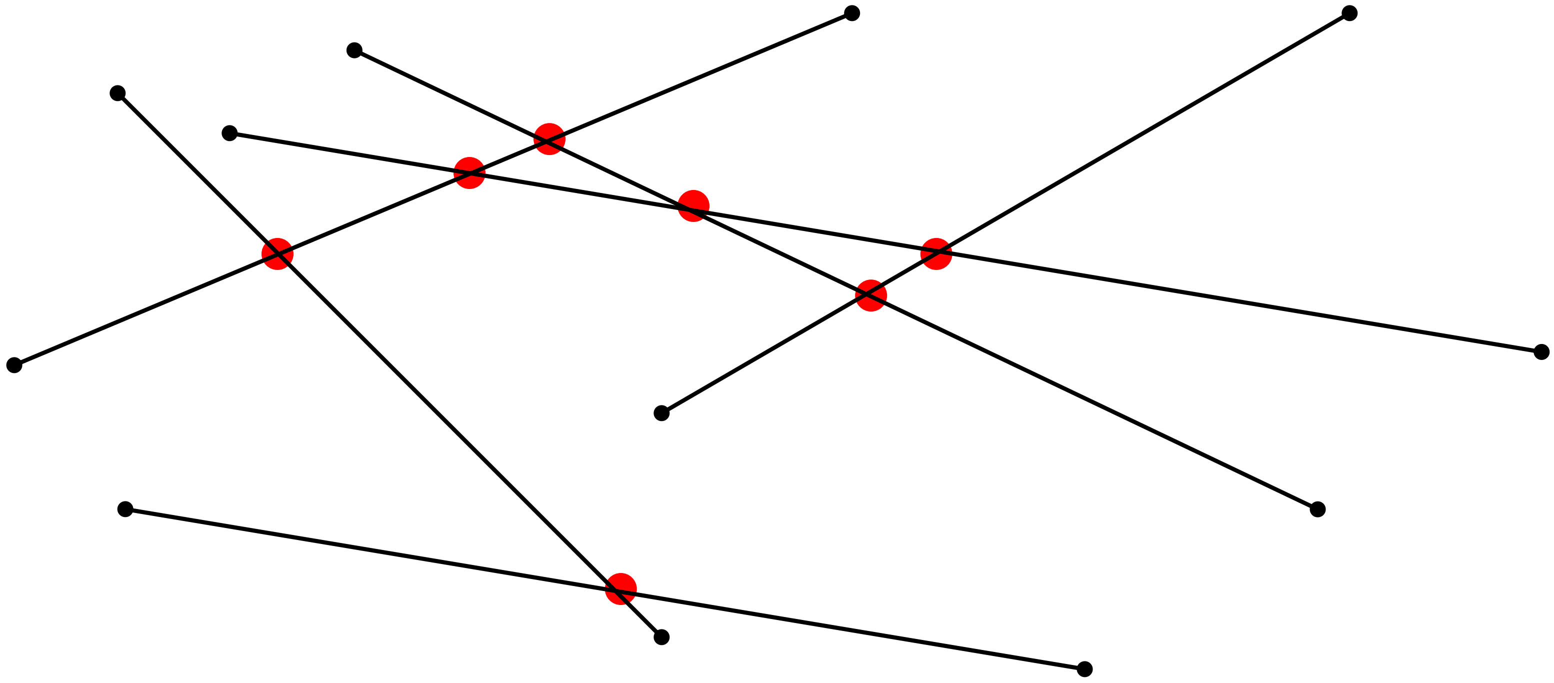
Line Segment Intersections

- Given a set of line segments, each defined by two end points, find all intersecting line segments

Line Segment Intersection



Line Segment Intersection

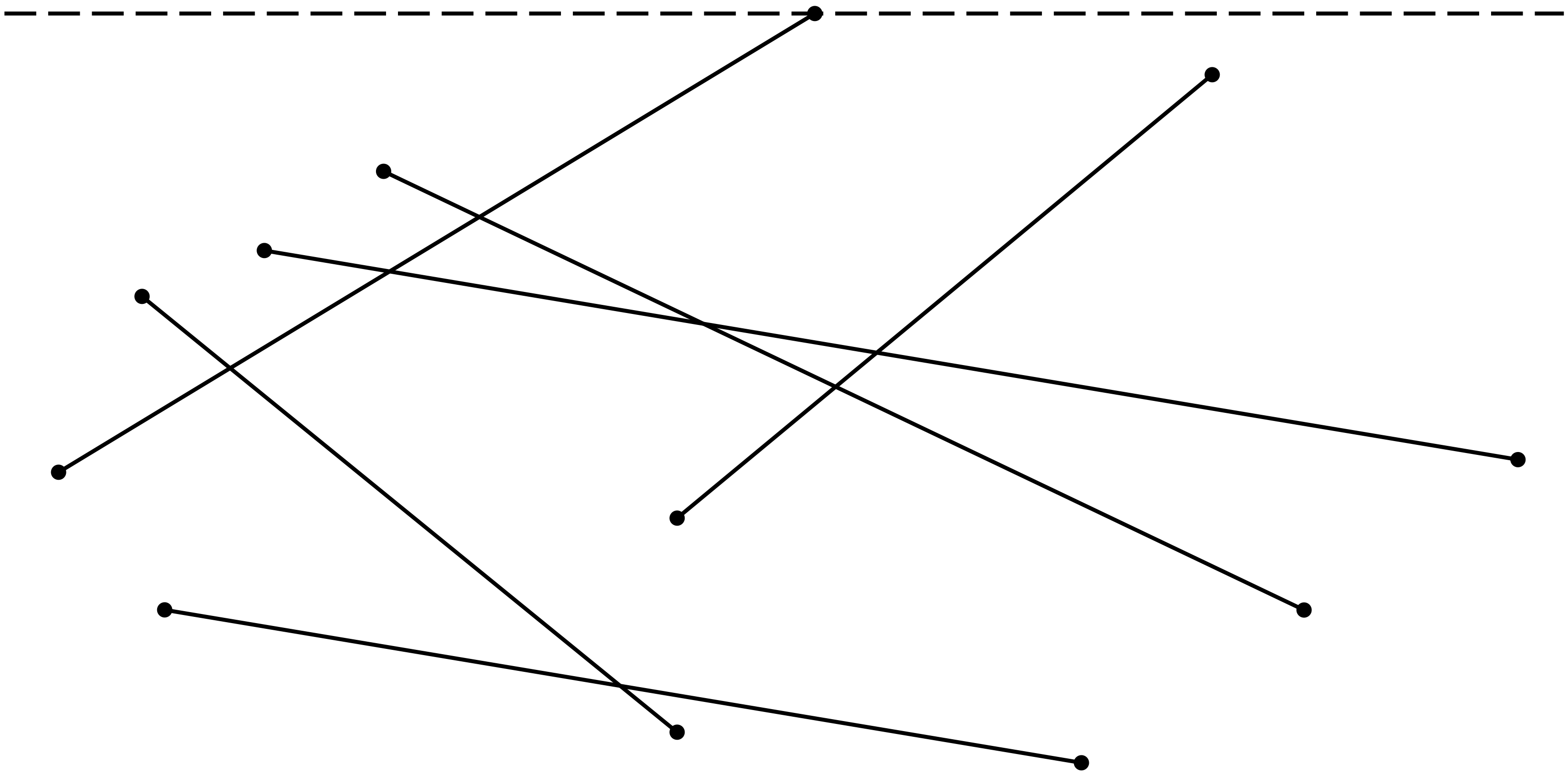


Naïve Algorithm

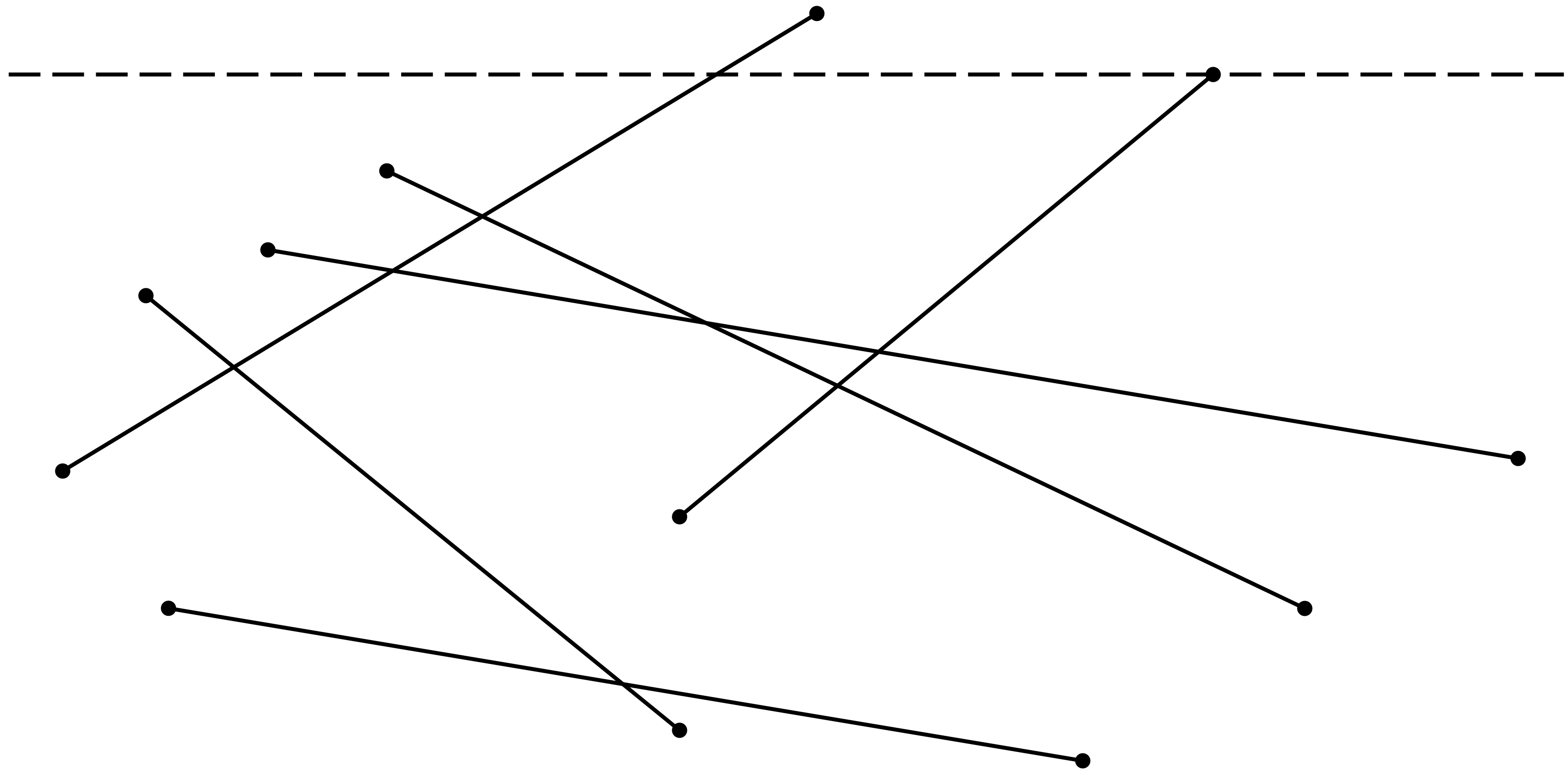
- Enumerate all possible pairs of lines
- Test for intersection

- Running time $O(n^2)$
- What is the lower bound of the running time?
- Worst case: $O(n^2)$
- Is this optimal?

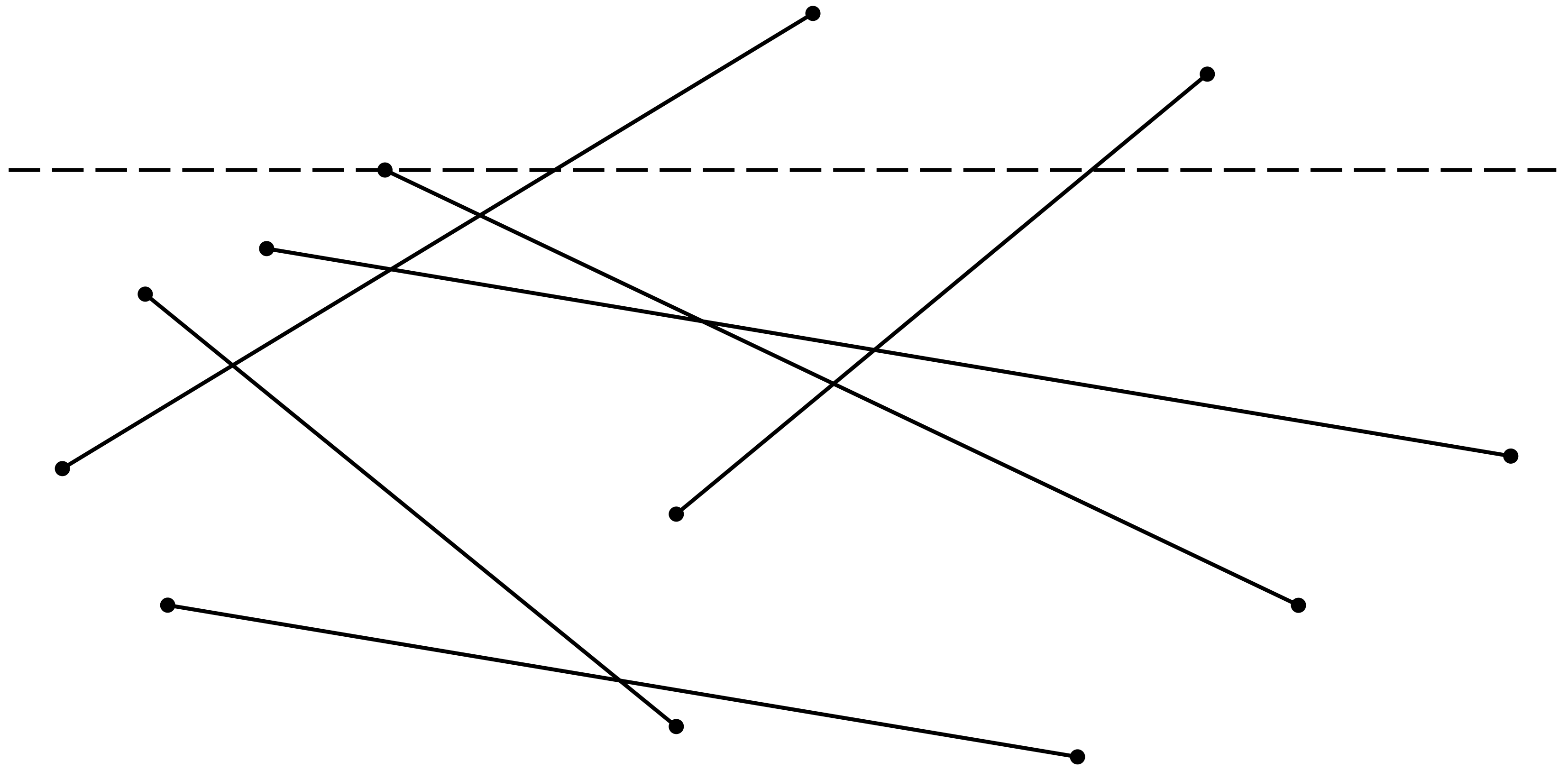
Plane-sweep Algorithm



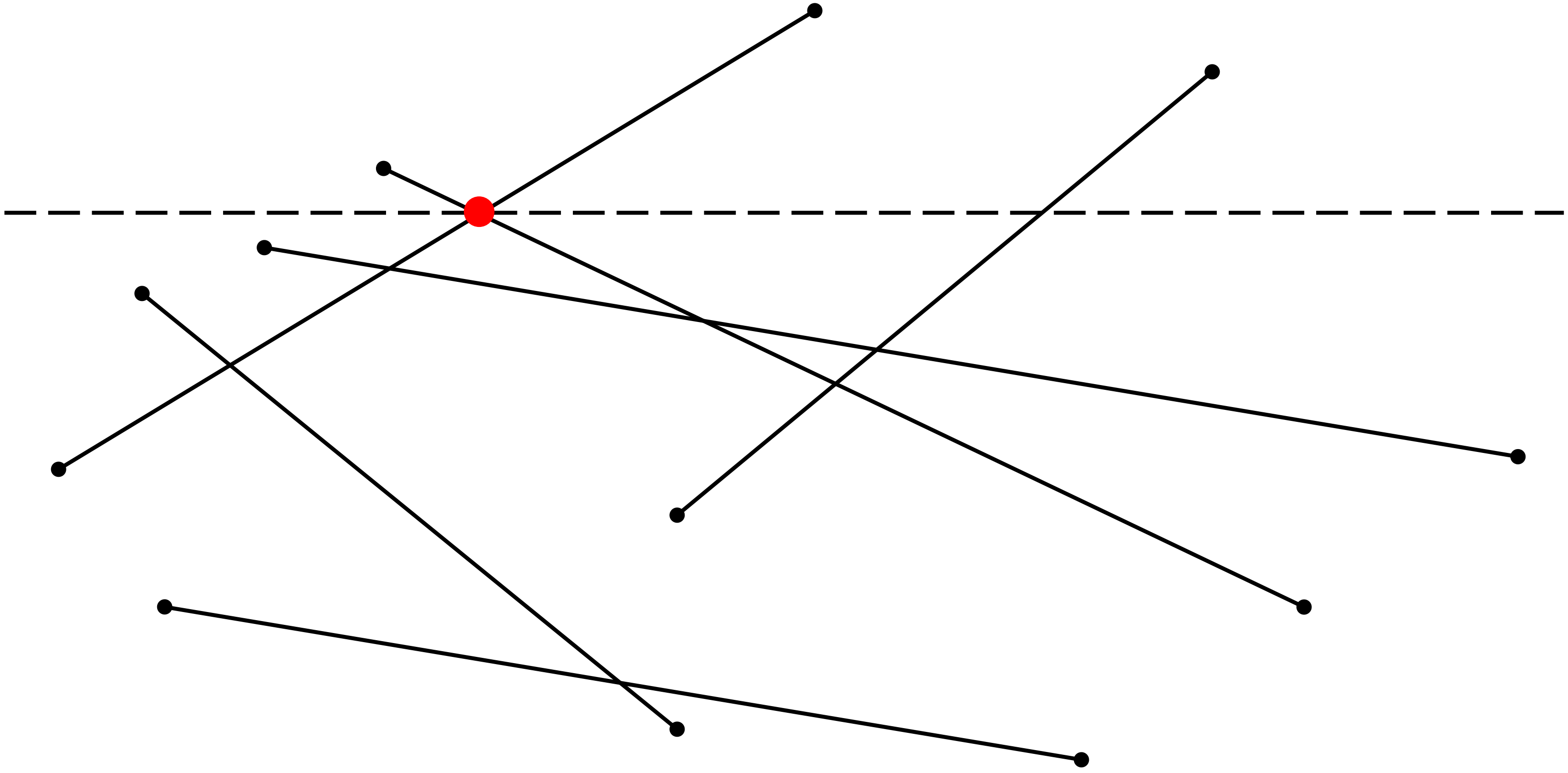
Plane-sweep Algorithm



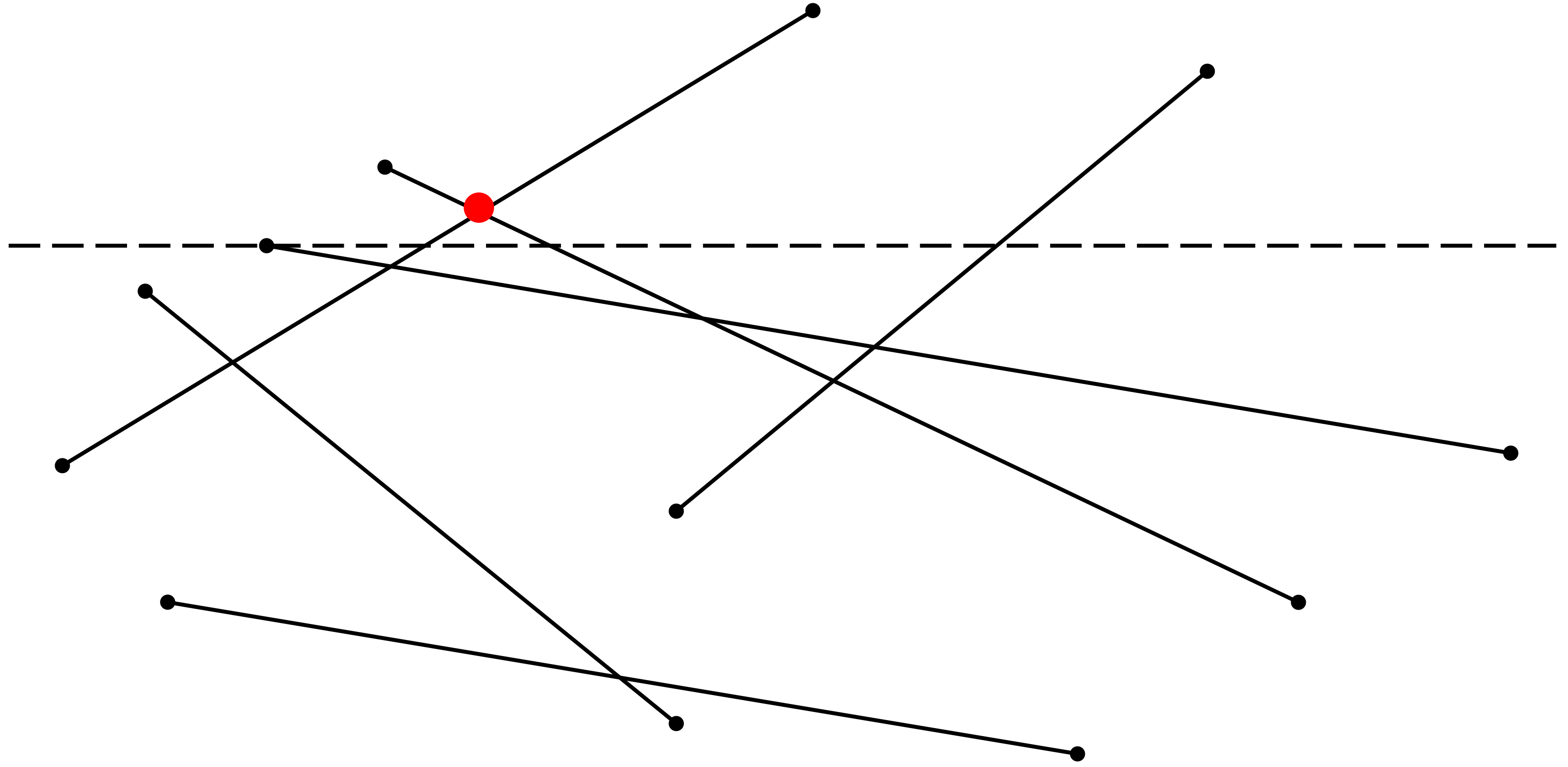
Plane-sweep Algorithm



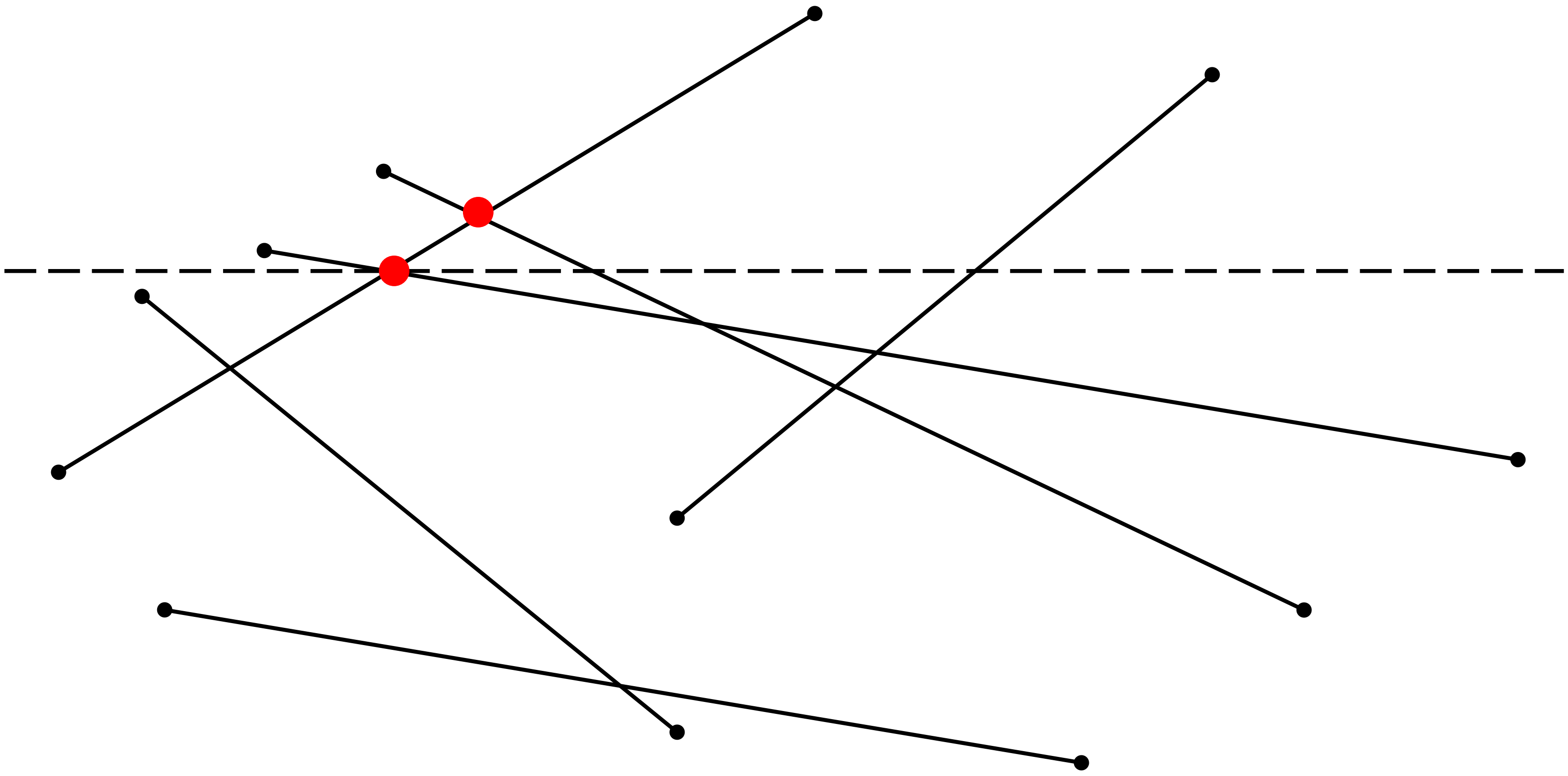
Plane-sweep Algorithm



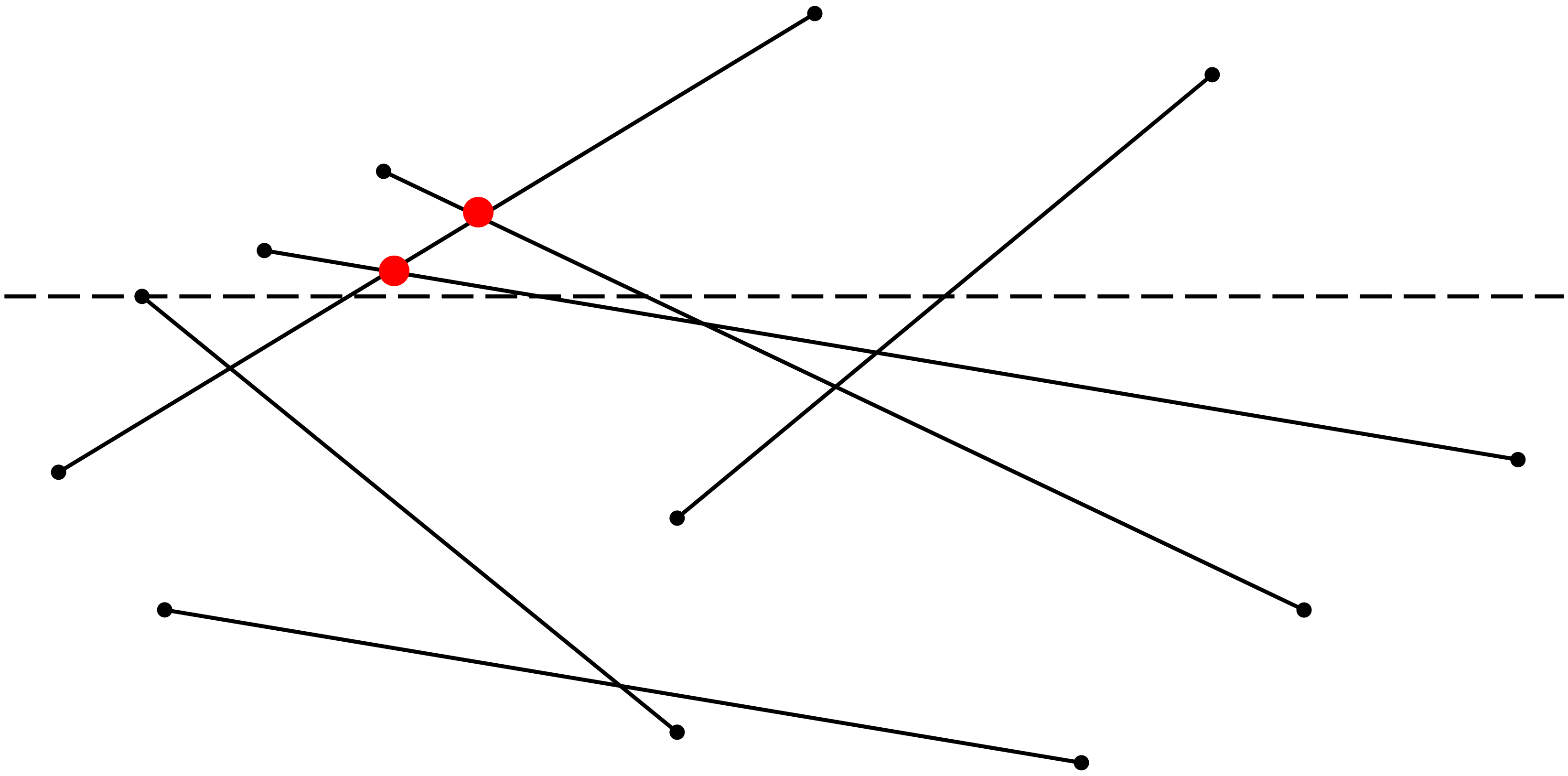
Plane-sweep Algorithm



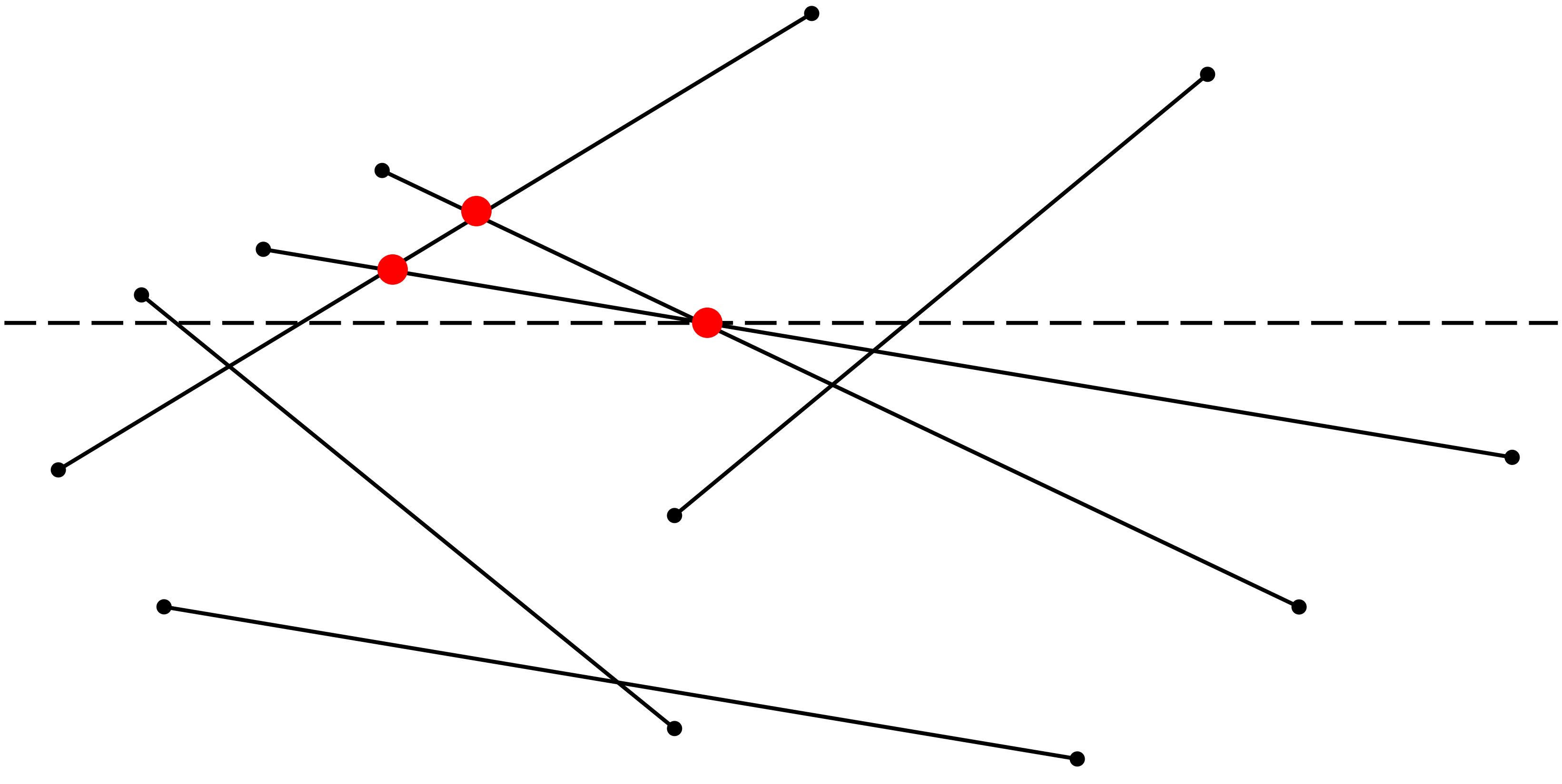
Plane-sweep Algorithm



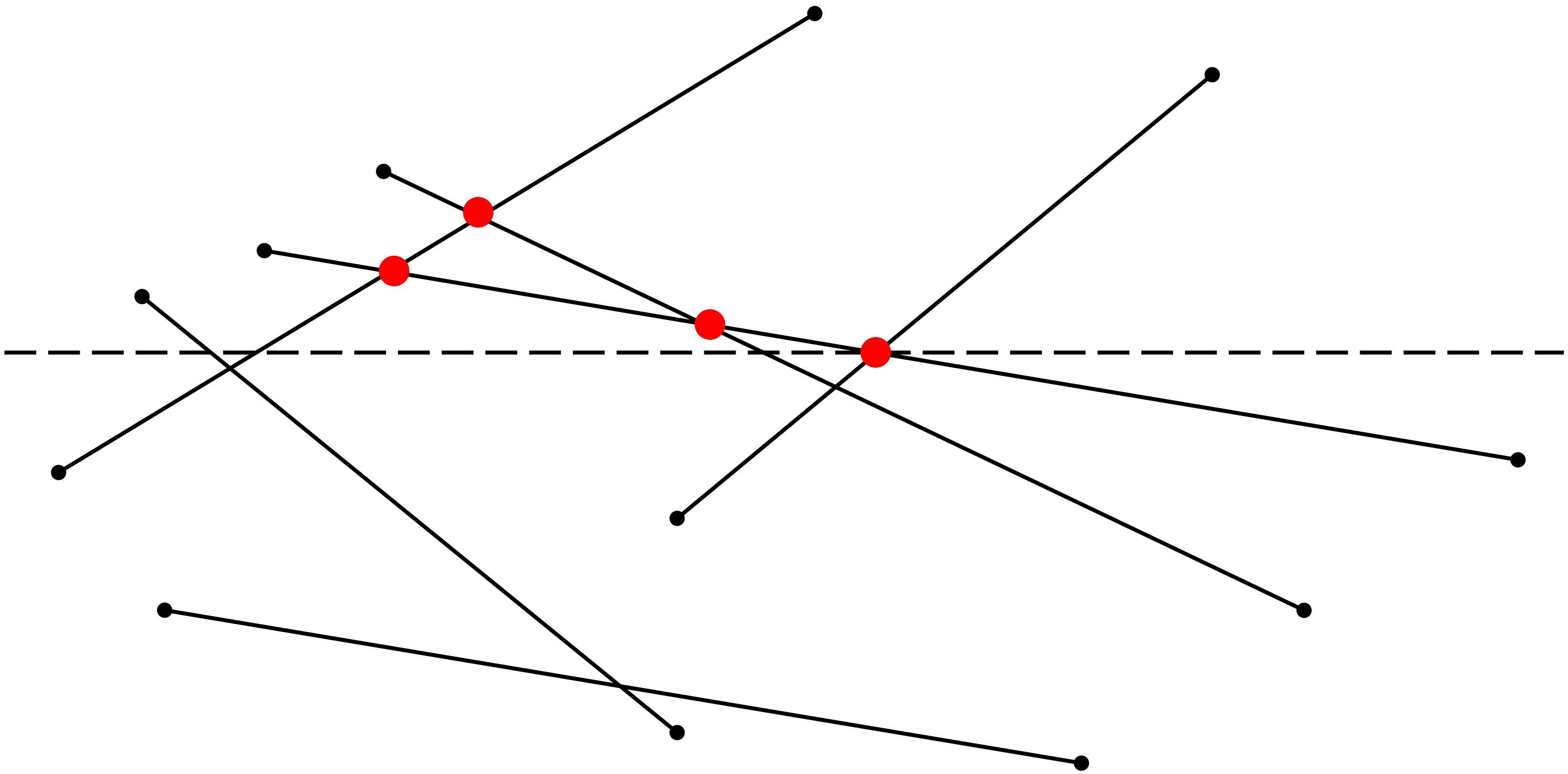
Plane-sweep Algorithm



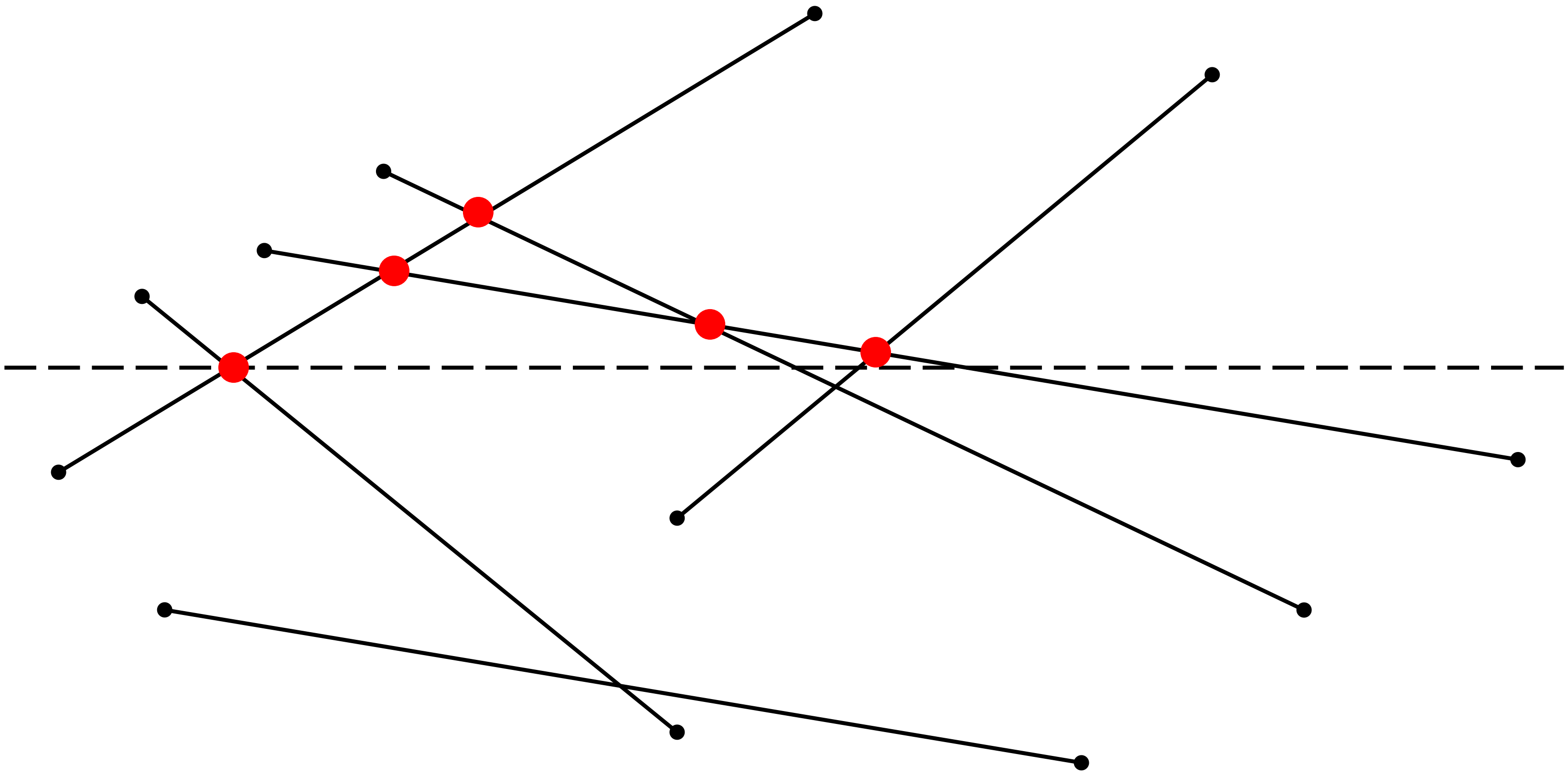
Plane-sweep Algorithm



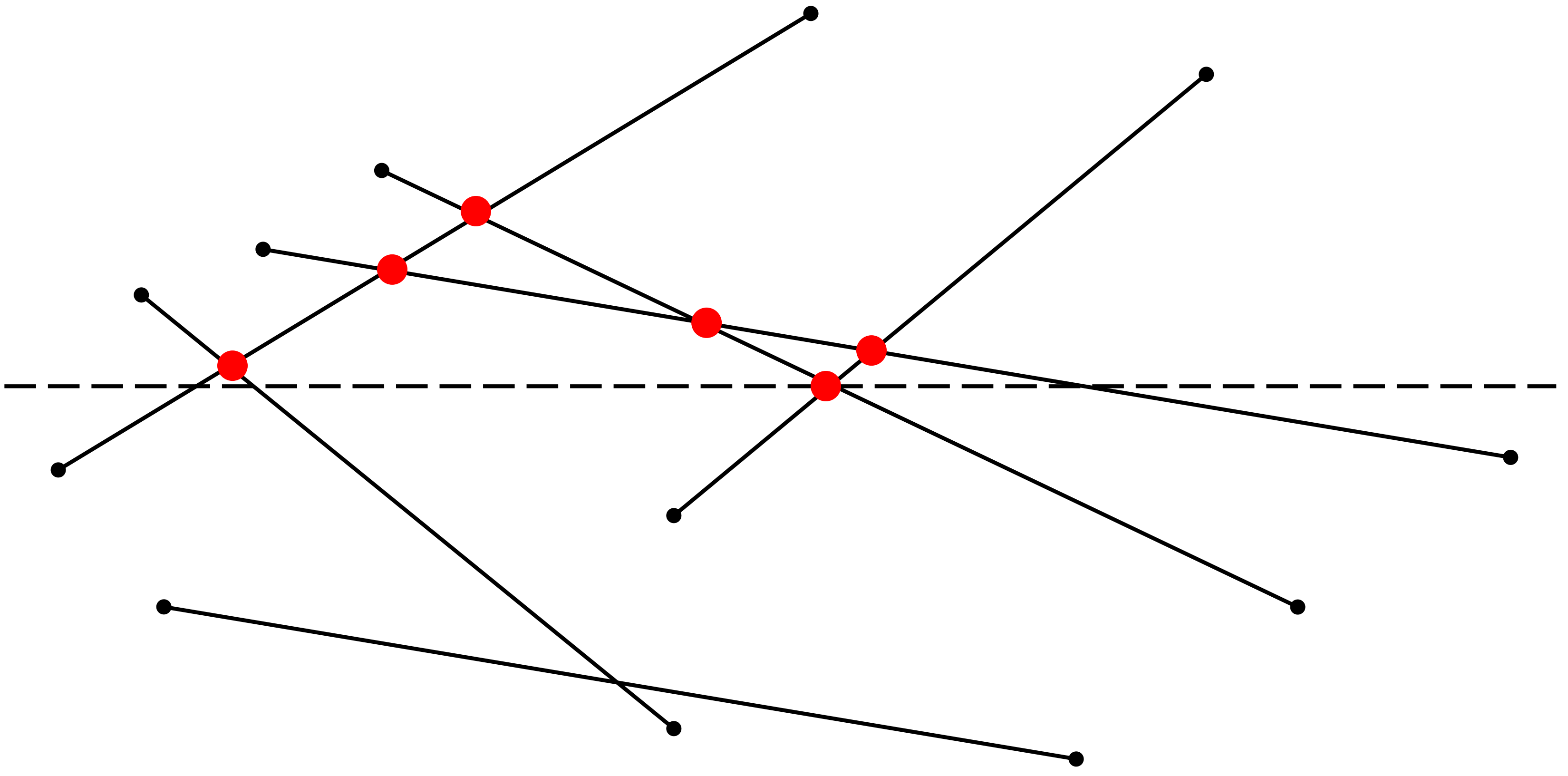
Plane-sweep Algorithm



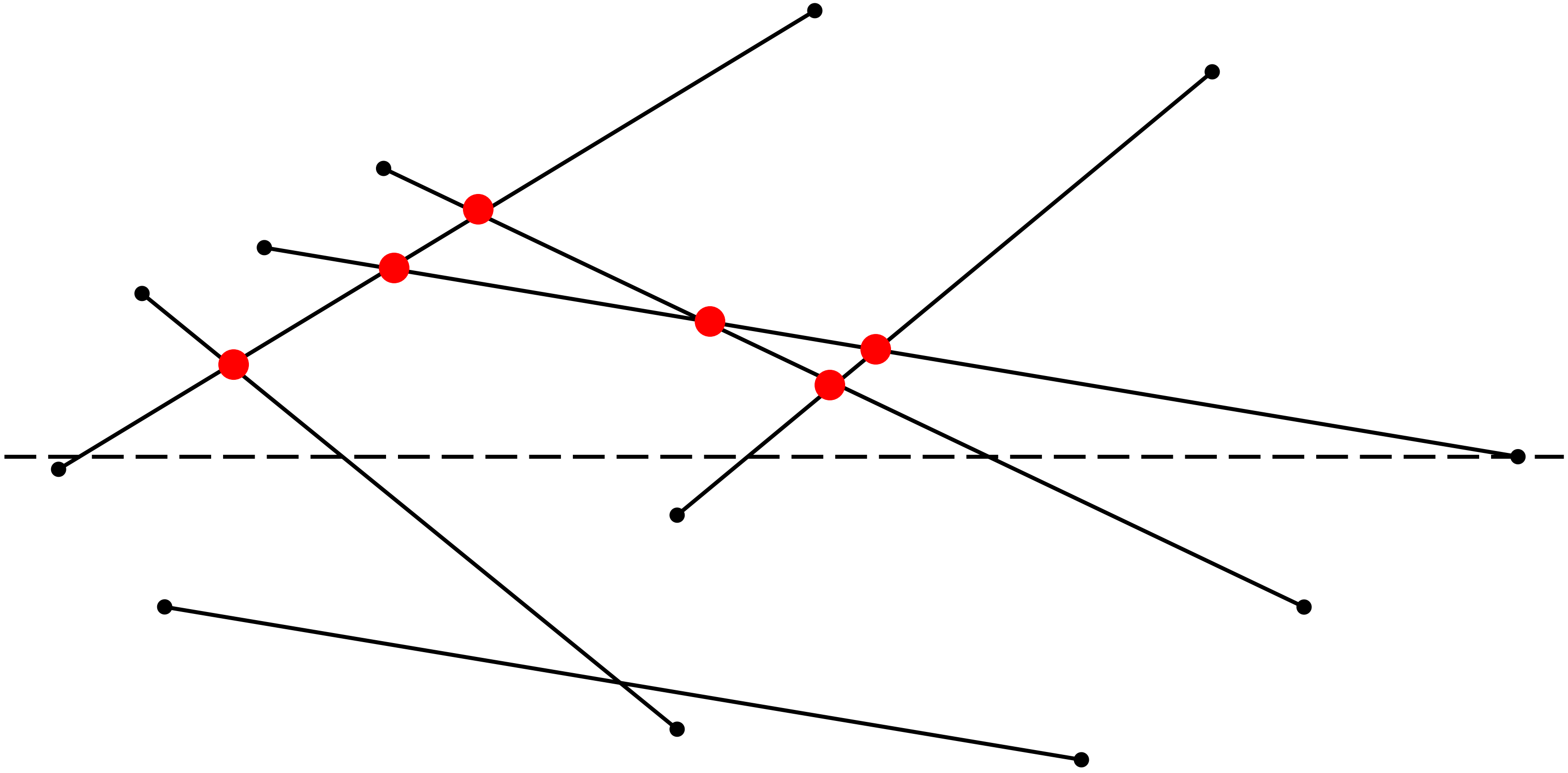
Plane-sweep Algorithm



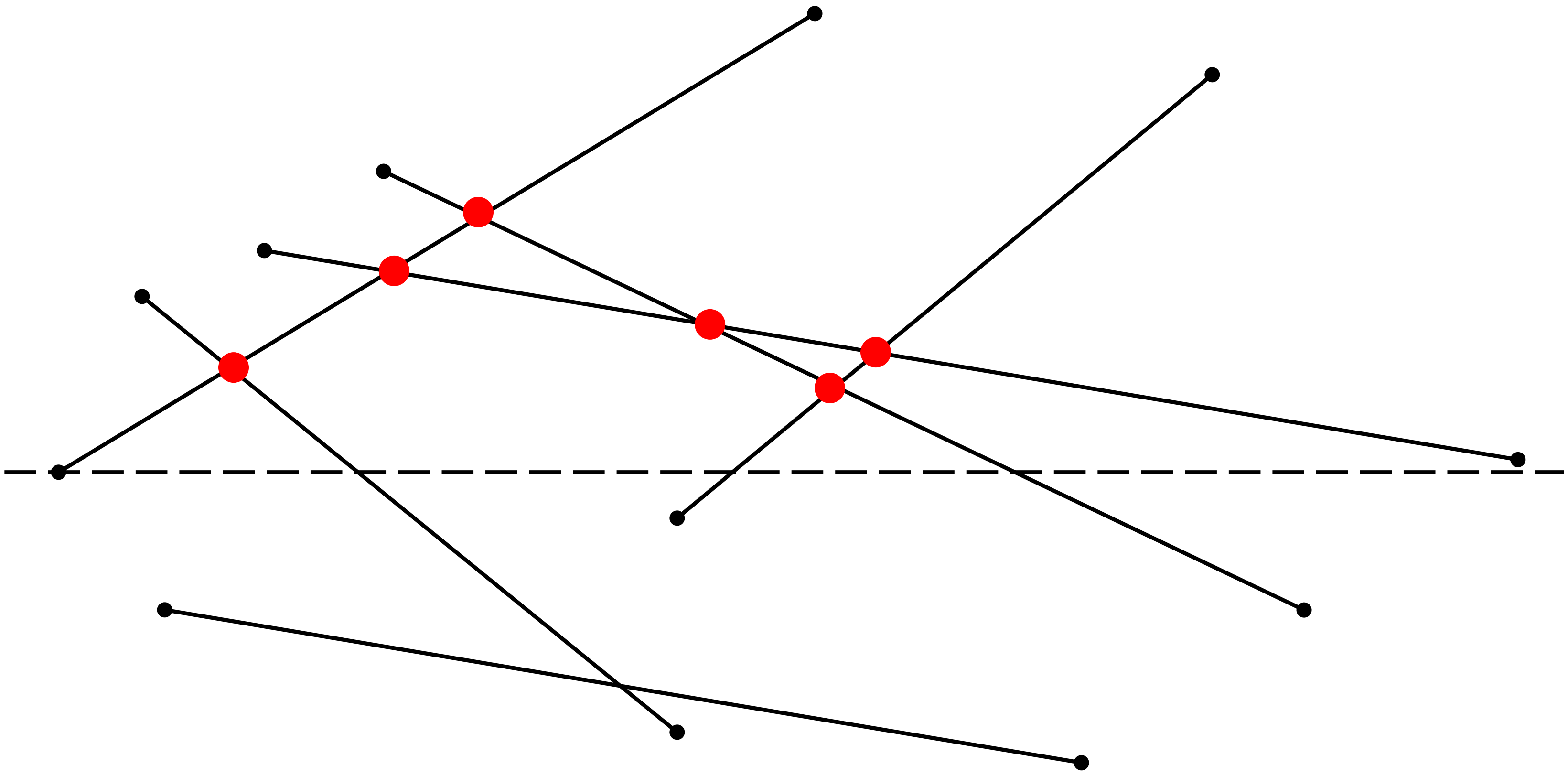
Plane-sweep Algorithm



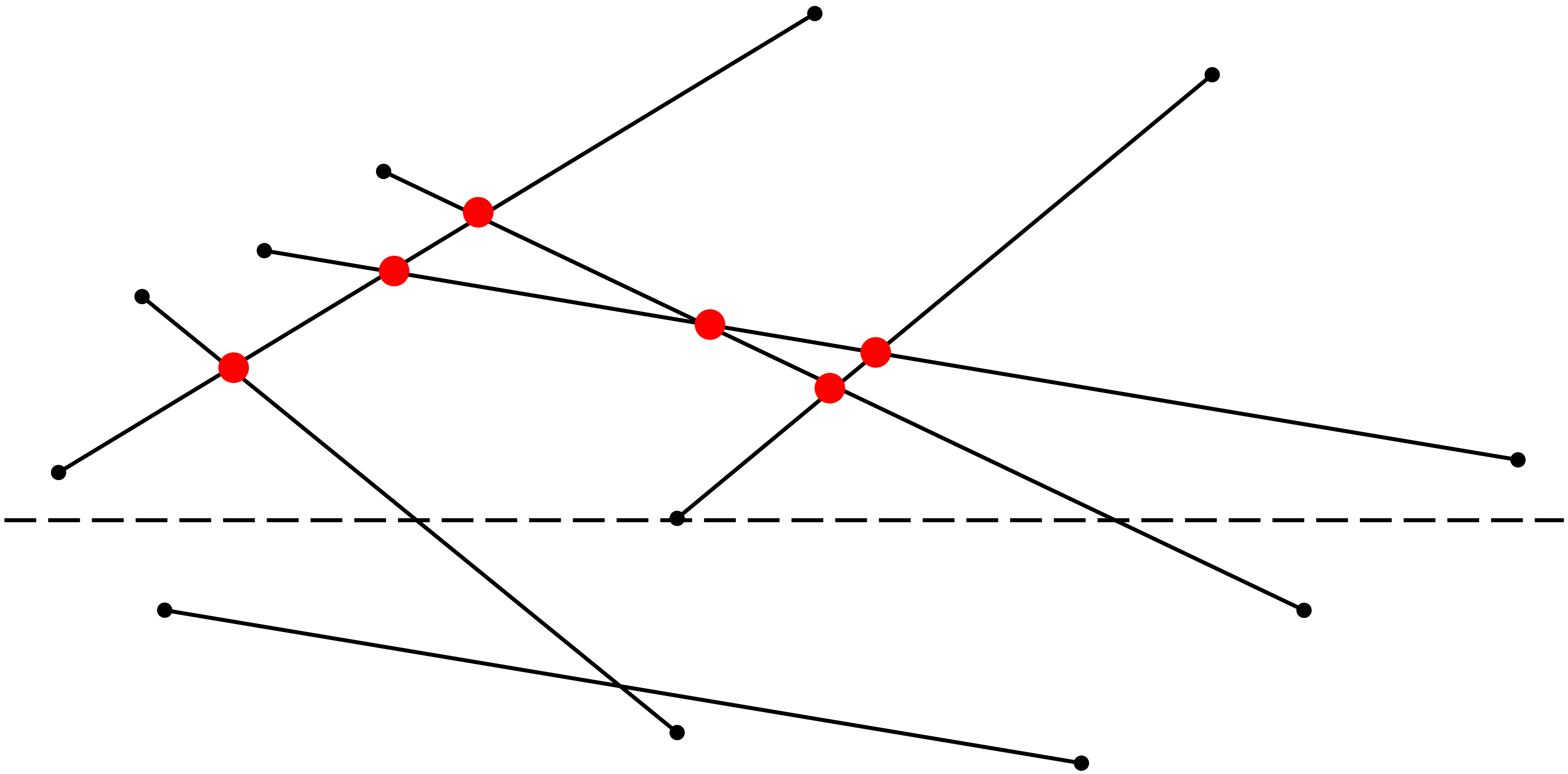
Plane-sweep Algorithm



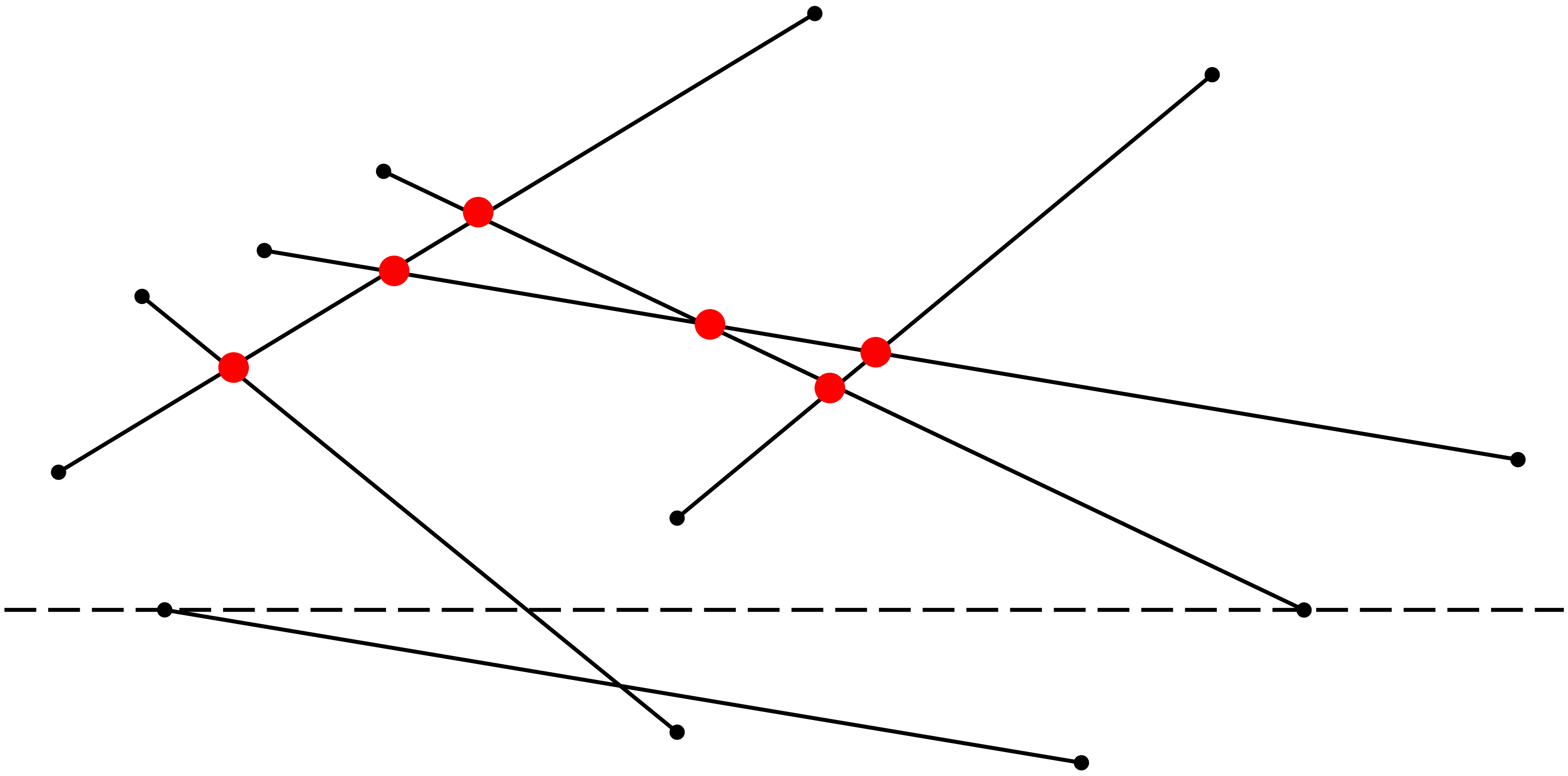
Plane-sweep Algorithm



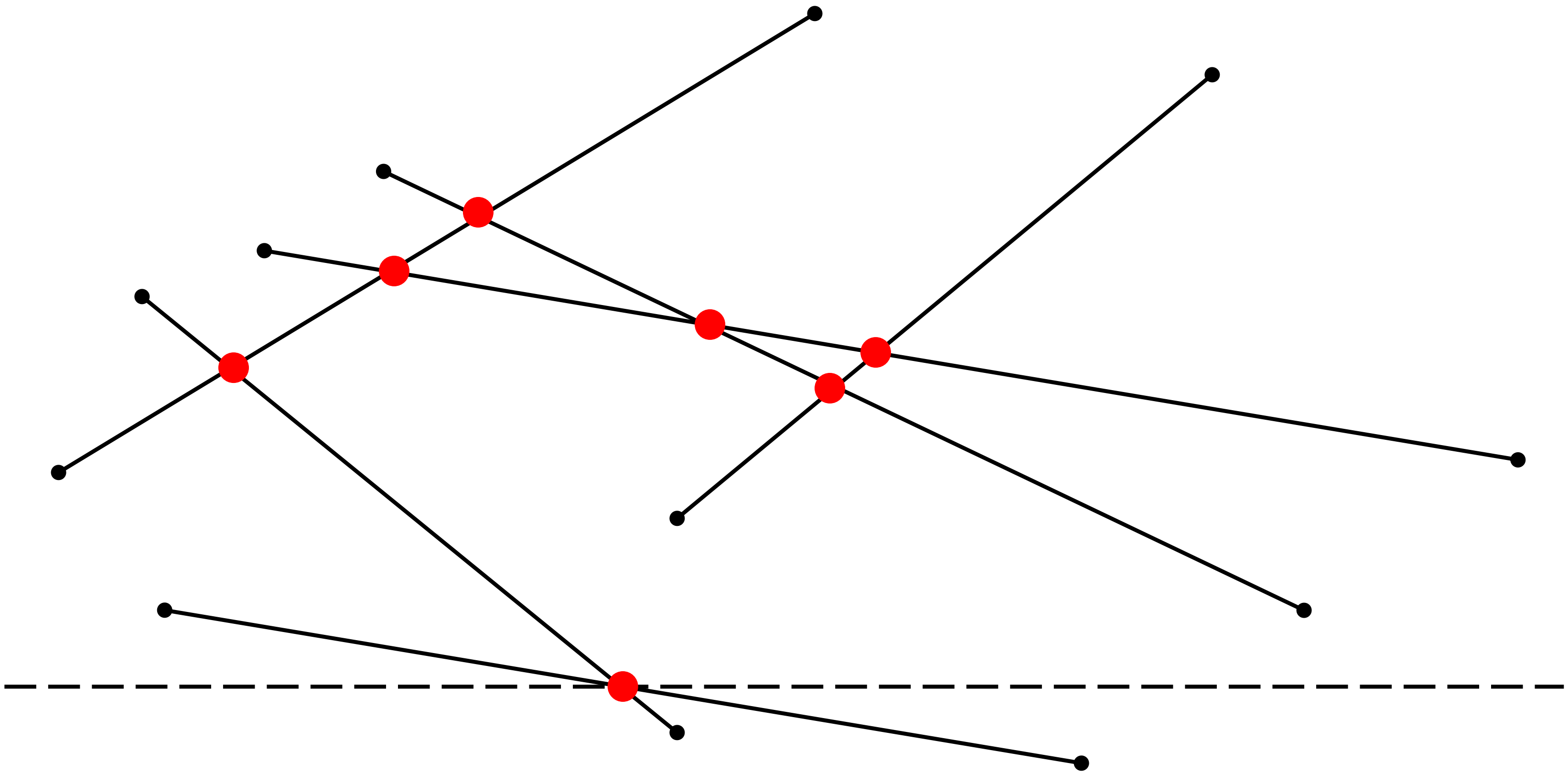
Plane-sweep Algorithm



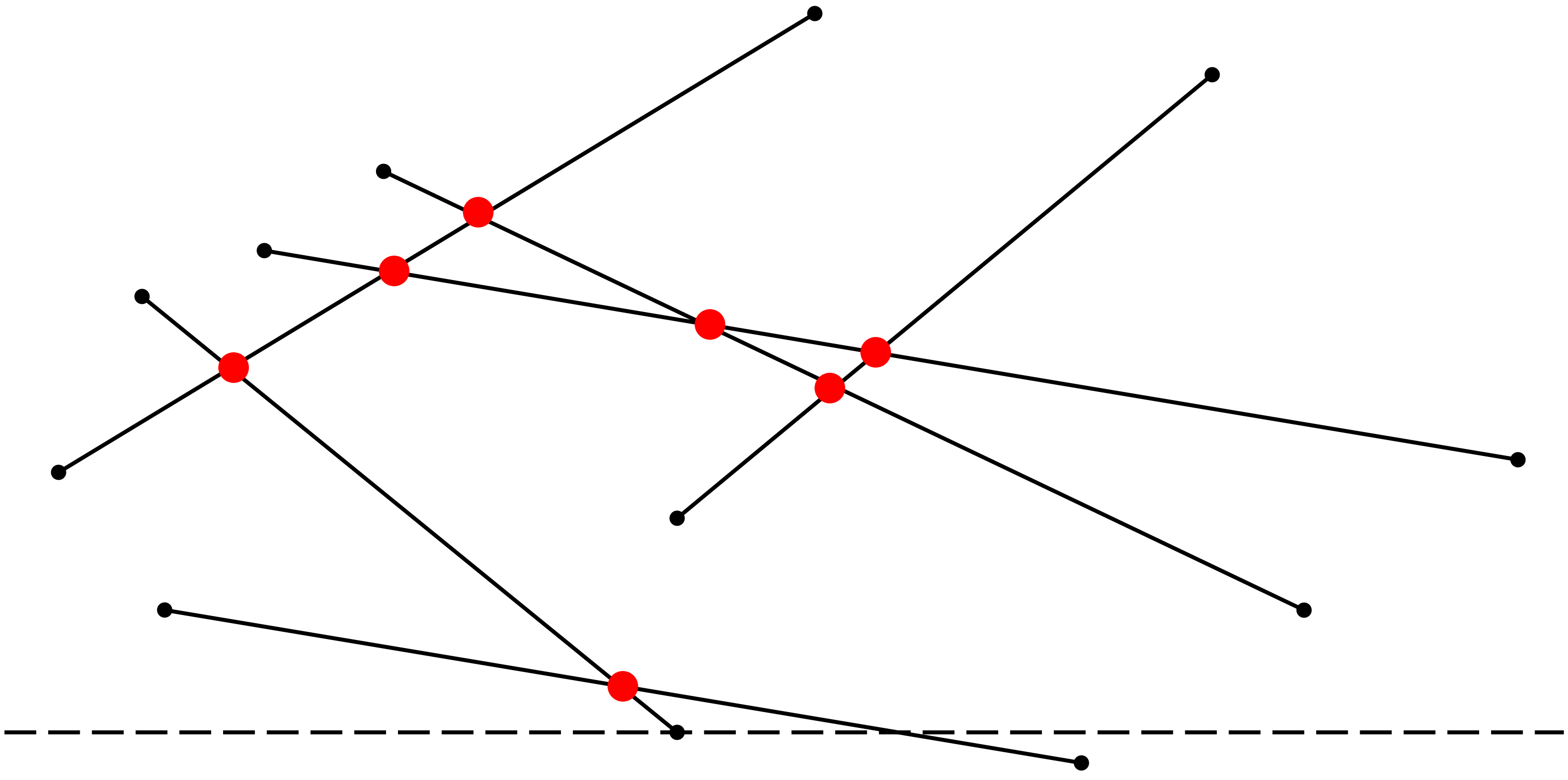
Plane-sweep Algorithm



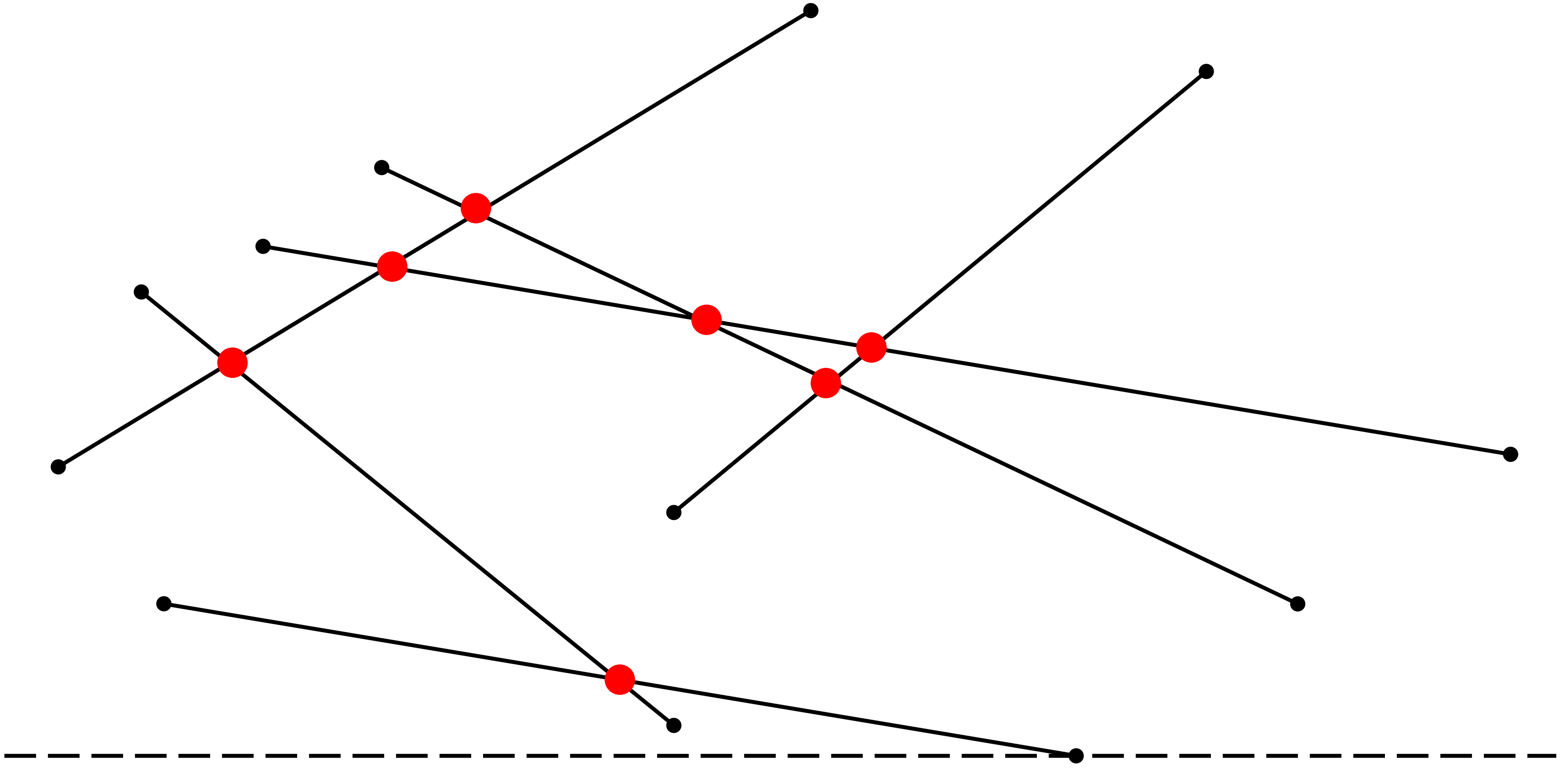
Plane-sweep Algorithm



Plane-sweep Algorithm



Plane-sweep Algorithm

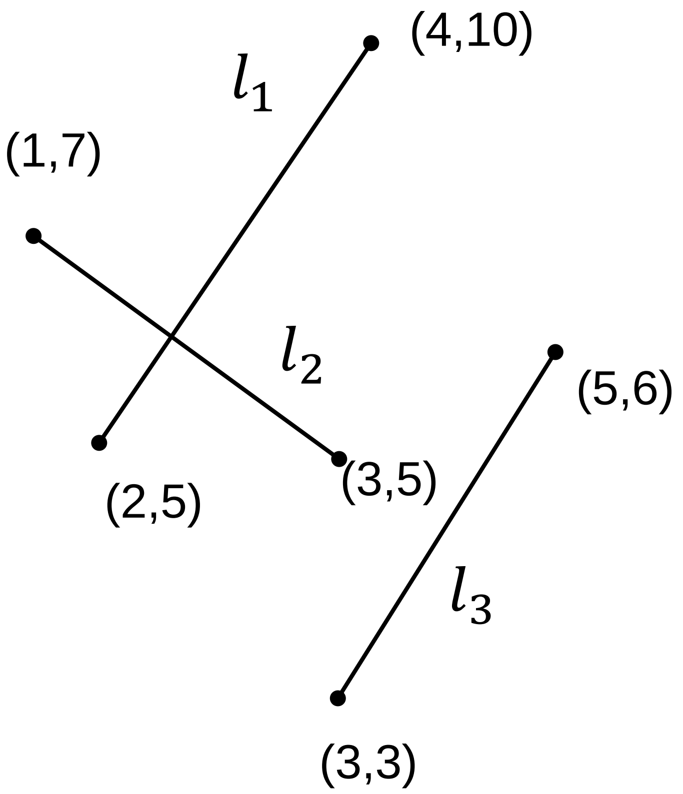


Elements of Plane-sweep

- The sweep line: Sweeps the plane in specific direction, e.g., top-down
- The state of the sweep line S: A set of all line segments that intersect the sweep line at any position. The state changes as the line move.
- The event points E: Is the set of points where the state S changes. In this case, the end points of the line segments comprise the event points.

Example: Event Points

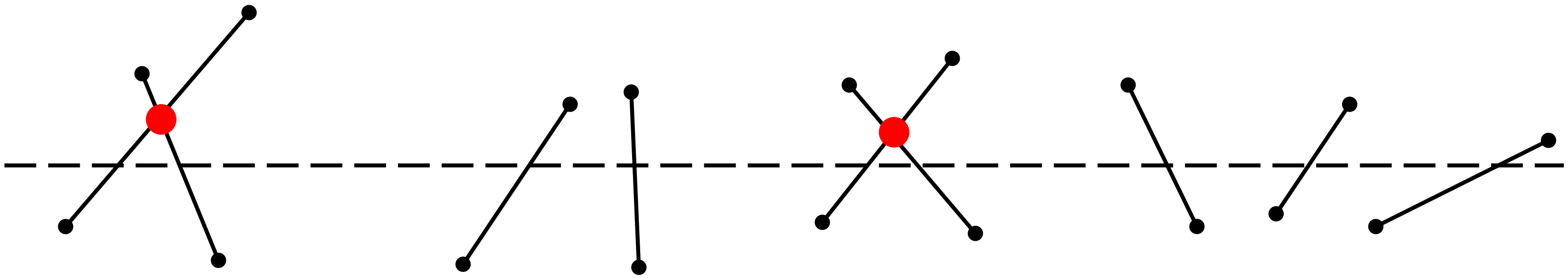
y	l_i	Start/End
10	l_1	Start
7	l_2	Start
6	l_3	Start
5	l_1	End
5	l_2	End
3	l_3	End



Plane-sweep Simple Impl.

- Input $L = \{l_i\}$
- $l_i = (l_i.p_1, l_i.p_2)$
- E= the list of y-coordinates sorted in decreasing order
- $S = \{\}$
- For each event with a corresponding line l_i
  - If top point
    - Compare l_i to each $s \in S$
    - Insert l_i to S
  - If end point
    - Remove l_i from S

Plane-sweep Poor Behavior

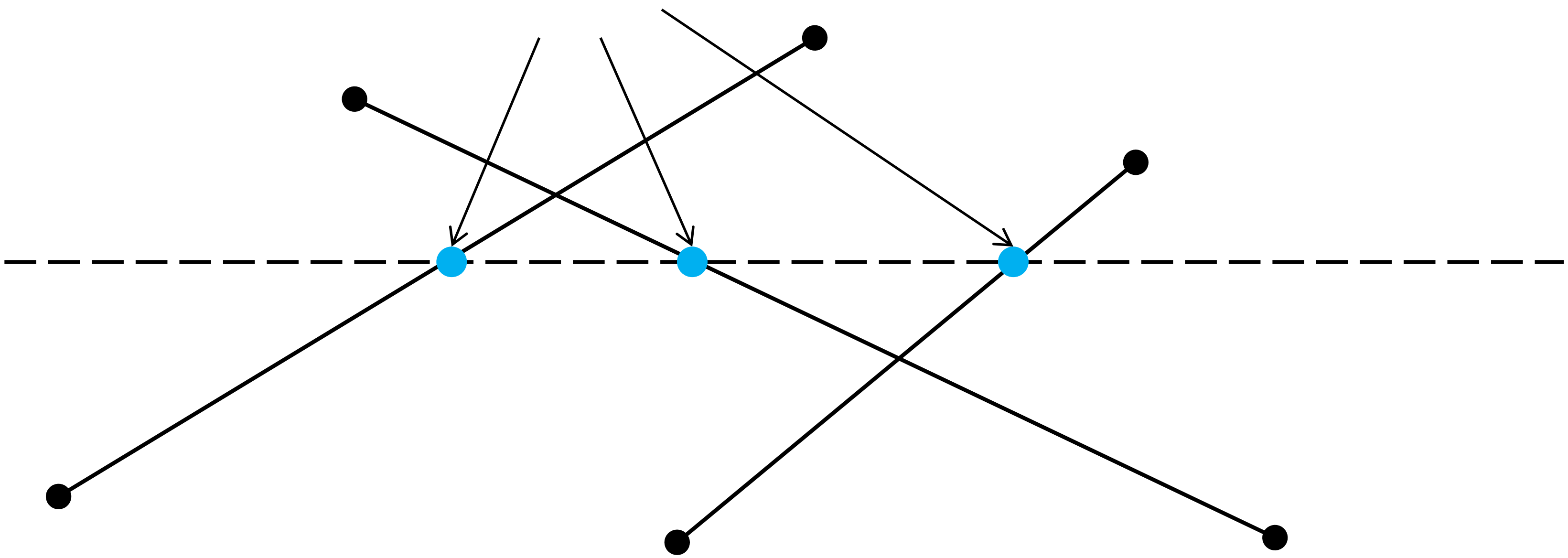


$O(n^2)$

Bentley-Ottmann Algorithm

- An improved scan-line algorithm
- Maintains the state S in a sorted order to speed up checking a line segment against segments in S

Example



Algorithm Pseudo code

- Create a list of event points P
  - P is always sorted by the y coordinate
- Initialize the state S to an empty list
- Initialize P with the first point (top point) of each line segment
- While P is not empty
  - $p \leftarrow P$.pop
  - $y_s = p.y$
  - processEvent(p)

Process Event Point (p)

- *// p is the top (starting) point*
- **If** p is the top point
  - Add $p.l$ to S at the order $p.x$ ($p.l = Si$)
  - checkIntersection(S_{i-1}, S_i)
  - checkIntersection(S_i, S_{i+1})
  - Add the end point of $p.l$ to P

Process Event Point (p)

- *// p is the bottom (ending) point*
- **If** p is the bottom point
  - *// let $p.l$ be at position S_i before removal*
  - Remove $p.l$ from S
  - checkIntersection(S_{i-1}, S_i)

Process Event Point (p)

- **If** p is an interior point
  - Report p as an intersection
  - Find $p.l$ in S ($p.l = \{S_i, S_{i+1}\}$)
  - Swap(S_i, S_{i+1})
  - checkIntersection(S_{i-1}, S_i)
  - checkIntersection(S_{i+1}, S_{i+2})

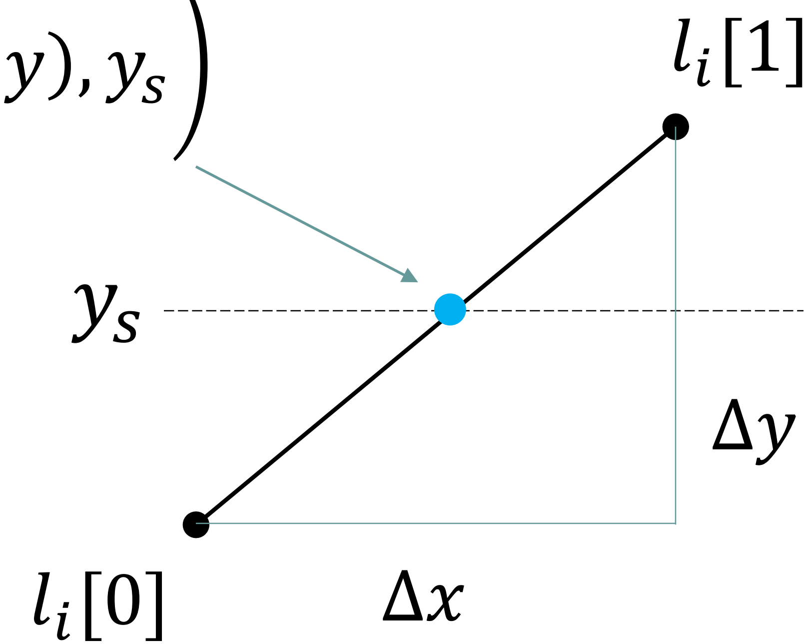
Check Intersection(l_1, l_2)

- If l_1 does not intersect l_2 **then return**
- Compute the intersection p_i of l_1 and l_2
- If $p_i.y$ is above the sweep line **then return**
- If $p_i \in P$ **then return**
- Insert p_i into P

Sweep Line State (S)

- A list of line segments $[l_i]$
- Sorted by the x-coordinate of the intersections between l_i and the sweep line

$$\left(l_i[0].x + \frac{\Delta x}{\Delta y}(y_s - l_i[0].y), y_s\right)$$

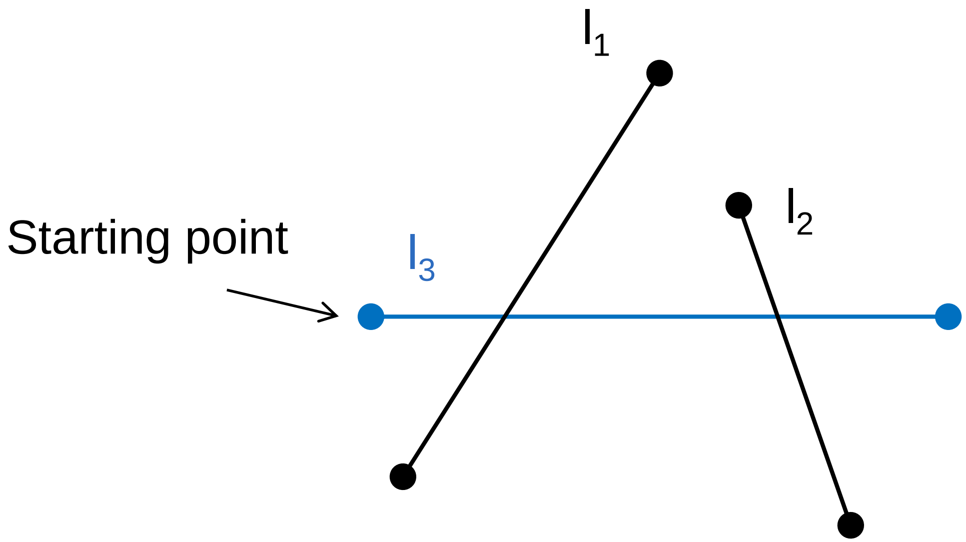


Analysis

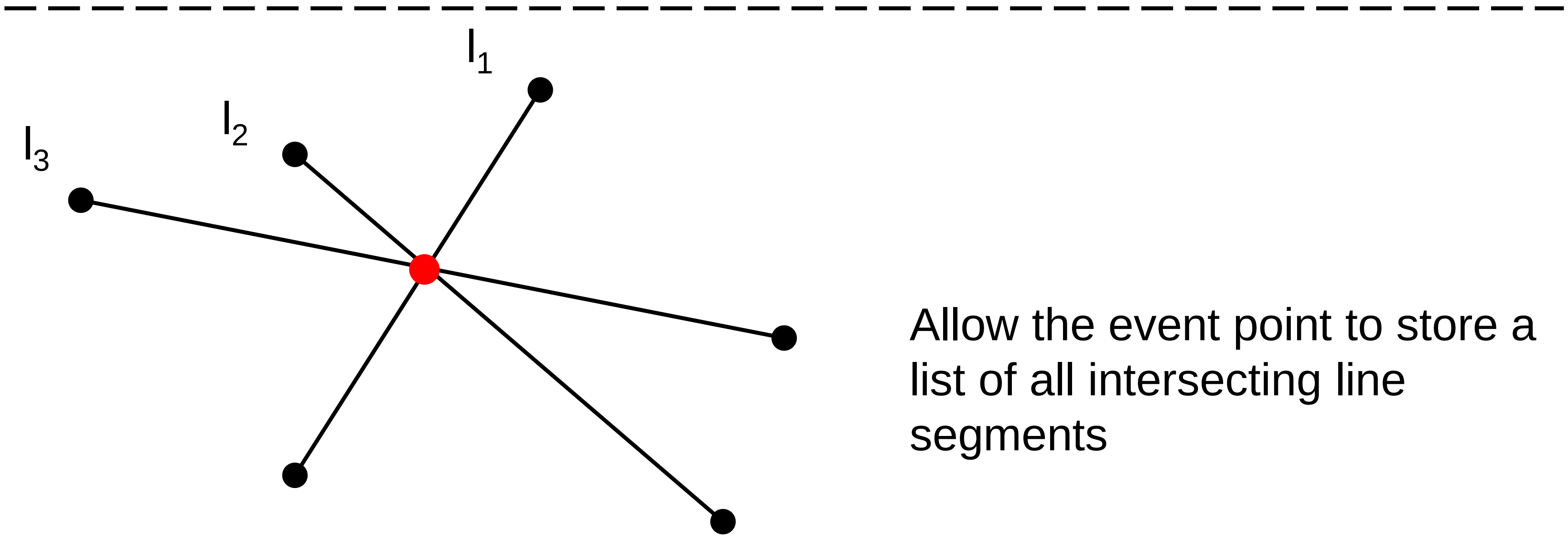
- Initial sort of starting points $O(n \cdot \log n)$
- Number of processed event points $(2n + k)$
- For each event point
  - Remove from P: $O(\log|P|)$
  - Insert or remove from S: $O(\log|S|)$
  - Check intersection with at most two lines: $O(1)$
  - Insert a new event points: $O(\log|P|)$
- Upper limit of $|P| = 2n$
- Upper limit of $|S| = n$
- Overall running time $O\big((n + k) \log n\big)$

Corner Case 1: Horizontal Line

If two points have the same y-coordinate, sort them by the x-coordinate



Corner Case 2: Three Intersecting Lines



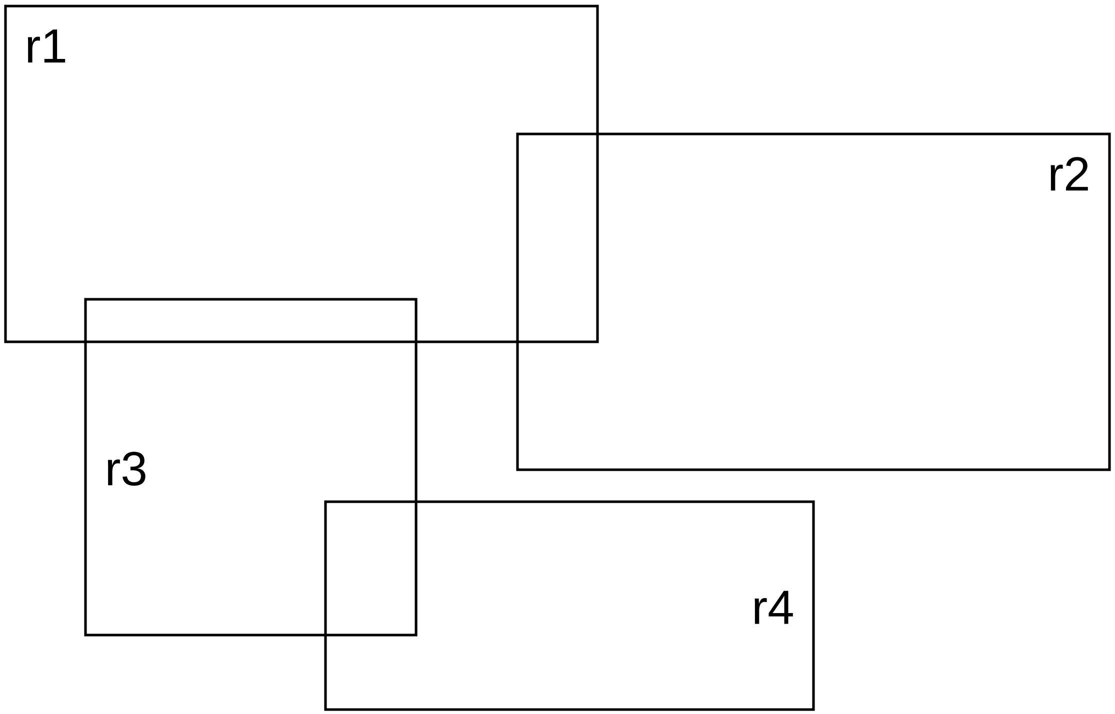
Allow the event point to store a list of all intersecting line segments

When processed, reverse the order of all the lines

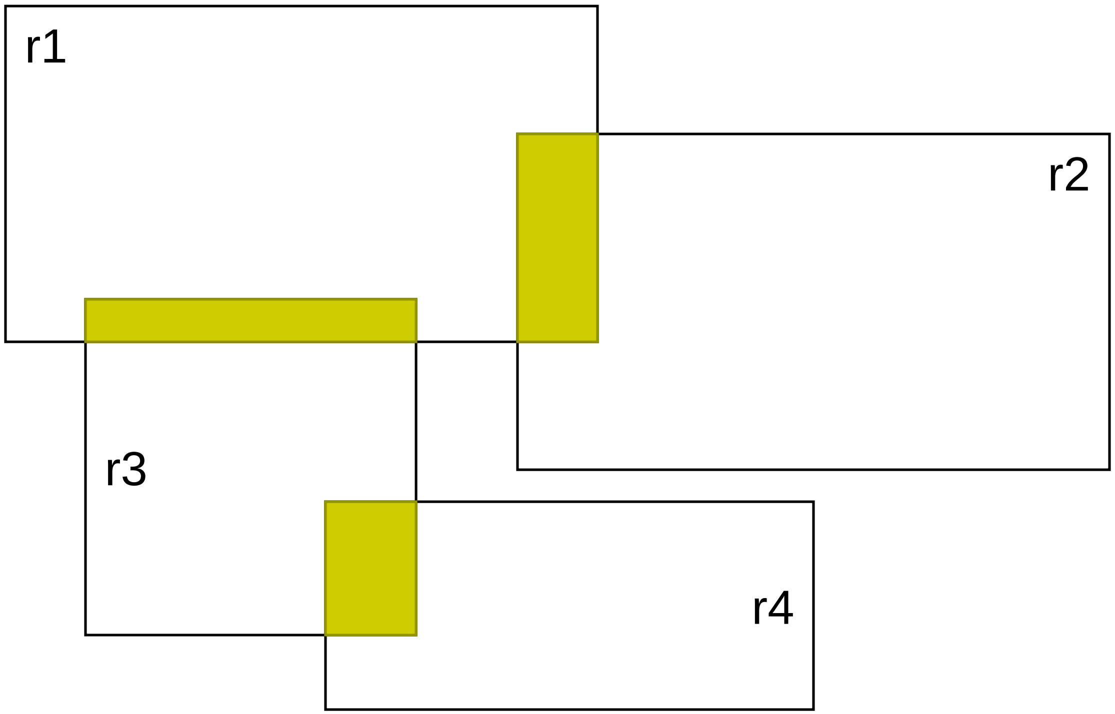
Rectangle Intersection

- Given a set of orthogonal rectangles (R), find the set of all intersections between pairs of rectangles
- $\{r_1 \cap r_2 : r_1, r_2 \in R\}$

Example



Example



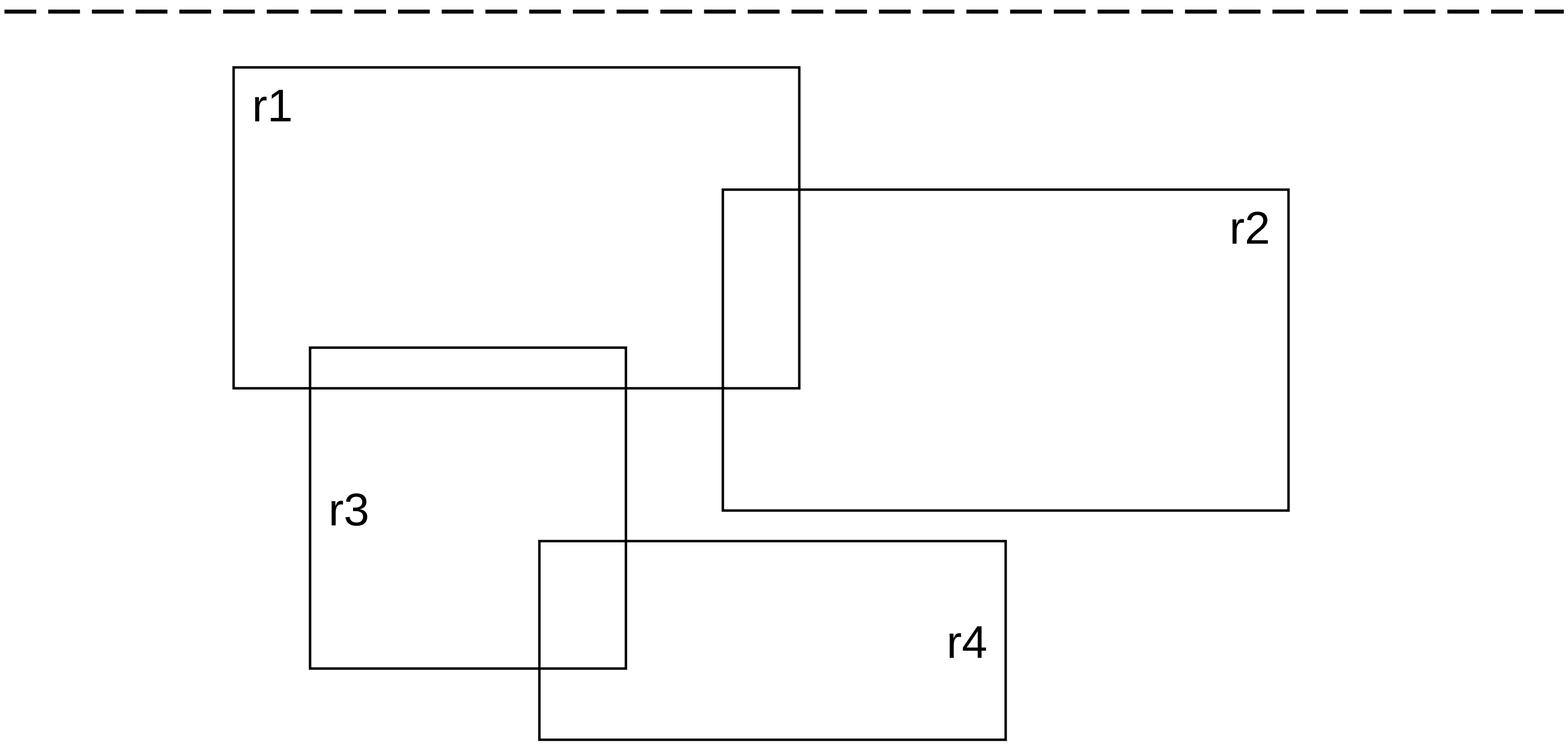
Rectangle Primitives

- An orthogonal rectangle is represented by its two corner points, lower and upper
- Test if two rectangles overlap
  - Two rectangles overlap if both their x intervals and y-intervals overlap
  - Intervals overlap [x1,x2], [x3,x4]: x4>=x1 and x2>=x3
- R1(x1,y1,x2,y2)×R2(x3,y3,x4,y4) ➔ R3(Max(x1,x3), Max(y1,y3), Min(x2,x4), Min(y2,y4))

Naïve Algorithm

- Test all pairs of rectangles and report the intersections
- Running time $O(n^2)$
- Is it optimal?

Simple Plane-sweep Algo.



Simple Plane-sweep Algo.

- What is the state of the sweep line?
- What is an event?
- What processing should be done at each event?

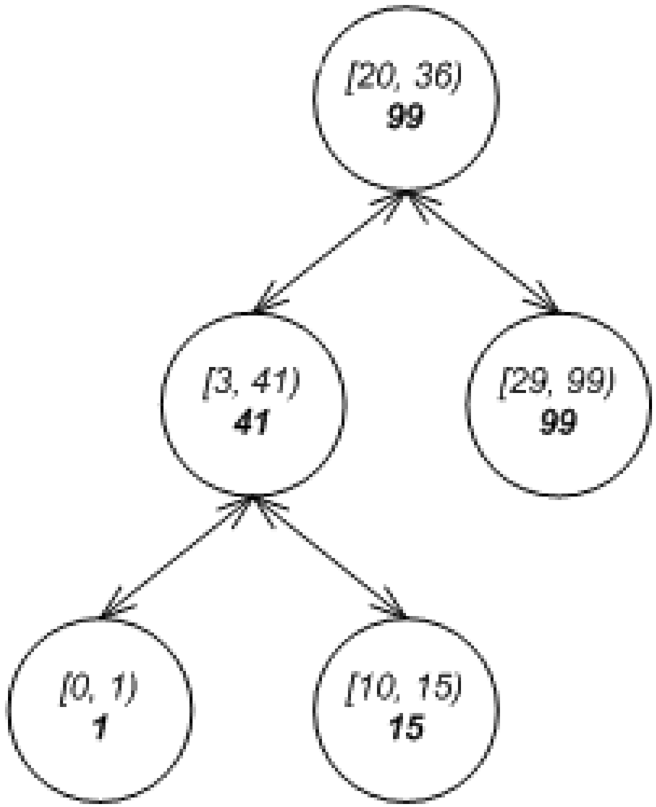
Improved Plane-sweep Algo.

- Keep the sweep line state sorted
  - But how?
- Interval tree
- A variation of BST
- Stores intervals
- Supports two operation
  - Find all intervals that overlap a query point p
  - Find all intervals that overlap a query interval q

A Simple Interval Tree

- Store the intervals in a BST ordered by $i.x_{min}$
- Augment the BST with the value x_{max} which stores the maximum value of all the intervals in the subtree

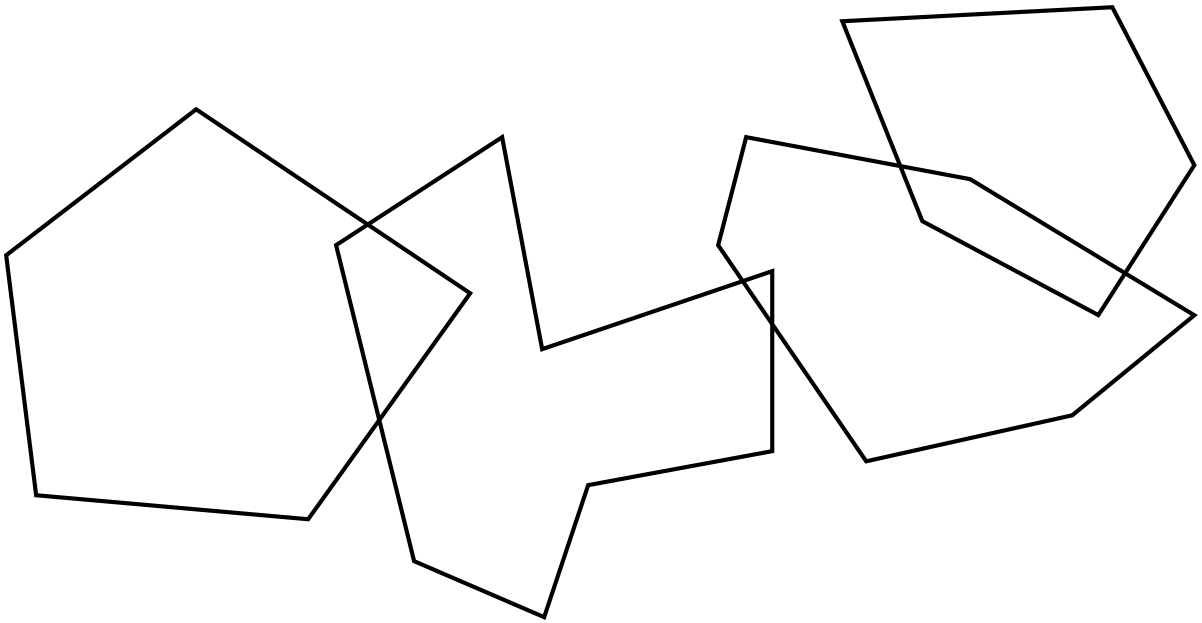
Augmented BST



Credit: https://en.wikipedia.org/wiki/Interval_tree

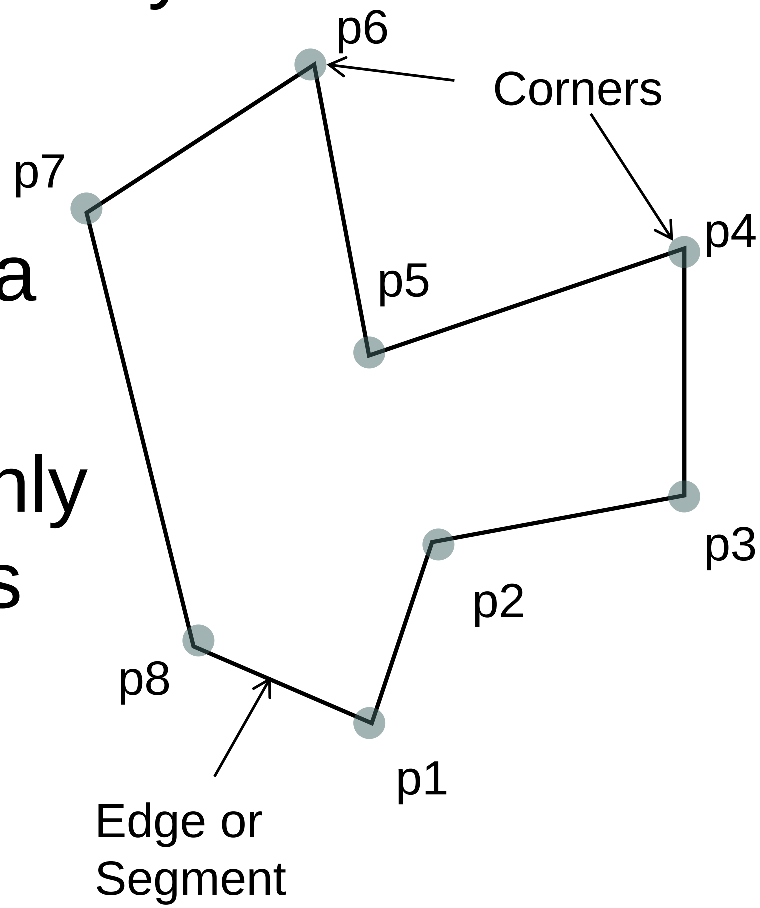
Polygon Intersection

- Given a set of polygons, find all intersecting polygons



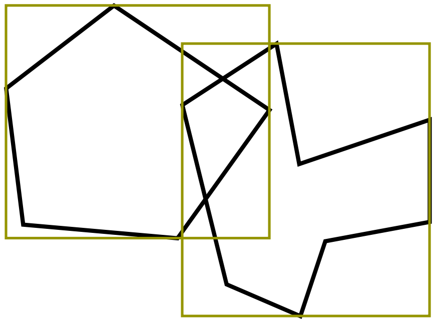
Polygon Representation

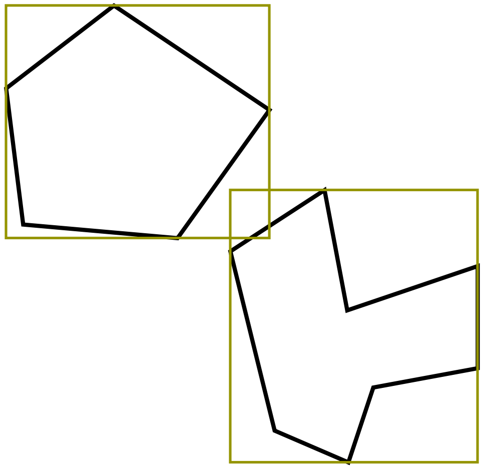
- A polygon is represented as a sequence of points that form its boundary
- A general polygon might also contain holes, e.g., a grass area with a lake inside
- For simplicity, we will only deal with solid polygons with no holes



Filter-and-refine Approach

- Convert all polygons to rectangles
  - For each polygon, compute its minimum bounding rectangle (MBR)
- Filter: Find all overlapping rectangles
- Refine: Test the polygons that have overlapping MBBs





Filter-and-refine Approach

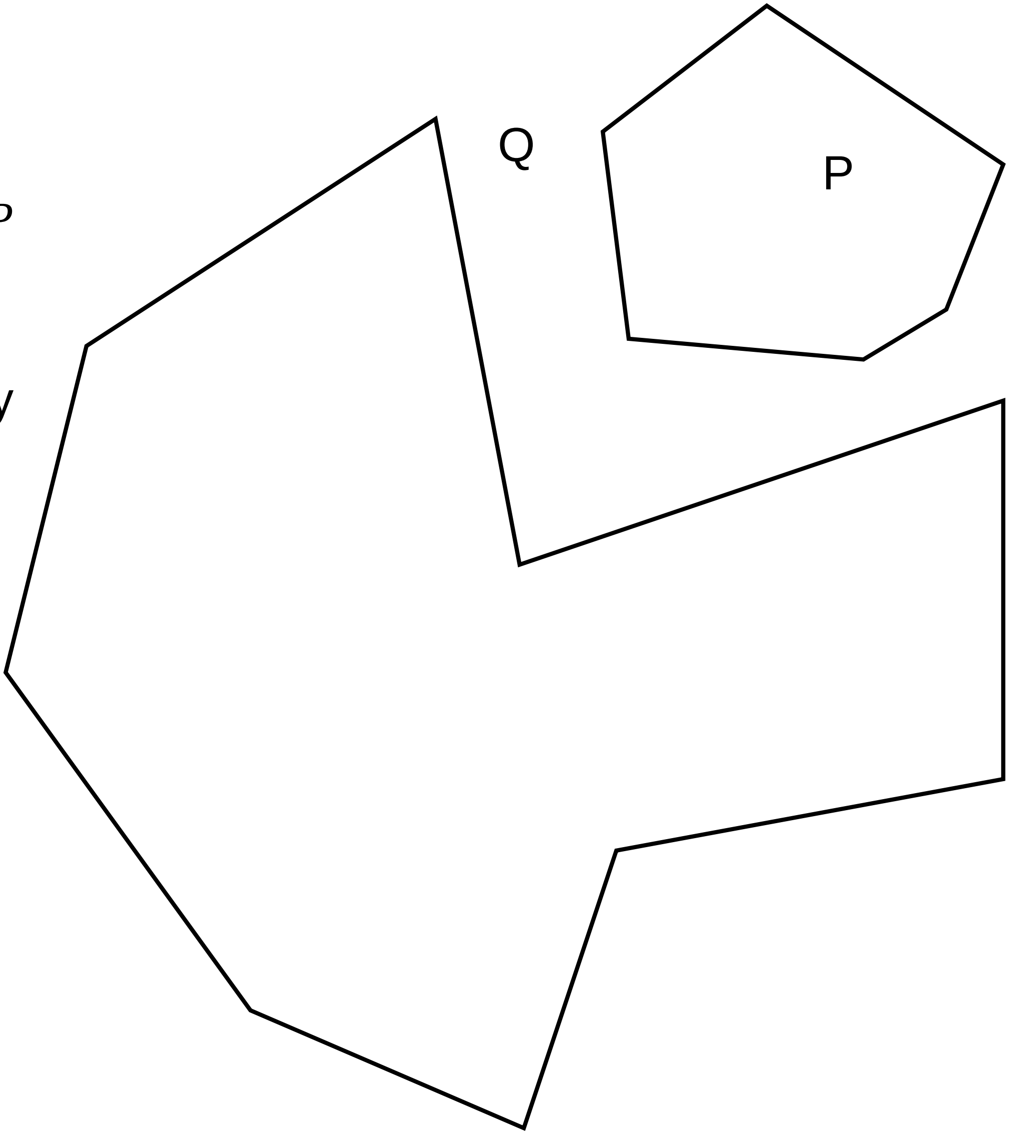
- Filter step: Already studied
- Refine: How to find the intersection of two polygons?
- For any two polygons, there are three cases:
  1. Polygons are disjoint
  2. One polygon is contained in the other polygon
  3. Polygon boundaries intersect

Case 1: Disjoint

No intersection points

Neither $P \subset Q$ nor $Q \subset P$

The intersection is empty



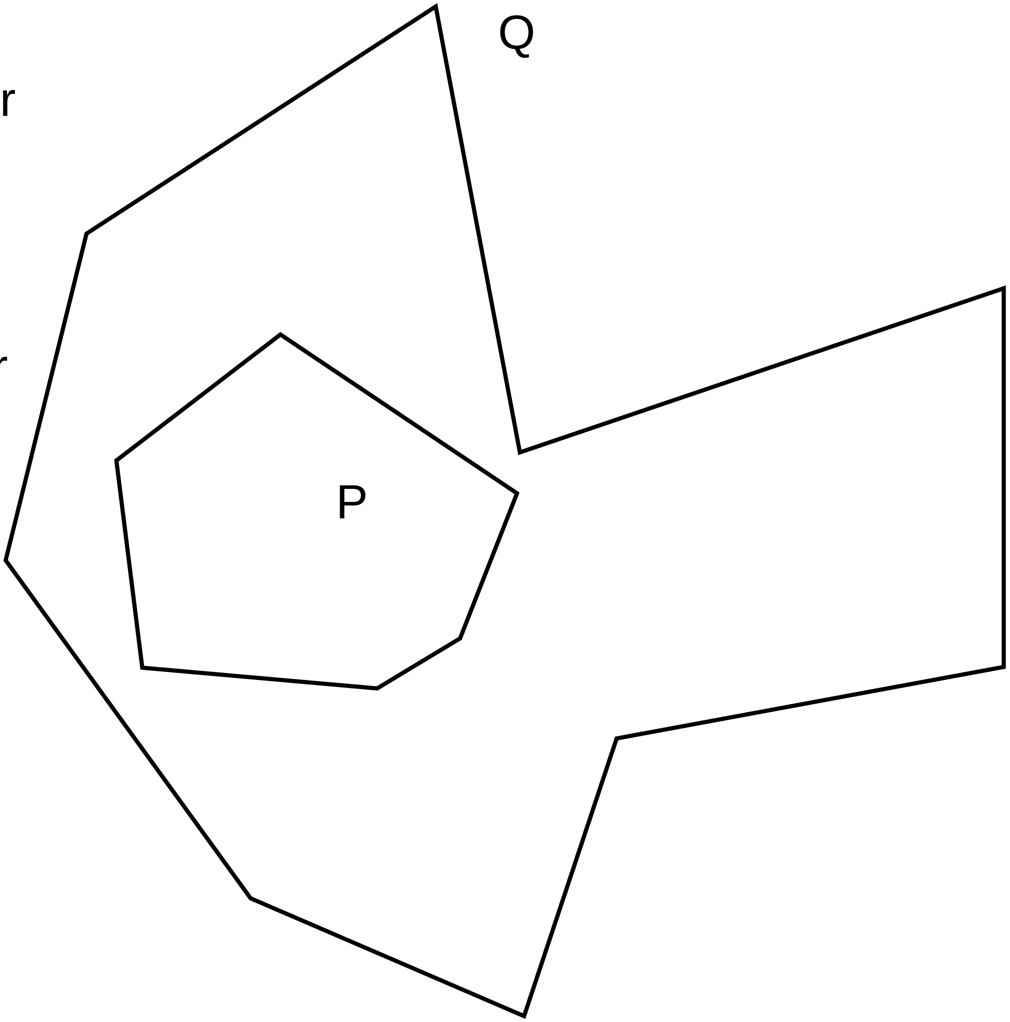
Case 2: Contained

No intersection points

If $P \subset Q$, then any corner of P is $\in Q$

If $Q \subset P$, then any corner of Q is $\in P$

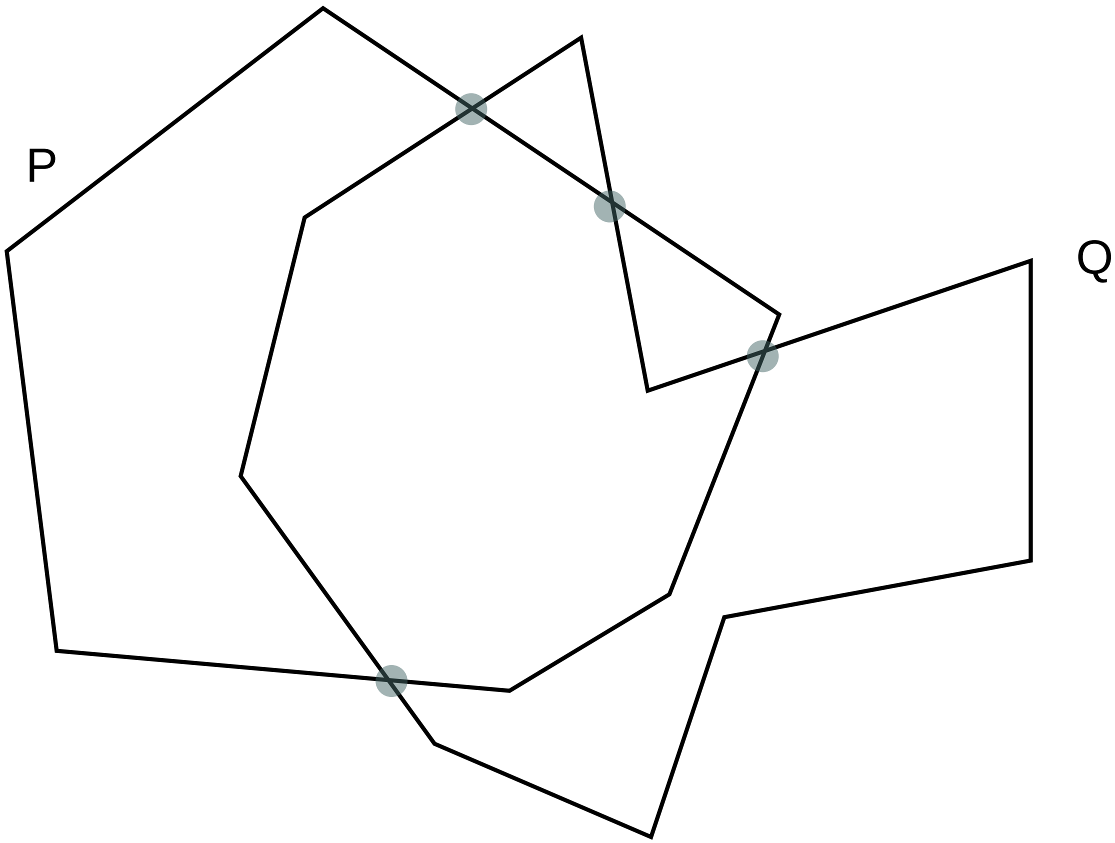
The intersection is the contained polygon



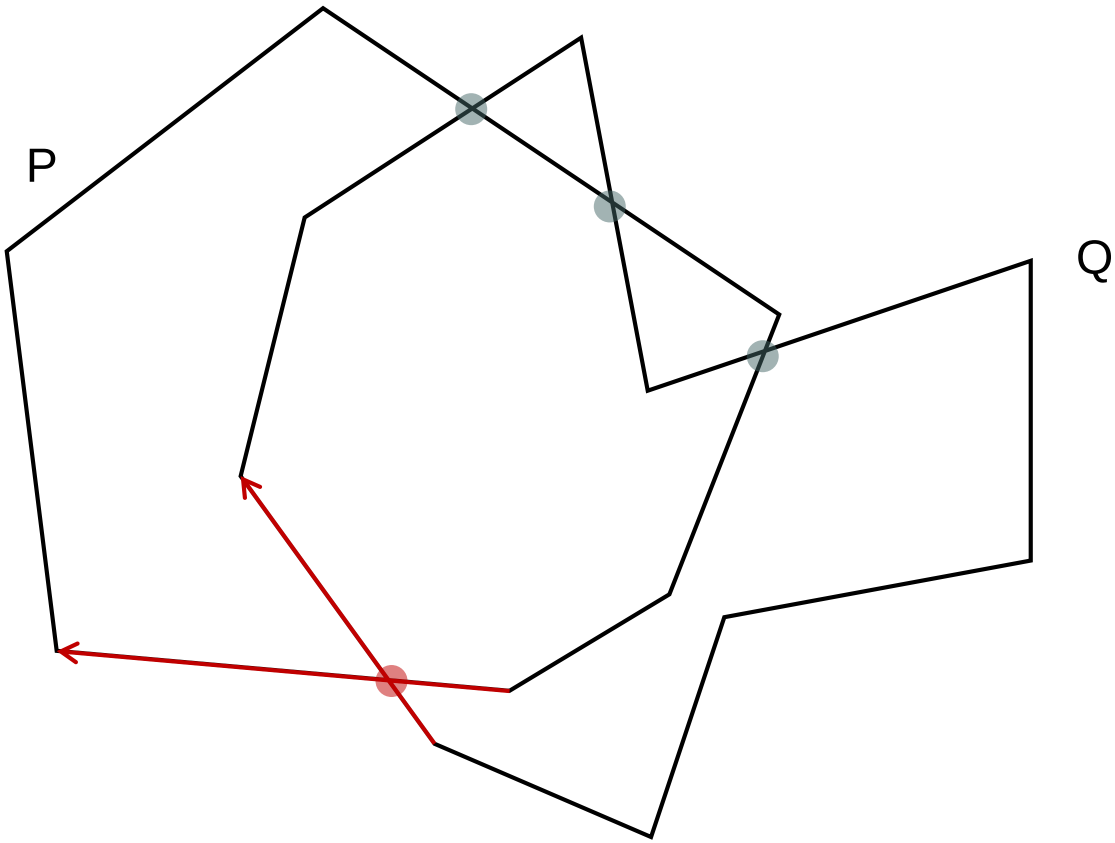
Case 3: Intersecting

- If the boundaries of the two polygons overlap, then there is at least two polygon edges that overlap
- Naïve solution: Compute all intersections between every pair of edges $O(m \cdot n)$
  - Where m and n are the sizes of the two polygons
- We can also use the line-segment sweep-line algorithm
  - Run in $O\big((k+I)\log k\big)$ where $k = m + n$ and I is the number of intersections
  - If we only need to test, we can stop at the first intersection

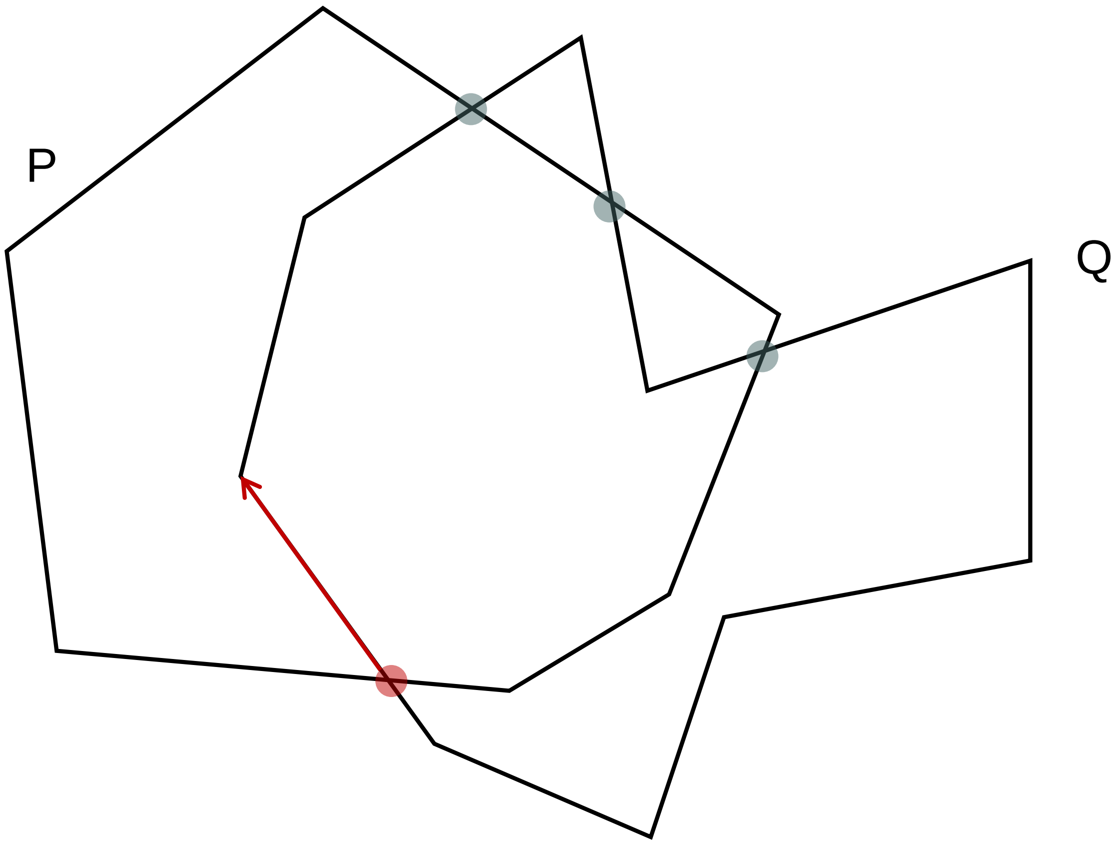
Computing the Intersection



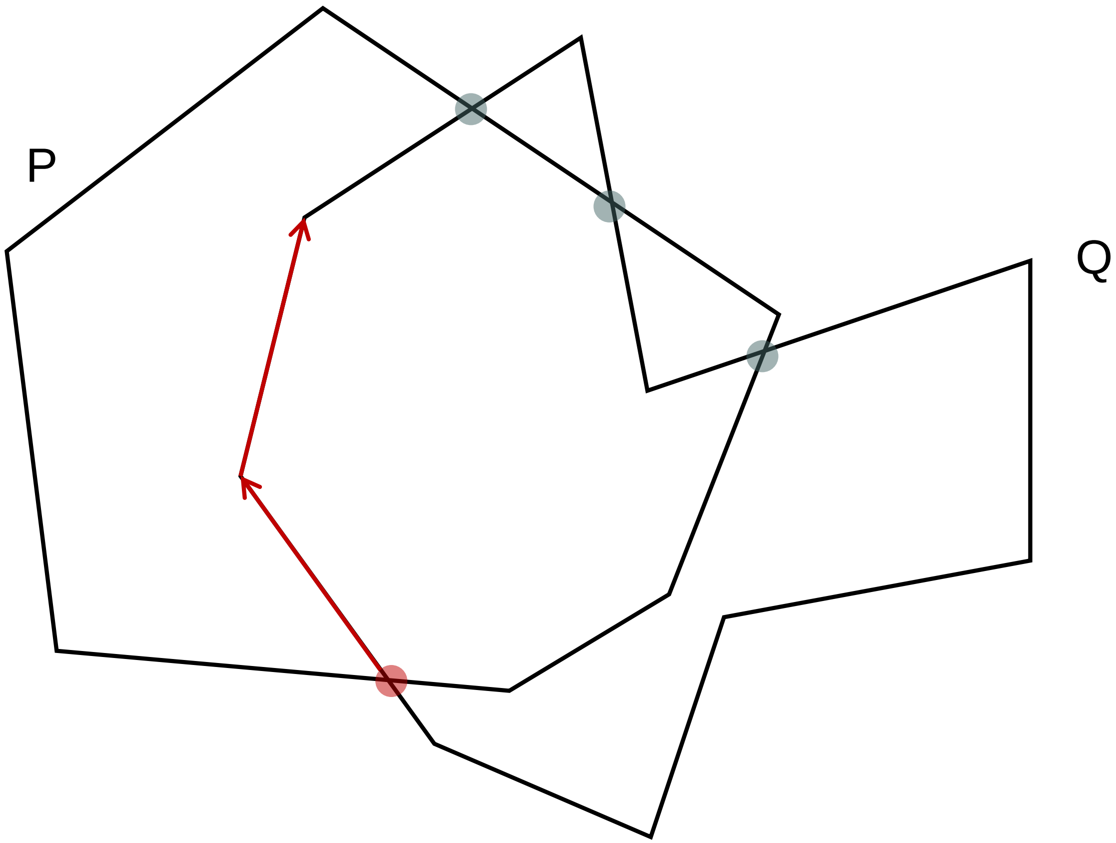
Computing the Intersection



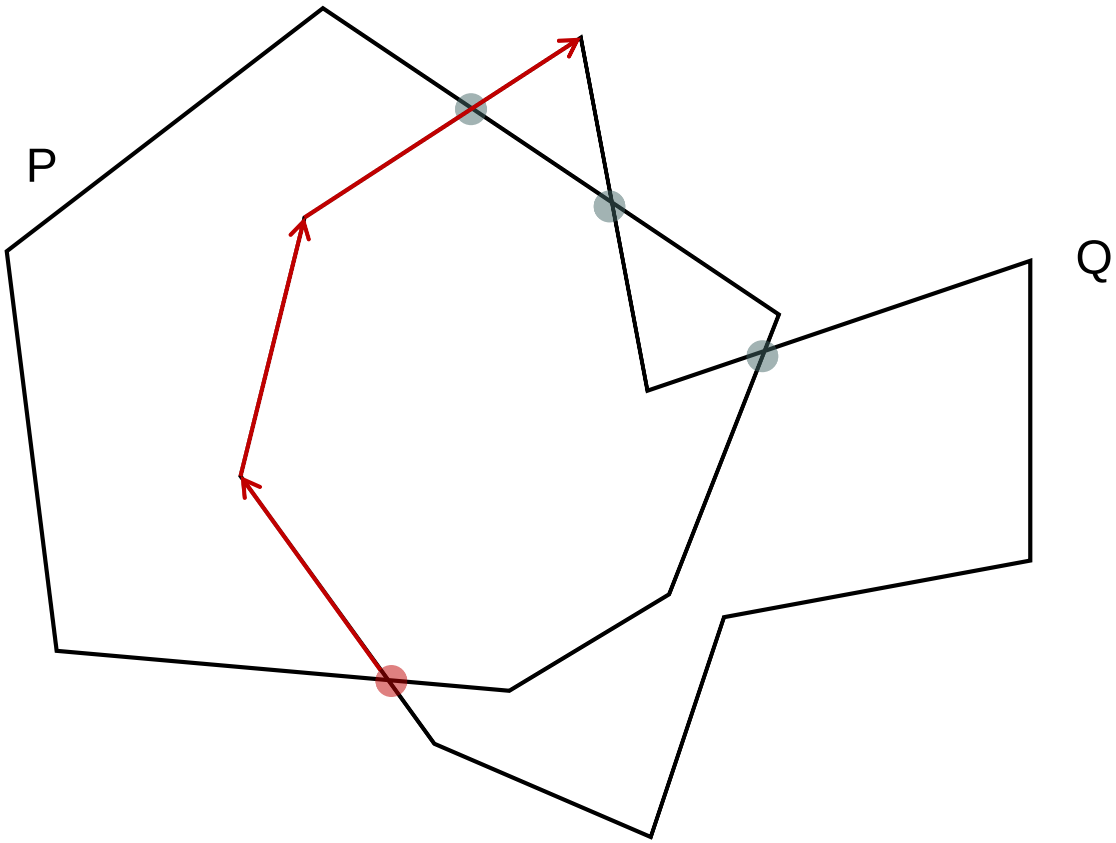
Computing the Intersection



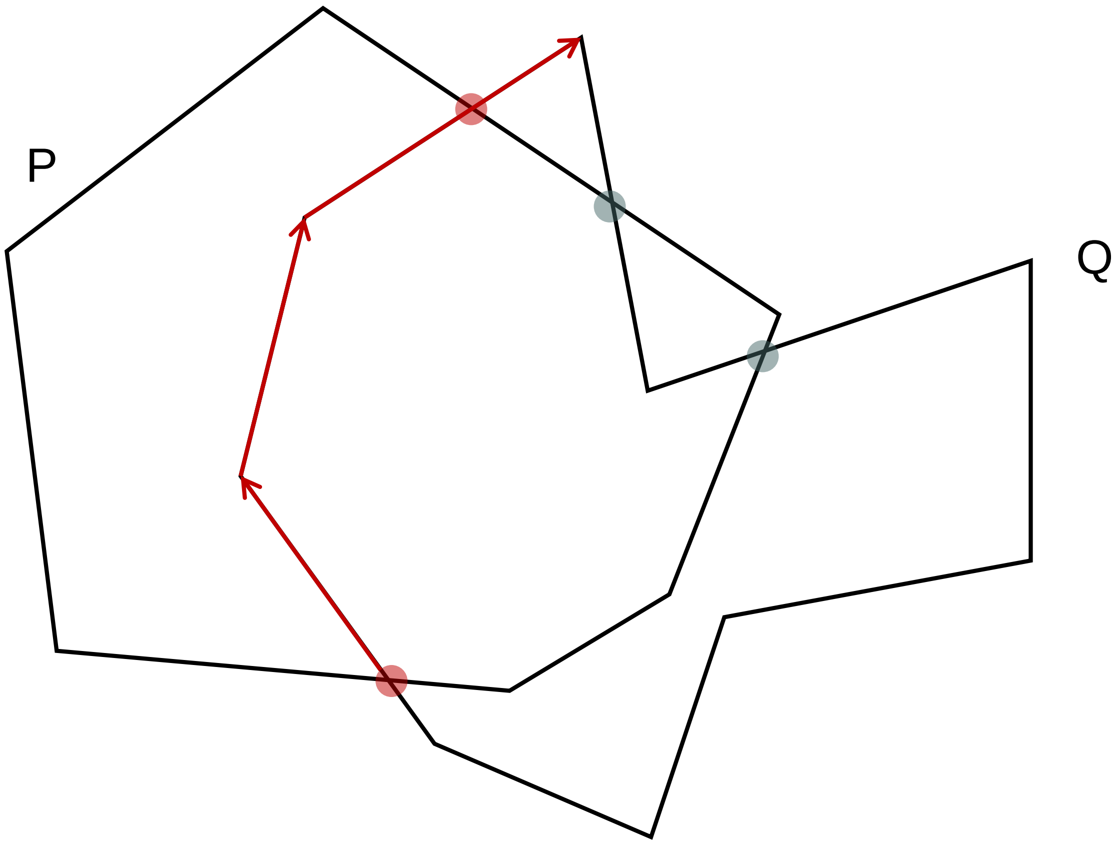
Computing the Intersection



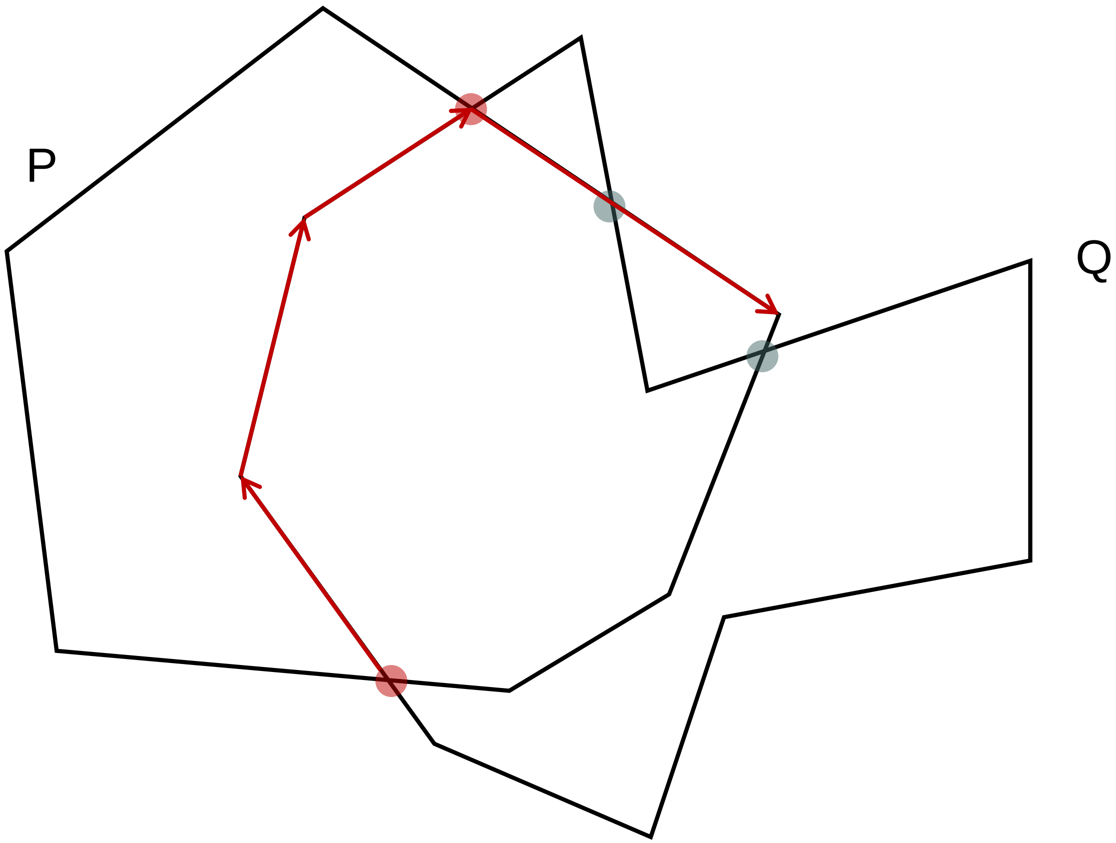
Computing the Intersection



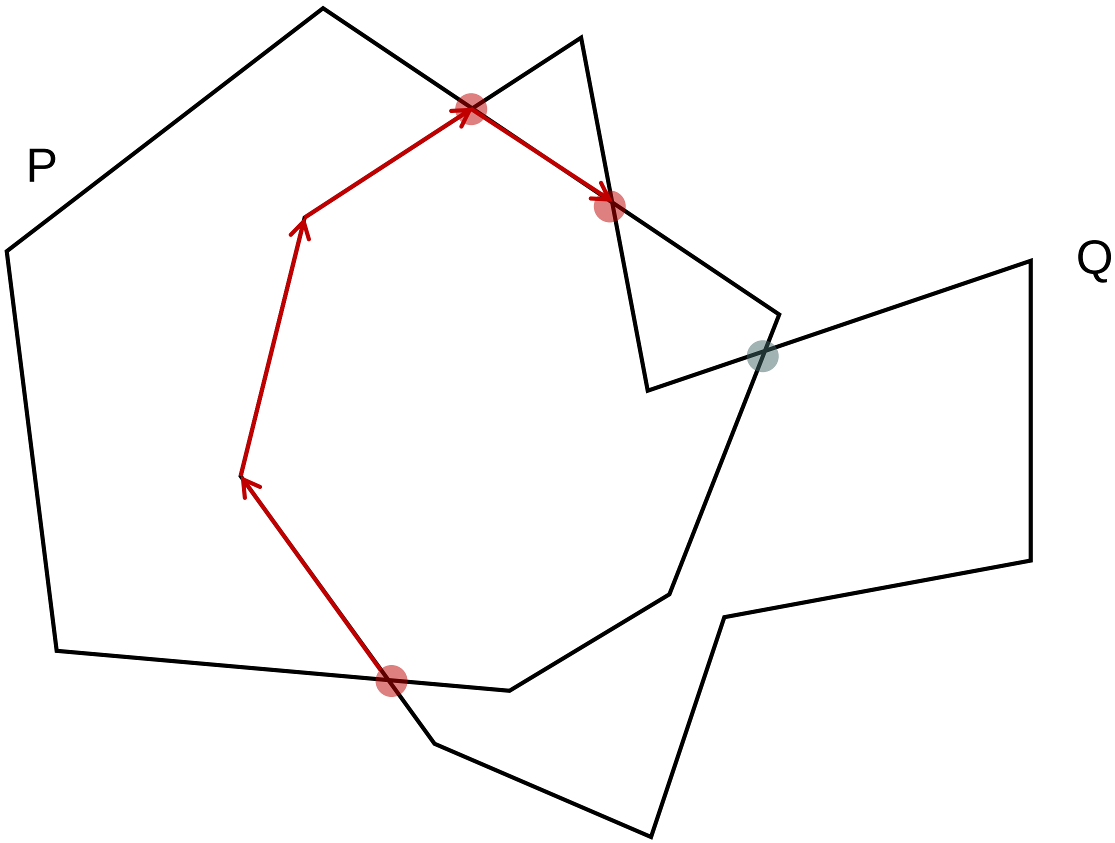
Computing the Intersection



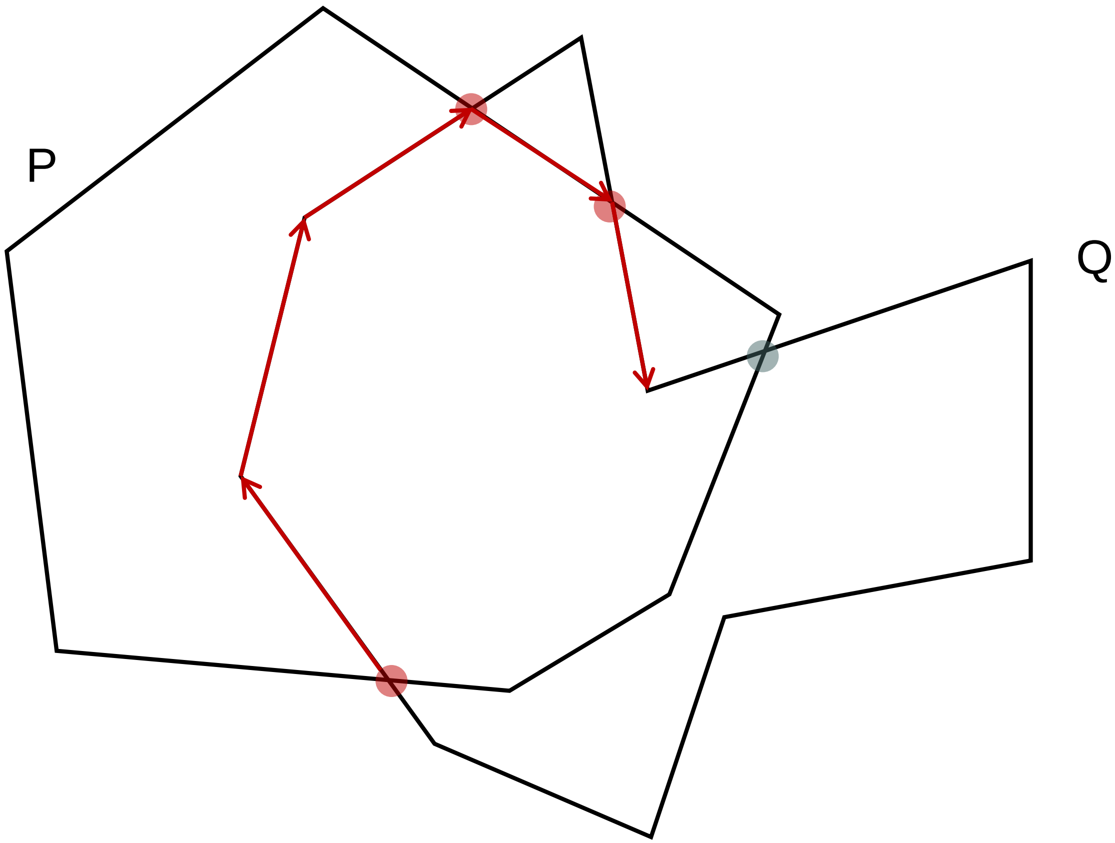
Computing the Intersection



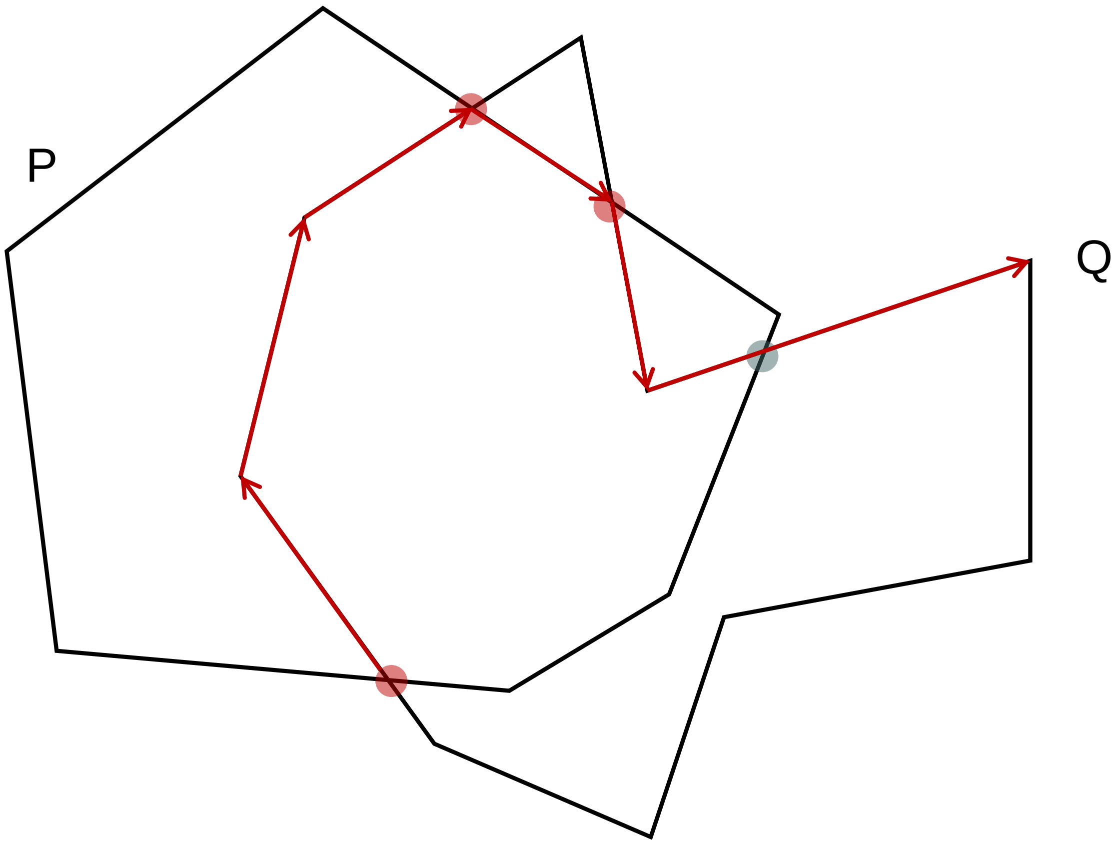
Computing the Intersection



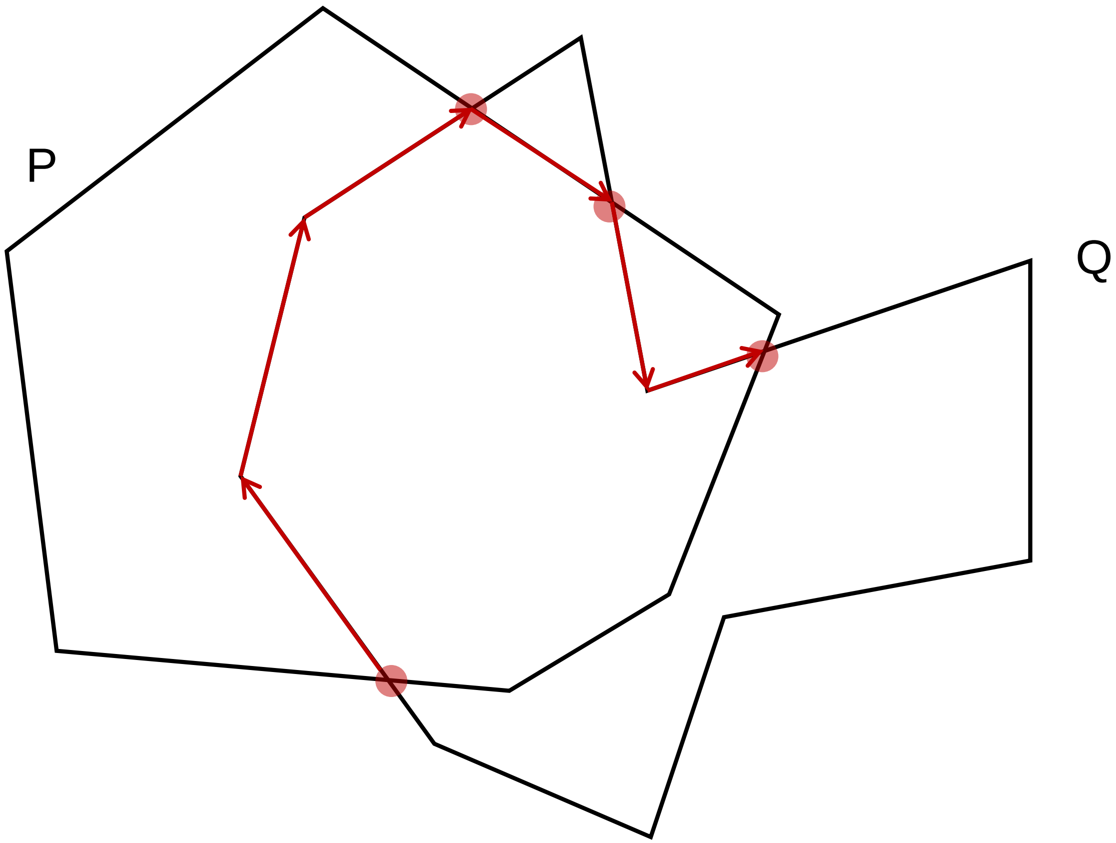
Computing the Intersection



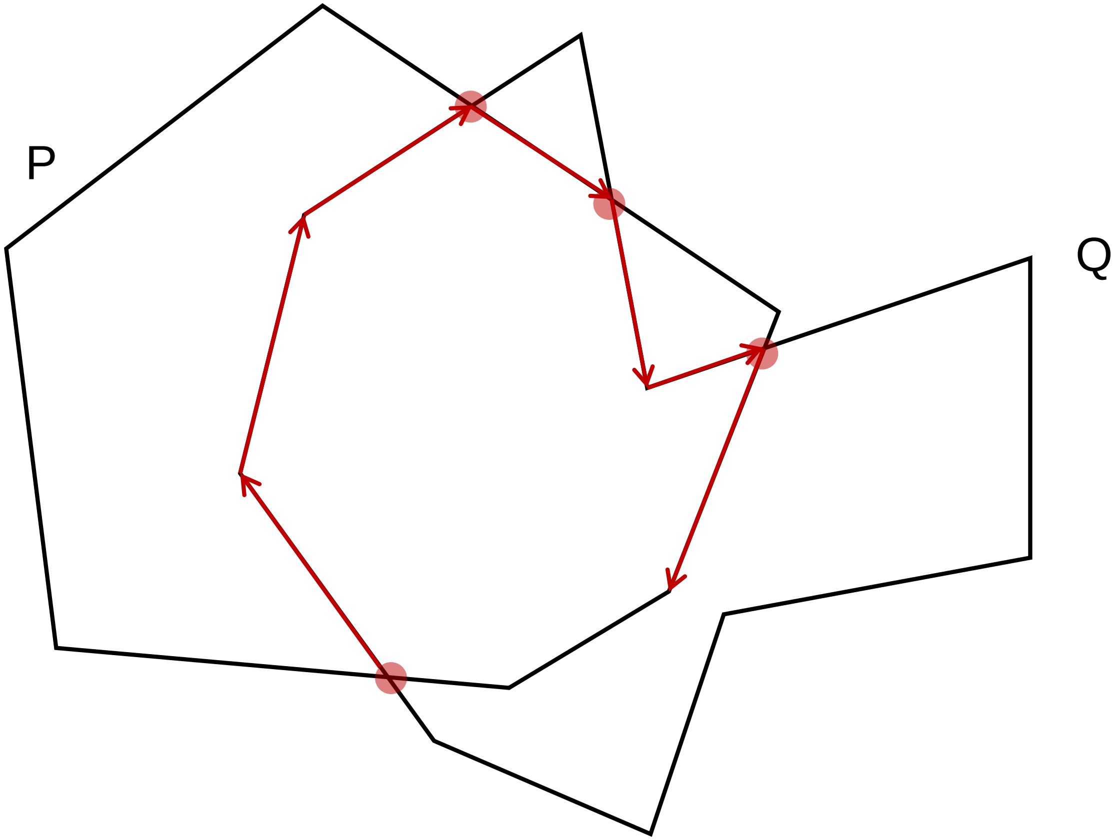
Computing the Intersection



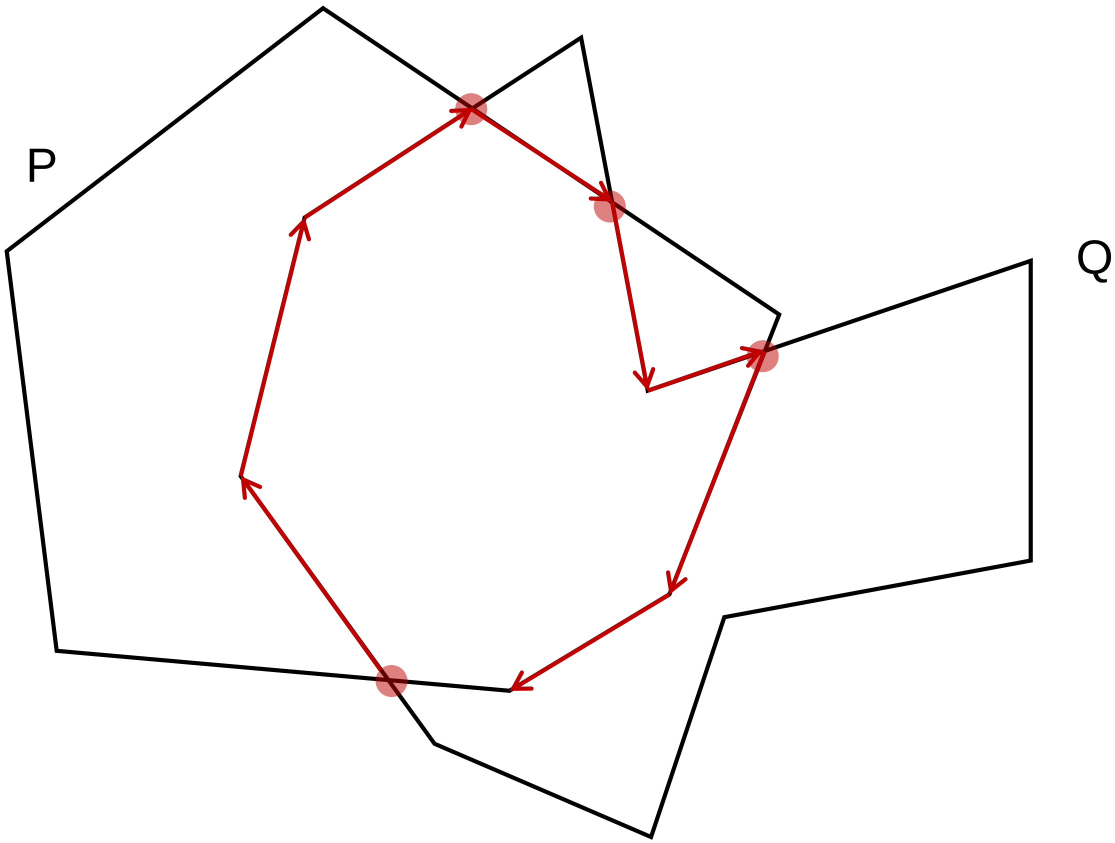
Computing the Intersection



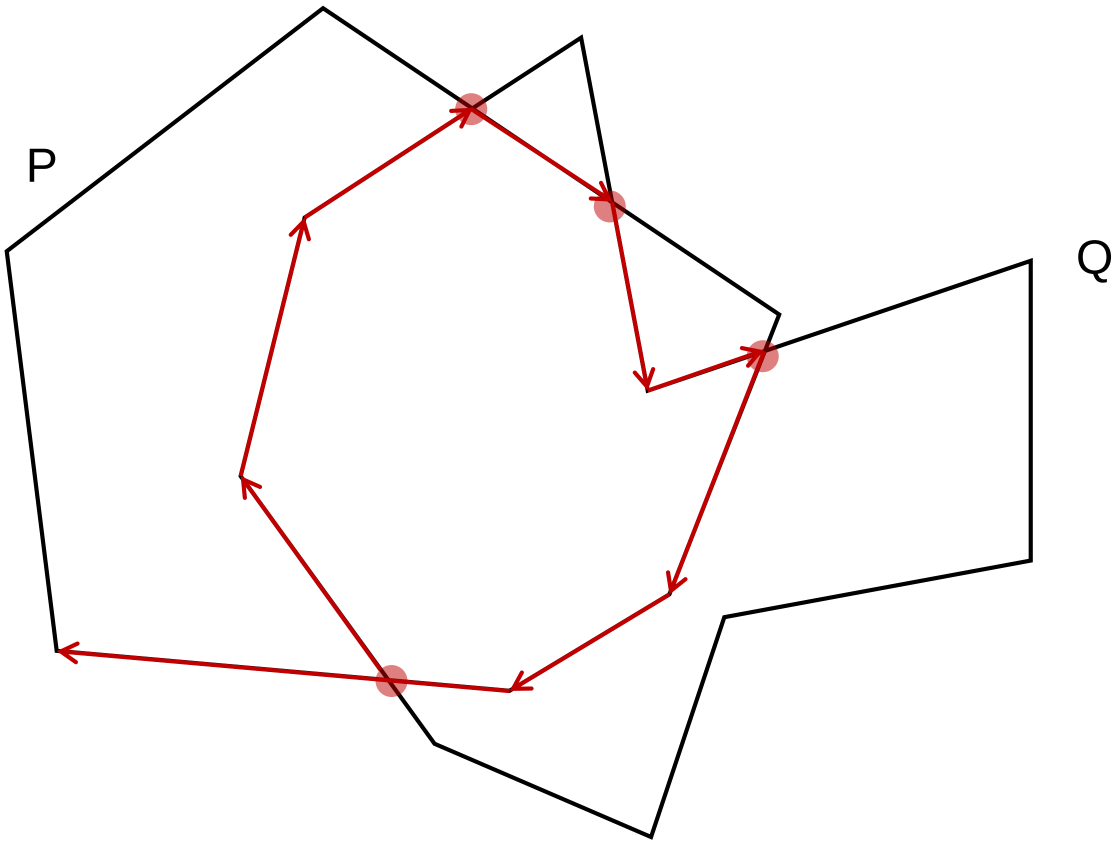
Computing the Intersection



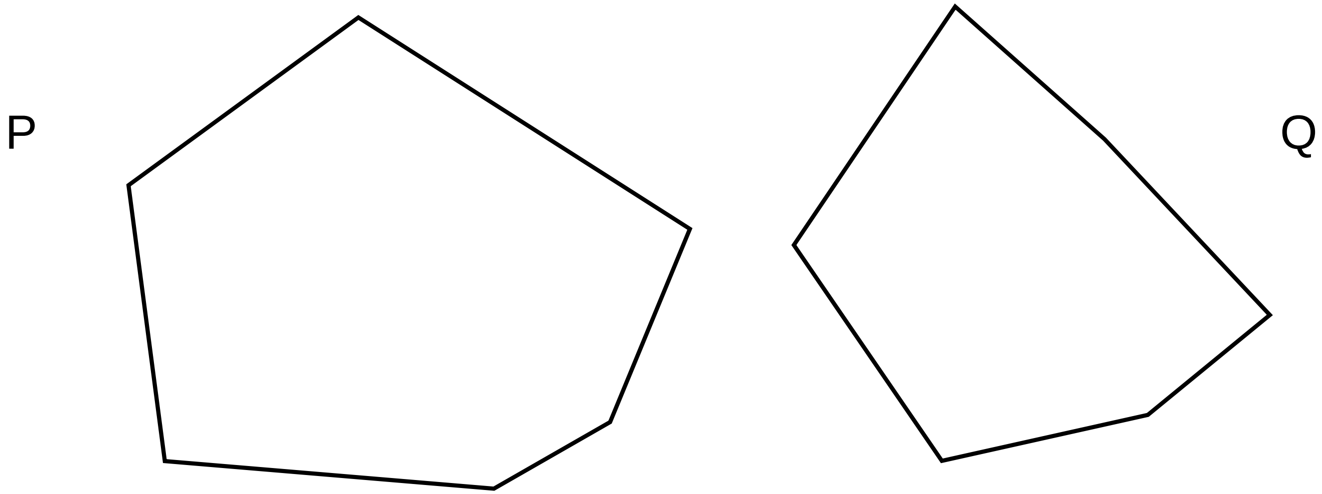
Computing the Intersection



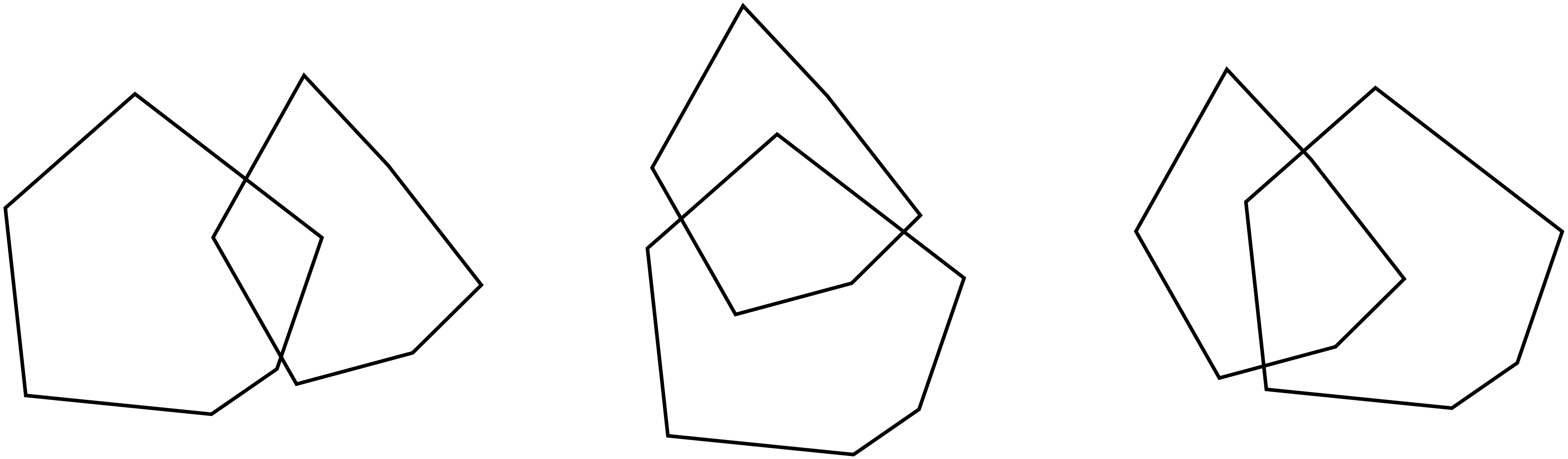
Computing the Intersection



Special Case: Convex Polygons



Convex Polygon Overlaps

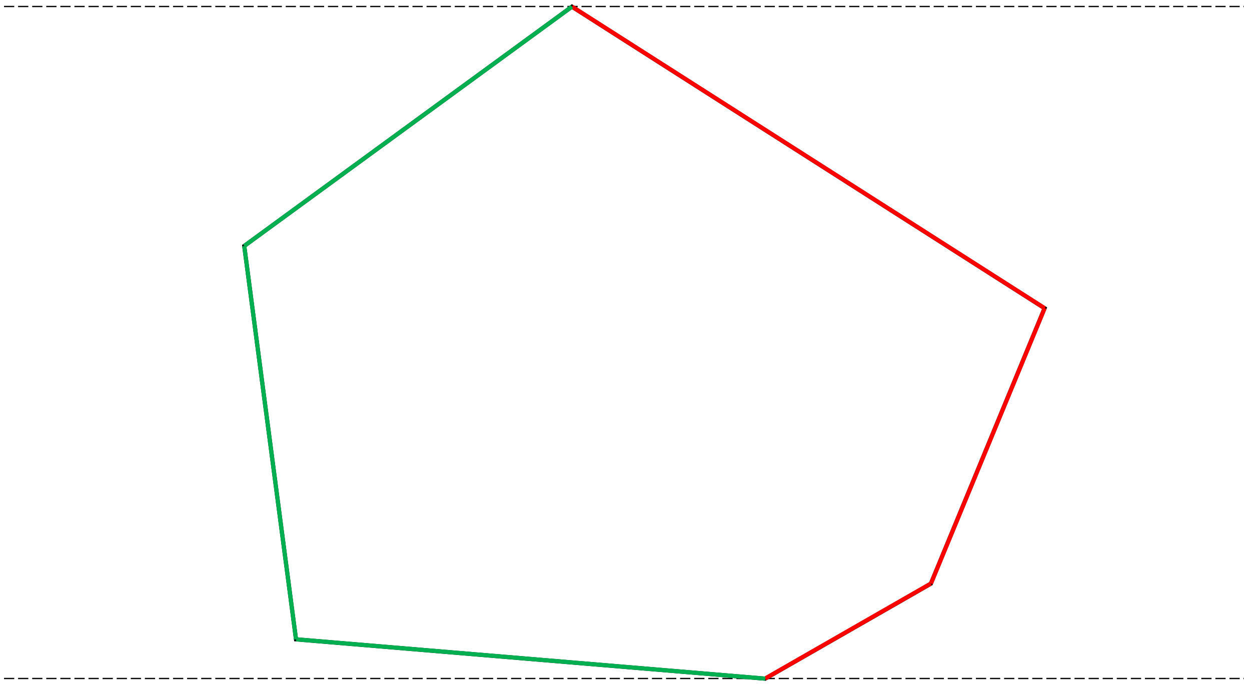


Right-left

Right-right
left-left

Left-right

Left-right Split

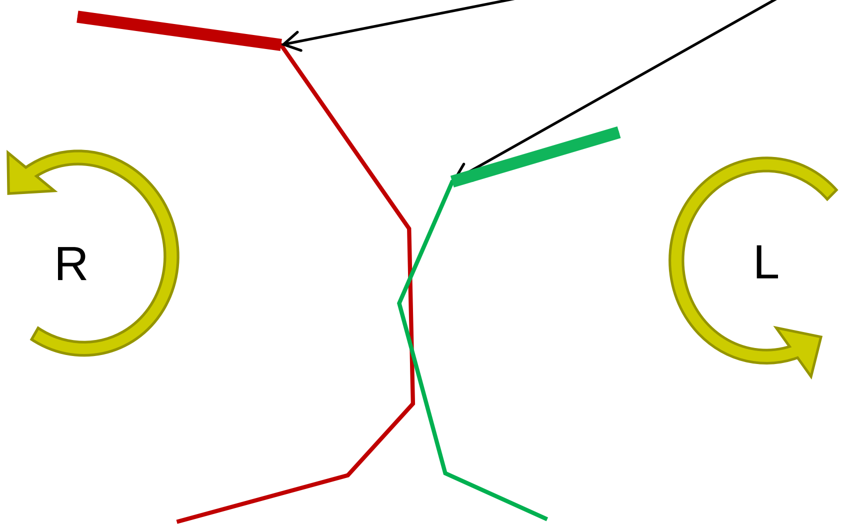


Left-right Overlap Test

Observation: the points of each half are monotone along the y-axis

Start with the top two segments from each half-hull

Do they intersect? No!

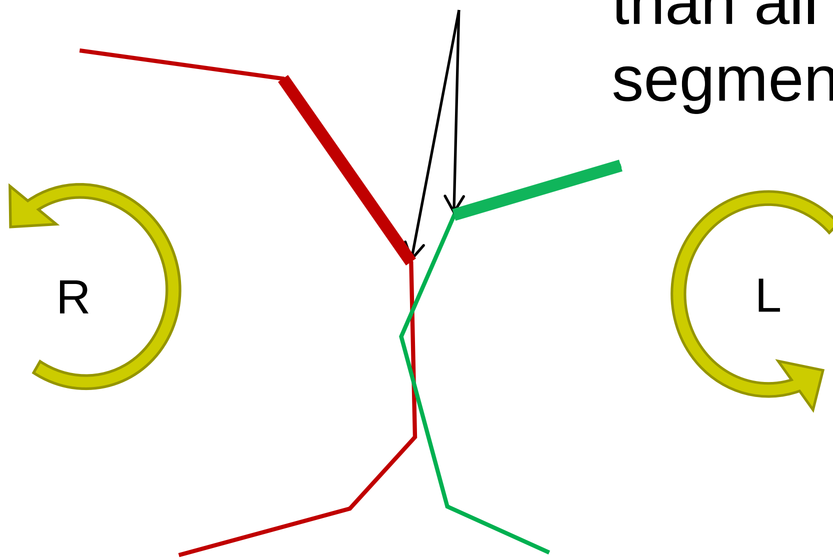


Since the segments that form each hull are monotone (i.e., going down), there is no chance for the top red segment to intersect the green half-hull

Left-right Overlap Test

Do the current segments intersect? No!

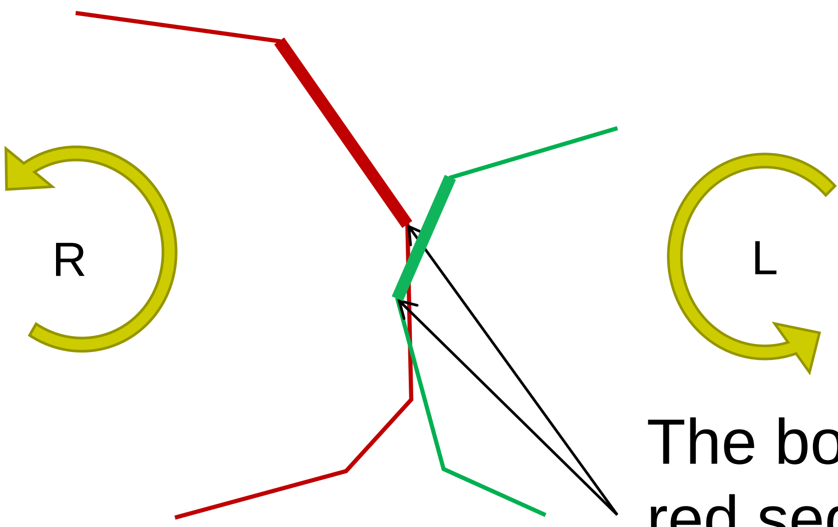
The bottom point of the green segment is higher than all remaining red segments



Skip to the next green line segment

Left-right Overlap Test

Do the current segments intersect? No!

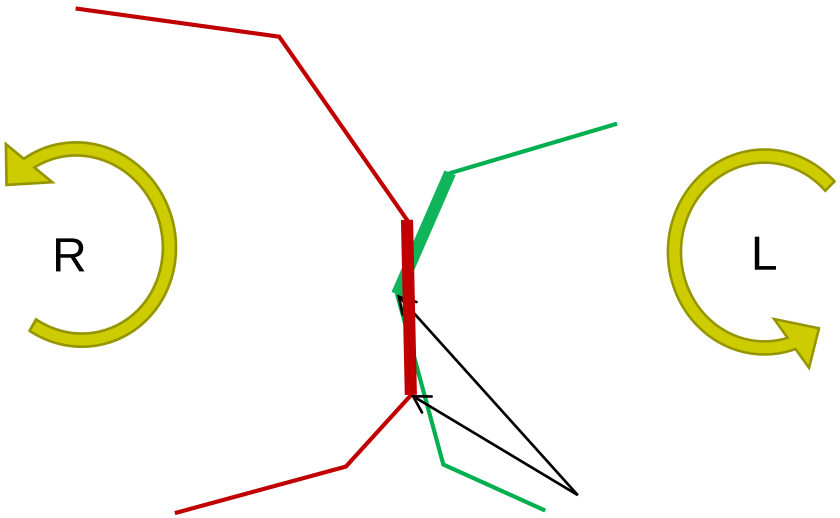


The bottom point of the red segment is higher. Skip to the next.

Left-right Overlap Test

Do the current segments intersect? Yes!

Report the intersection

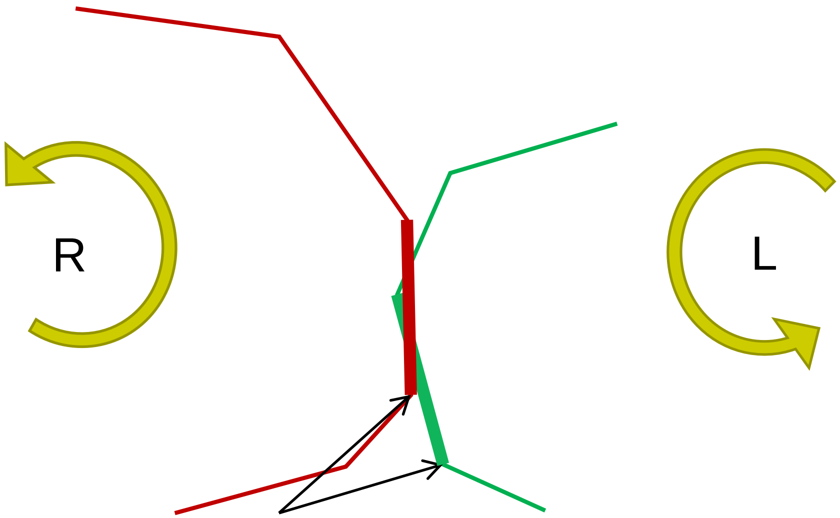


Bottom green point is higher. Skip to the next green segment.

Left-right Overlap Test

Do the current segments intersect? Yes!

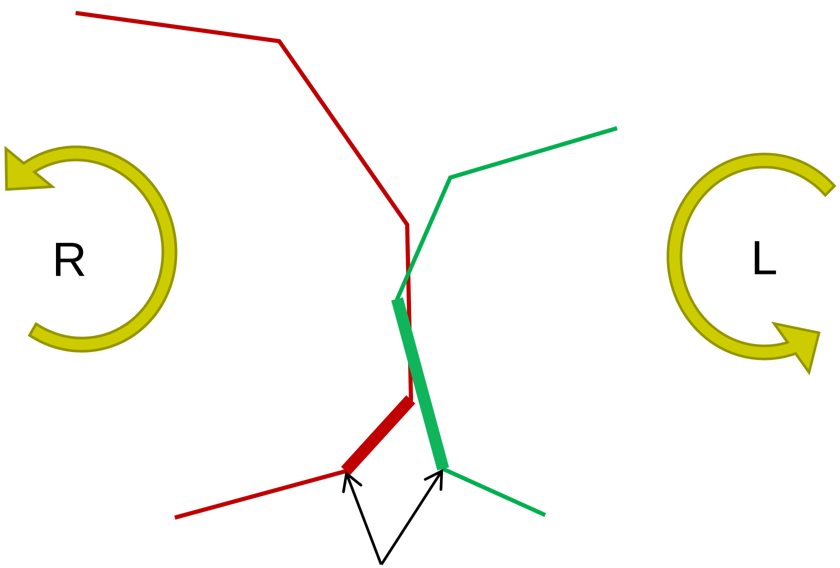
Report the intersection



Bottom red point is higher.
Skip to the next red segment.

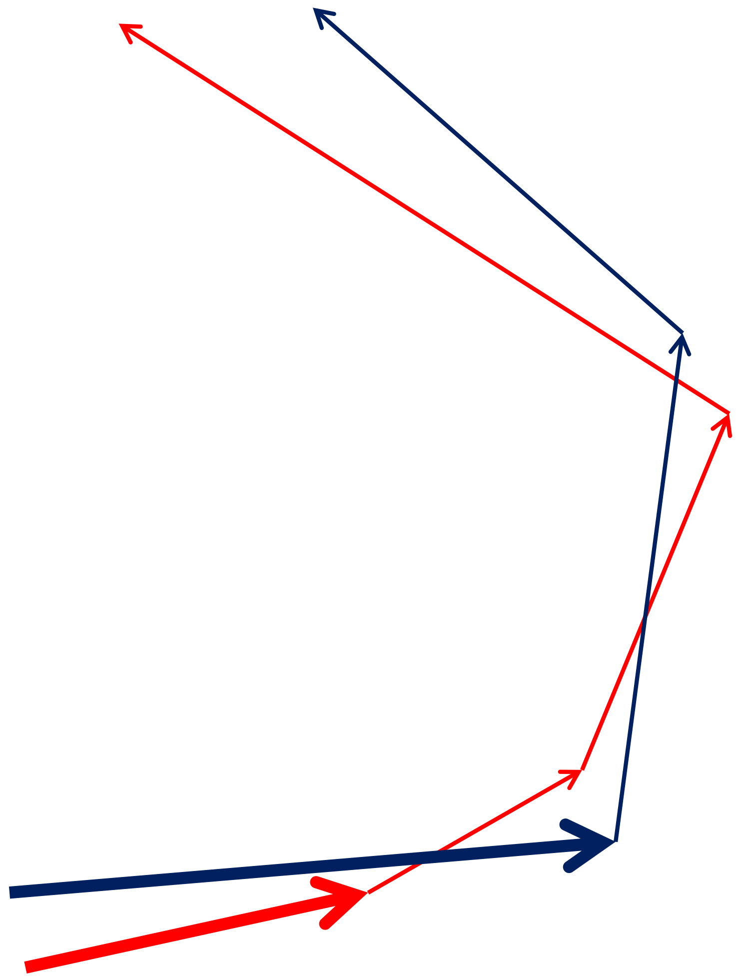
Left-right Overlap Test

Do the current segments intersect? No!

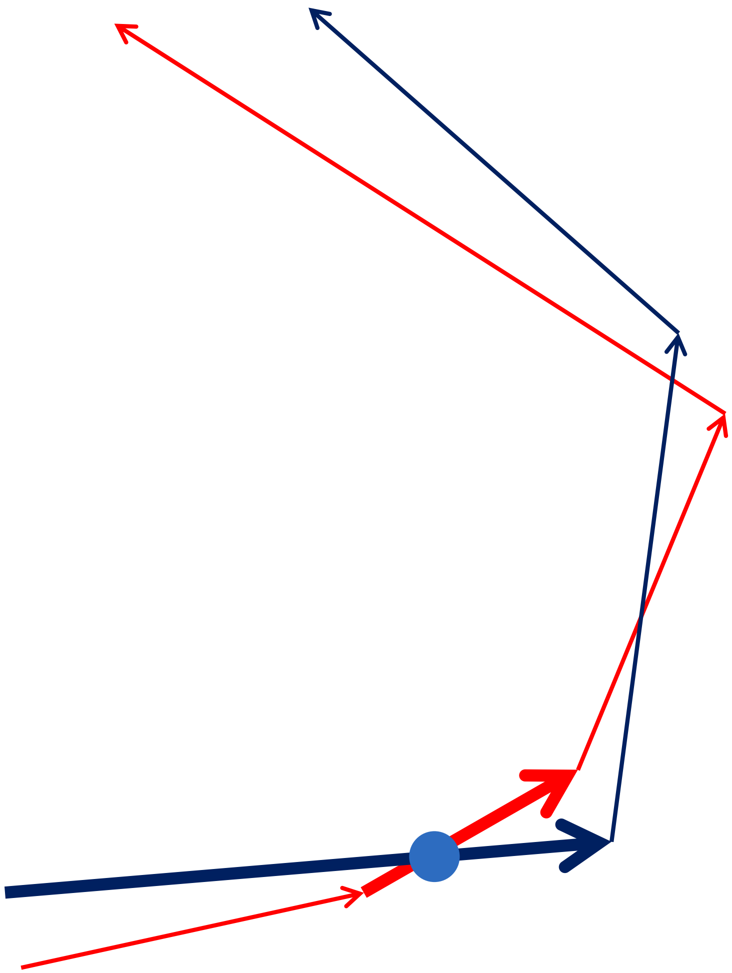


Both bottom points are at equal y-coordinate. Skip any of them

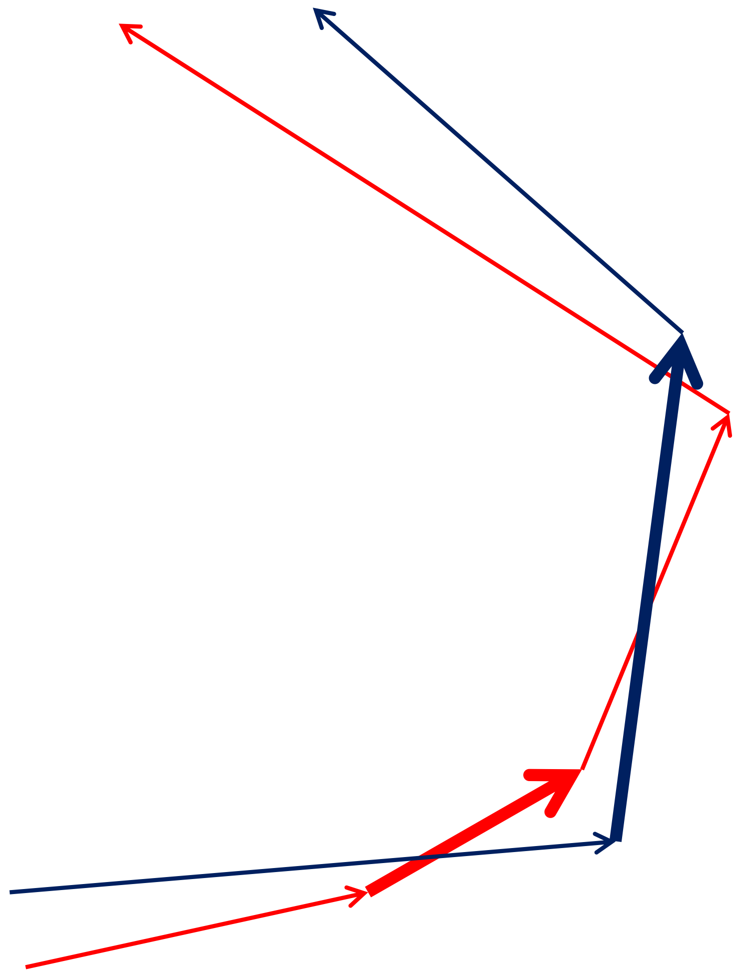
Right-right Overlap Test



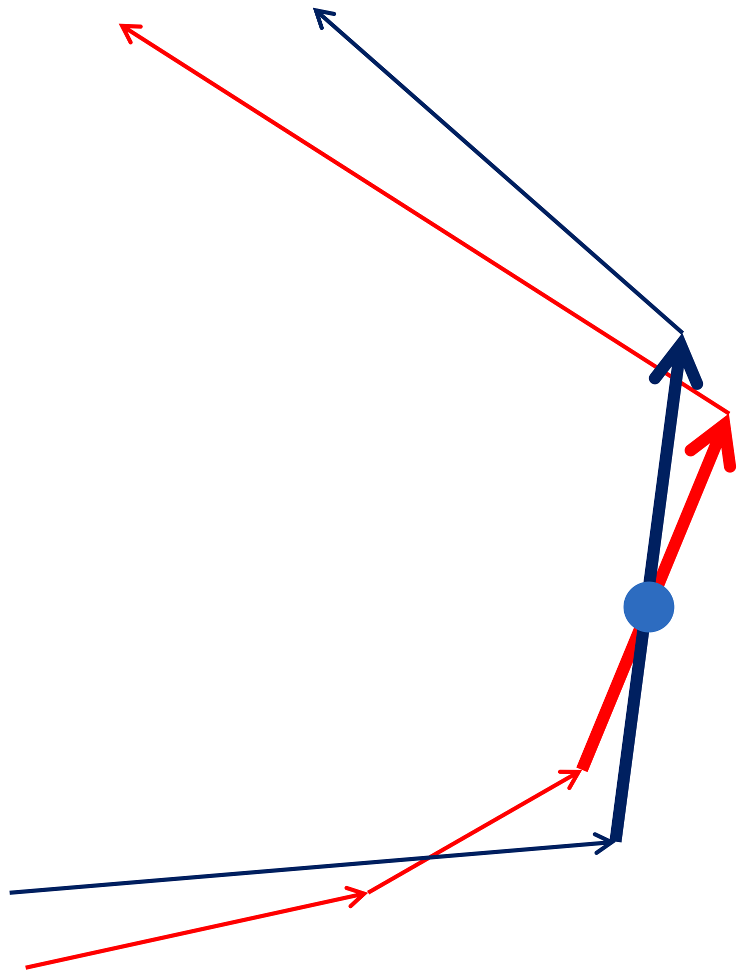
Right-right Overlap Test



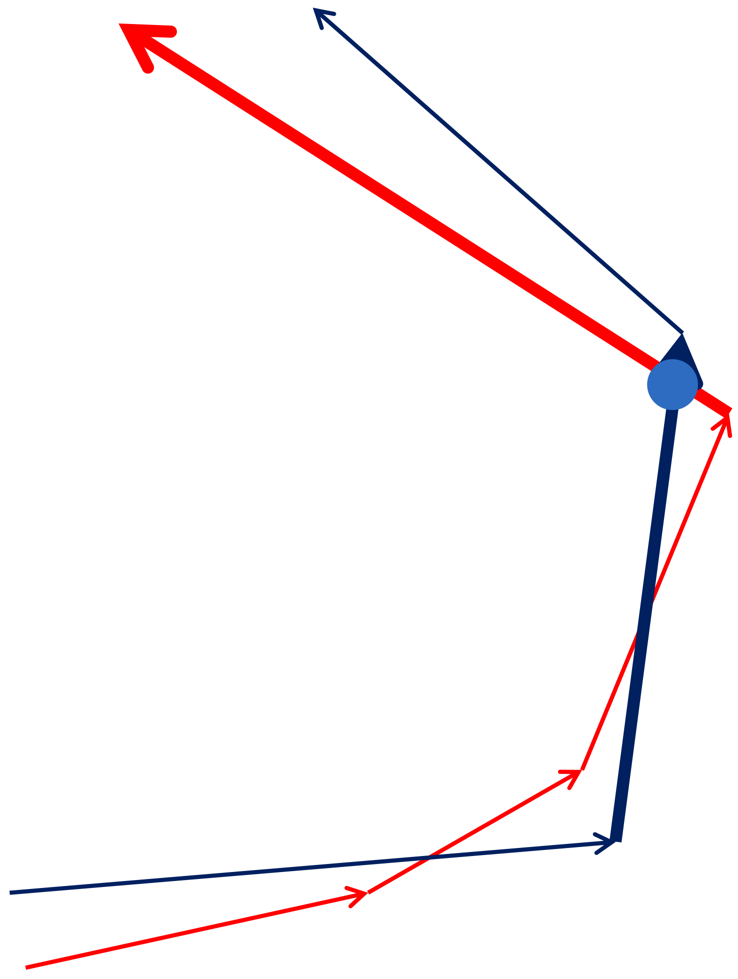
Right-right Overlap Test



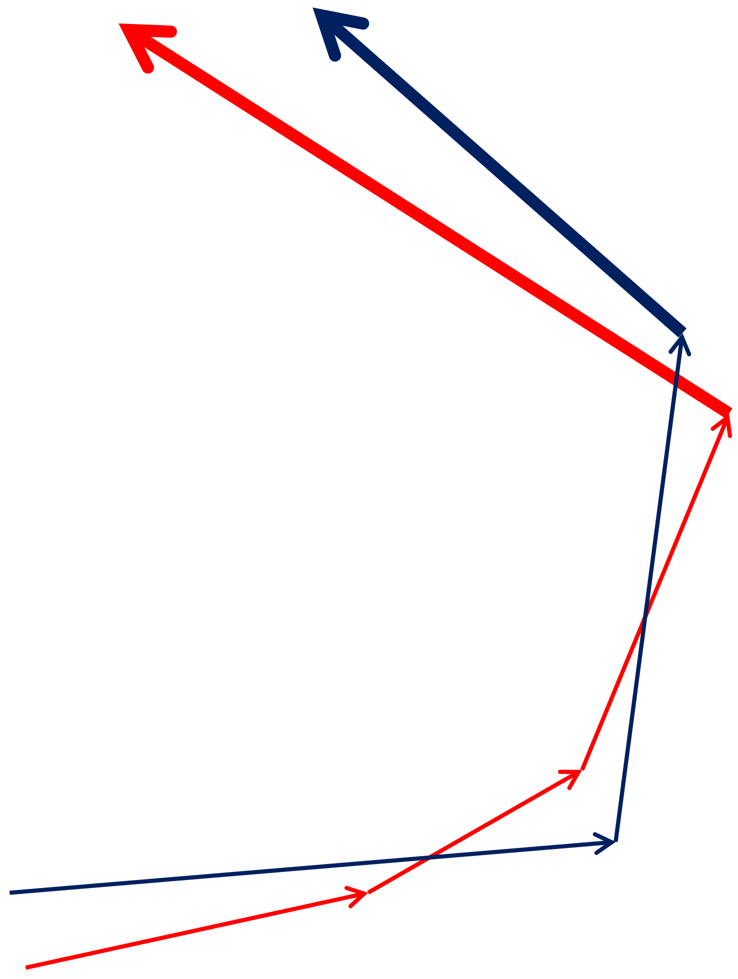
Right-right Overlap Test



Right-right Overlap Test



Right-right Overlap Test



Convex Polygon Intersection

- If only testing is required, the algorithm can terminate as soon as the first intersection is found
- If the polygon intersection is needed, the algorithm reports all intersections
- The algorithm terminates when all segments are inspected
- Running time: $O(m+n)$, each iteration skips one segment