Line Segment Intersection

Presented by

Nilesh Mukherjee, Faizan Pathan

Department of CSE, IIIT Bhubaneswar

January 1, 2026

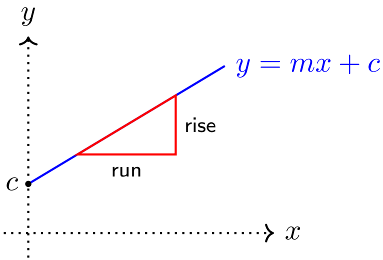
Contents

Revising the Equation of a Line

- The equation of a line is:

$$y = mx + c$$

- x is the input value (horizontal axis)
- m is the slope (ratio of rise over run)
- c is the y-intercept (where the line touches the y-axis)



Revising the Cross Product

- For two vectors $\vec{v}$ and $\vec{w}$:
  - In **2D**: the cross product gives an **area** of the parallelogram formed by those two vectors
  - In **3D**: the cross product is a vector **perpendicular** to both v and w, with magnitude equal to the parallelogram area
- So cross product $=$ **area + direction information**

This is what we need

Introduction

Line Segment Intersection is a fundamental concept in computational geometry.

It helps us determine whether two finite line segments in a plane intersect, and if they do, find the **exact intersection point**.

This technique is widely used in:

- Computer graphics
- Collision detection
- Geographic Information Systems (GIS)
- Path planning and robotics

Concept Overview

Two line segments, say AB and CD, may either:

1. Not intersect at all,
2. Intersect at a single point, or
3. Overlap (if they are collinear).

Goal: Check if they intersect, and if so, find that point.

Understanding Orientation

What is Orientation? Orientation describes the *relative turning direction* made when moving from one point to another in the plane.

Intuitive Meaning

If we move from point P to Q, and then to R:

- If the turn is to the **left**, it is **counterclockwise (CCW)**.
- If the turn is to the **right**, it is **clockwise (CW)**.
- If all three points lie on the same line, they are **collinear**.

Why we need this:

- Orientation tells us on which side of a segment another point lies.
- By comparing orientations, we can detect whether two segments cross each other.

Slope Intuition: A Starting Idea

Consider the line through $A(x_1, y_1)$ and $B(x_2, y_2)$.
Slope of the line:

$$m_{AB} = \frac{y_2 - y_1}{x_2 - x_1}$$

Slope from the same base point to any point $C(x_3, y_3)$:

$$m_{AC} = \frac{y_3 - y_1}{x_3 - x_1}$$

- If m_{AC} is "steeper" than m_{AB}, the point is on one side.
- If less steep, the point is on the other side.

But It doesn't work in all cases.

Slope Is Not Enough

Slope cannot reliably determine which side of a segment a point lies on.

- Cannot handle vertical lines $(x_2 = x_1)$
- Same slope does not guarantee same direction

Therefore, we use another test for this:

The Orientation Test

Orientation Test: Overview

To determine intersection, we use the **Orientation Test**, which expresses the **relative position of a point** with respect to a directed line segment.

Definition:

For a directed line segment joining $A(x_1, y_1)$ and $B(x_2, y_2)$, and another point $C(x_3, y_3)$, the orientation of $C(x_3, y_3)$ with respect to this line is given by:

$$\text{orientation} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

- > 0: C lies to the **left** of the line segment AB (Counterclockwise)
- < 0: C lies to the **right** of the line segment AB (Clockwise)
- $= 0$: C lies **on the line** AB (Collinear)

How Orientation Helps Detect Intersection

Now that we know how to compute the orientation of a point with respect to a directed line segment, the next step is to understand how this applies to **two entire line segments**.

Consider two line segments: AB **and** CD

To check whether these two segments intersect, we compute the orientation of each endpoint with respect to the other segment:

- Orientation of C w.r.t. AB
- Orientation of D w.r.t. AB
- Orientation of A w.r.t. CD
- Orientation of B w.r.t. CD

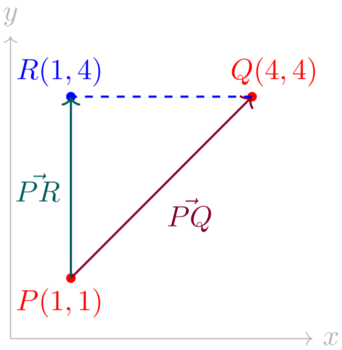
Key Idea

If the endpoints of one segment lie on **opposite sides** of the other segment (i.e., orientations have different signs), and the same is true in the reverse direction, then the two line segments **must intersect**.

Visualizing Orientation

Goal: Determine the orientation of point $R(x_R, y_R)$ with respect to the line segment PQ.

$$P(1,1), \quad Q(4,4), \quad R(1,4)$$



Step 1: Determinant Calculation

$$\Delta = \begin{vmatrix} x_Q - x_P & x_R - x_P \\ y_Q - y_P & y_R - y_P \end{vmatrix} = \begin{vmatrix} 3 & 0 \\ 3 & 3 \end{vmatrix} = 9$$

Step 2: Interpret Sign

$\Delta > 0 \Rightarrow$ Counterclockwise orientation

Step 3: Vector Meaning

$$\vec{PQ} \times \vec{PR} = (0, 0, 9)$$

The negative z-component means the vector points **out of the screen**.

Finding the Intersection Point

Consider two lines, AB and CD. Let $A(x_1, y_1)$ and $B(x_2, y_2)$ define the first line, and $C(x_3, y_3)$ and $D(x_4, y_4)$ define the second line.
We imagine a point $P(x, y)$ lying on line AB. If P lies on AB, then the slope of AP must be the same as the slope of AB:

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}.$$

Cross-multiplying to remove the fraction:

$$(y - y_1)(x_2 - x_1) = (x - x_1)(y_2 - y_1).$$

Expanding and simplifying gives the equation of line AB:

$$(y_2 - y_1)x - (x_2 - x_1)y = x_1 y_2 - y_1 x_2.$$

Finding the Intersection Point (contd.)

Now, let $Q(x, y)$ be a point lying on line CD, where $C(x_3, y_3)$ and $D(x_4, y_4)$.
For Q to lie on CD, the slope of CQ must equal the slope of CD:

$$\frac{y - y_3}{x - x_3} = \frac{y_4 - y_3}{x_4 - x_3}.$$

Cross-multiplying:

$$(y - y_3)(x_4 - x_3) = (x - x_3)(y_4 - y_3).$$

Simplifying, the equation of line CD becomes:

$$(y_4 - y_3)x - (x_4 - x_3)y = x_3 y_4 - y_3 x_4.$$

Finding the Intersection Point (contd.)

At the intersection, points P and Q coincide, meaning both equations hold for the same (x, y).
Hence, we have the system:

$$\begin{cases} (y_2 - y_1)x - (x_2 - x_1)y = x_1y_2 - y_1x_2, \\ (y_4 - y_3)x - (x_4 - x_3)y = x_3y_4 - y_3x_4. \end{cases}$$

Let us define the constants:

$$A_1 = y_2 - y_1, \quad B_1 = -(x_2 - x_1), \quad C_1 = x_1y_2 - y_1x_2,$$
$$A_2 = y_4 - y_3, \quad B_2 = -(x_4 - x_3), \quad C_2 = x_3y_4 - y_3x_4.$$

The system can then be rewritten as:

$$A_1x + B_1y = C_1, \quad A_2x + B_2y = C_2.$$

Finding the Intersection Point (contd.)

Solving these two equations using Cramer's rule, we find:

$$x = \frac{C_1B_2 - C_2B_1}{\Delta}, \qquad y = \frac{A_1C_2 - A_2C_1}{\Delta}$$

Where, $\Delta = A_1B_2 - A_2B_1.$

If $\Delta = 0$, the lines are parallel or overlapping and do not have a unique intersection.

Finding the Intersection Point (final form)

Substituting the actual coordinate terms gives:

$$\Delta = (y_2 - y_1)(x_3 - x_4) - (y_4 - y_3)(x_1 - x_2),$$

$$C_1 = x_1 y_2 - y_1 x_2, \qquad C_2 = x_3 y_4 - y_3 x_4.$$

Hence, the intersection point is obtained as:

$$x = \frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_3 y_4 - y_3 x_4)(x_1 - x_2)}{(y_2 - y_1)(x_3 - x_4) - (y_4 - y_3)(x_1 - x_2)}$$

$$y = \frac{(y_2 - y_1)(x_3 y_4 - y_3 x_4) - (y_4 - y_3)(x_1 y_2 - y_1 x_2)}{(y_2 - y_1)(x_3 - x_4) - (y_4 - y_3)(x_1 - x_2)}$$

This is the final coordinate form of the intersection point of the two lines AB and CD.

Pseudocode: Line Segment Intersection

Algorithm: Orientation-Based Line Segment Intersection

Pseudocode

```
function doIntersect(A, B, C, D):
   o1 = orientation(A, B, C)
   o2 = orientation(A, B, D)
   o3 = orientation(C, D, A)
   o4 = orientation(C, D, B)

   if (o1 != o2) and (o3 != o4):
       return True   # Proper Intersection

   if (o1 == "collinear" and C lies on AB) or
      (o2 == "collinear" and D lies on AB) or
      (o3 == "collinear" and A lies on CD) or
      (o4 == "collinear" and B lies on CD):
      return True    # Overlapping segments

    return False     # No intersection
```

```
function orientation(P, Q, R):
    val = (Q.x - P.x)*(R.y - P.y)
        - (Q.y - P.y)*(R.x - P.x)
    if val == 0:
        return "collinear"
    else if val < 0:
        return "clockwise"
    else:
        return "counterclockwise"
```

Complexity Analysis: Single Pair of Segments

Time Complexity

- Each orientation computation takes $O(1)$ time.
- A constant number of orientation and comparison checks are performed.
- Therefore, total time complexity:

$$T(1) = O(1)$$

Space Complexity

- Only a few variables are stored (coordinates and orientation values).
- No additional data structures are required.
- Hence, space complexity:

$$S(1) = O(1)$$

Complexity Analysis: Multiple Line Segments

Time Complexity

- Each pair of segments must be checked for intersection.
- Total number of unique pairs:

$$\frac{n(n-1)}{2} = O(n^2)$$

- Therefore, overall time complexity: $T(n) = O(n^2)$

Space Complexity

- Orientation values and coordinates are reused.
- No extra memory grows with n: $S(n) = O(1)$

Bentley–Ottmann Algorithm: Overview

What is it?

- The **Bentley–Ottmann Algorithm** is an efficient method to find **all intersection points** among a set of n line segments.
- Unlike the pairwise test, it uses a **sweep line technique** to reduce comparisons.

Key Idea:

- Imagine a vertical line sweeping from left to right.
- The algorithm keeps track of which segments intersect this line (the "active" segments).
- Intersections are detected only when segments become neighbors along the sweep line.

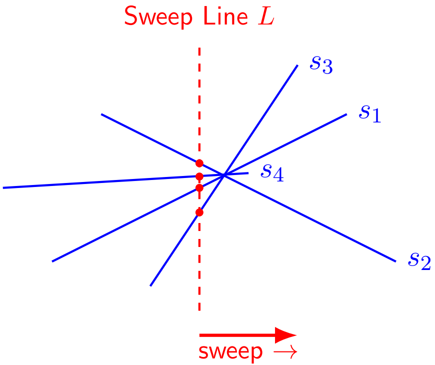
Working Principle

Main Steps:

1. **Event Points:** Collect all segment endpoints and potential intersections as events.
2. **Sweep Line:** Move a vertical sweep line from left to right across the plane.
3. **Status Structure:** Maintain an ordered structure (like a balanced BST) of "active" segments intersecting the sweep line, sorted by their y-coordinate.
4. **Neighbor Checking:** When two segments become adjacent, check if they intersect. If they do, record the intersection as a new event.

Repeat until all events are processed — at the end, all intersection points are found.

Main Idea: Sweep Line

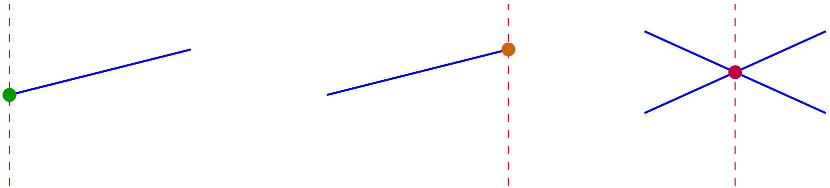


- Vertical line sweeps left to right
- Maintains segments intersecting the sweep line
- Processes events at endpoints and intersections

Data Structures

1. **Event Queue (Q):** Priority queue ordered by x-coordinate
   - Left endpoints of segments
   - Right endpoints of segments
   - Intersection points (discovered dynamically)
2. **Status Structure (T):** Balanced BST
   - Active segments intersecting sweep line
   - Ordered by y-coordinate of intersection with sweep line
   - Supports insert, delete, successor, predecessor in $O(\log n)$

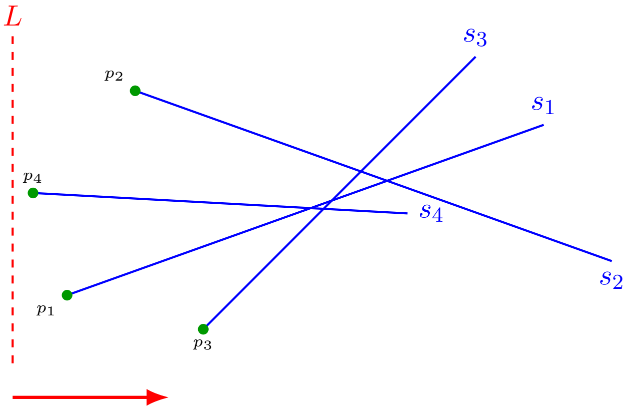
Event Types



Left Endpoint   Right Endpoint   Intersection

Each event type requires different handling to maintain invariants

Example: Initial Configuration



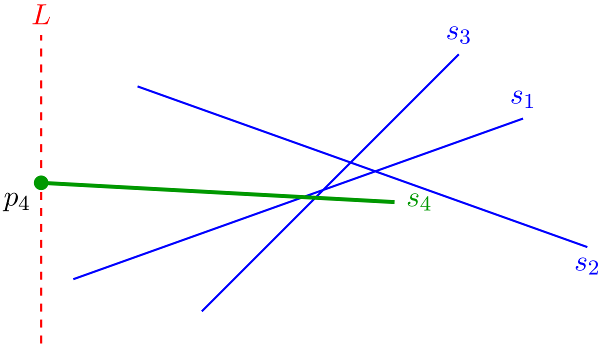
Event Queue:

$p_4, p_1, p_2, p_3, \ldots$

Status:

(empty)

Step 1: Process Left Endpoint p_4



Action:

Insert s_4 into status

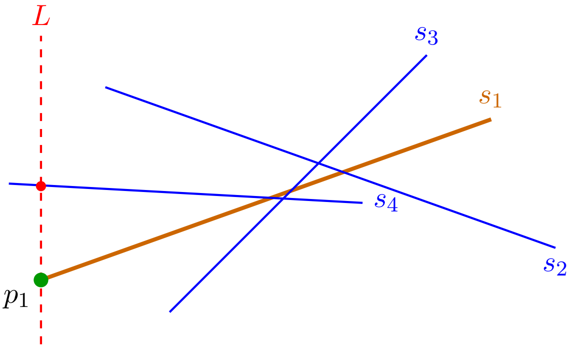
Status:

s_4

Check:

Above$(s_4) = \emptyset$
Below$(s_4) = \emptyset$
No new intersections

Step 2: Process Left Endpoint p_1



Action:

Insert s_1 into status

Status:

s_4 (top)

s_1 (bottom)

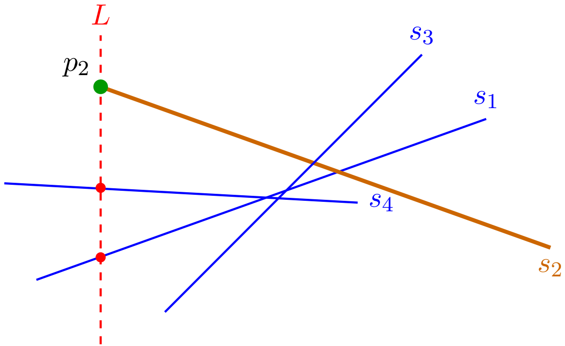
Check:

Above(s_1) $= s_4$

Check for intersection!

Add to event queue

Step 3: Process Left Endpoint p_2



Action:

Insert s_2 into status

Status (ordered):

s_2 (top)

s_4

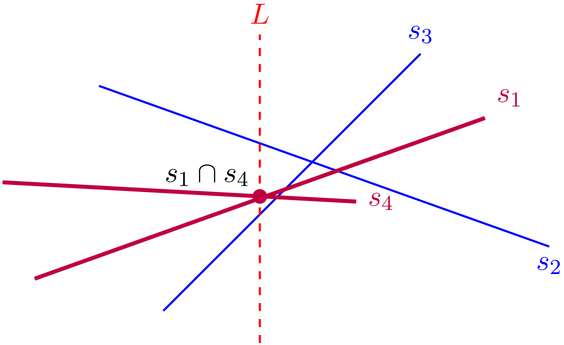
s_1 (bottom)

Check:

Check $s_2 \cap s_4$

Add intersections to Q

Step 4: Process Intersection Event



Action:

Report intersection

Swap s_1 and s_4 in status

Before swap:

s_2, s_4, s_1

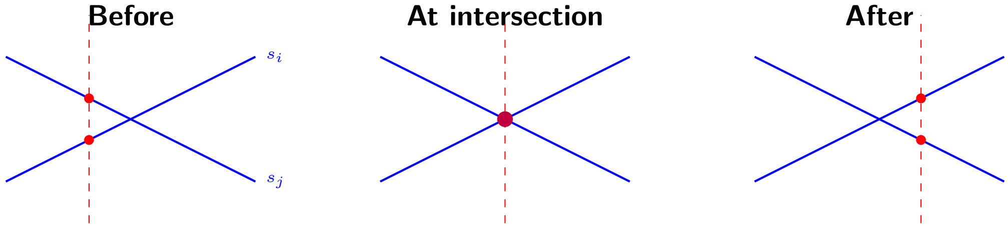
After swap:

s_2, s_1, s_4

Check new neighbors!

Key Idea: Order changes at intersections! Must check new adjacent pairs.

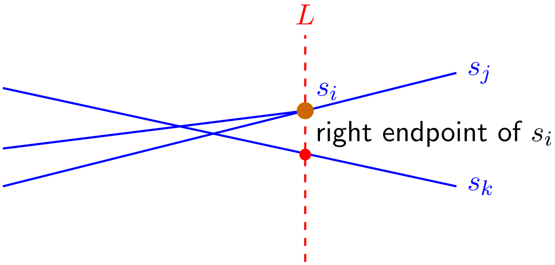
Handling Intersection Events



Status: (s_j, s_i) Report (s_i, s_j) Status: (s_i, s_j)

1. Report the intersection point
2. Swap the two segments in status structure
3. Check new neighbor above s_i for intersection
4. Check new neighbor below s_j for intersection

Handling Right Endpoint



Action:

1. Remove s_i from status
2. Find above$(s_i) = s_k$
3. Find below$(s_i) = s_j$
4. Check if $s_k \cap s_j$ exists (to the right of L)
5. If yes, add to event queue

When a segment ends, its former neighbors become adjacent!

Pseudocode: Bentley–Ottmann Algorithm

Algorithm

```
Input: Set S of n line segments
Output: All intersection points among segments in S

Initialize event queue Q with all segment endpoints
Initialize empty status structure T

while Q is not empty:
    event = Q.pop()       # smallest x-coordinate
    if event is a segment start:
        insert segment into T
        check intersection with neighbors in T
    else if event is a segment end:
        remove segment from T
        check intersection between former neighbors
    else if event is an intersection:
        record intersection point
        swap order of the two intersecting segments in T
        check for new neighbor intersections
```

Complexity Analysis

Let: n = number of segments, k = number of intersection points.

Time Complexity

- Each event (endpoint or intersection) is processed in $O(\log n)$ time.
- There are at most $O(n+k)$ events.
- Therefore, total time complexity:

$$T(n) = O((n+k)\log n)$$

Space Complexity

- Event queue Q: $O(n+k)$
- Status structure T: $O(n)$
- Overall:

$$S(n) = O(n+k)$$

Conclusion

Summary:

- The intersection of two line segments can be tested using orientation.
- If they intersect, the point is found using parameterized equations.
- For handling multiple line segments, the Bentley–Ottmann Algorithm is the standard solution.
- This approach is efficient and forms the basis of many computational geometry techniques.

References

- J. L. Bentley and T. Ottmann, "Algorithms for Reporting and Counting Geometric Intersections," *IEEE*, vol. C-28, no. 9, pp. 643–647, Sept. 1979. *DOI: 10.1109/tc.1979.1675432*
  Accessed on: Nov. 12, 2025.
- GeeksforGeeks, "Cramer's Rule," *GeeksforGeeks*, 2022.
  Link: `geeksforgeeks.org/maths/cramers-rule`
  Accessed on: Nov. 12, 2025.
- "Search for a Pair of Intersecting Segments - Algorithms for Competitive Programming," *CP-Algorithms*, 2025. *Link:* `cp-algorithms.com/geometry/intersecting_segments.html`
  Accessed on: Nov. 12, 2025.
- Wikipedia Contributors, "Cramer's Rule," *Wikipedia*, 2020.
  Link: `wikipedia.org/wiki/Cramer%27s_rule`.
  Accessed on: Nov. 12, 2025.
- Wikipedia Contributors, "Line–Line Intersection," *Wikipedia*, 2021.
  Link: `wikipedia.org/wiki/Line%E2%80%93line_intersection`.
  Accessed on: Nov. 12, 2025.

Thank You :)