COMPUTATIONAL GEOMETRY PROJECT REPORT

on

"Sweep-line Algorithm"

at

"Department of Mathematics, SPPU"

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY,
PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

MASTER OF SCIENCE

BY

Ms. **RAJI UTEKAR** (IMCA - II, Roll No. **23092029**)

Ms. **ARCHANA SONAWANE** (MSc - II, Roll No. **23091049**)

Mr. **AVINASH DESAI** (MSc - II, Roll No. **23091009**)

UNDER THE GUIDANCE OF
Prof. SMITA KANDEKAR

DEPARTMENT OF MATHEMATICS
SAVITRIBAI PHULE PUNE UNIVERSITY

DECLARATION OF STUDENTS

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources.

We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in our submission.

We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

MS. Raji Utekar	**MS. Archana Sonawane**	**MR. Avinash Desai**
IMCA - II	MSc - II	MSc - II
23092029	23091049	23091009

Date:- 09/11/2024

SAVITRIBAI PHULE PUNE UNIVERSITY
DEPARTMENT OF MATHEMATICS

CERTIFICATE

This is to certify that **MS. Raji Utekar** (IMCA - 2), **MS. Archana Sonawane** (MSc - 2) and **MR. Avinash Desai** (MSc - 2) from the Department of Mathematics, Savitribai Phule Pune University, Pune worked under the guidance of **Prof. Smita Kandekar** at Department of Mathematics, Savitribai Phule Pune University, Pune during 15/09/2024 to 09/11/2024 under Computational Geometry Project. Their performance was satisfactory/unsatisfactory. We award her _____ marks out of _____

Prof. (Dr.) Smita Kandekar
Mentor
Department of Mathematics
SPPU, Pune

Dr. Yashwant Borse
HOD
Department of Mathematics
SPPU, Pune

Date: 09/11/2024

Place: Pune

Acknowledgements

We take this opportunity to thank our project guide **Mr. Shubham Jagtap**, Alumni of Department of Mathematics, Savitribai Phule Pune University, Pune for providing us the opportunity to work in an exciting and challenging field of Computational Geometry. Our interactions with him has been of immense help in defining our project goals and in identifying ways to achieve them.

Our honorable mention goes to **Prof. (Dr.) Yashwant Borse**, Head of Department of Department of Mathematics, Savitribai Phule Pune University, Pune and **Prof. (Dr.) Smita Kandekar**, Professor of Department of Mathematics, for granting us the opportunity to get a project experience, for support and encouragement.

This is a great pleasure and immense satisfaction to express our deepest sense of gratitude and thanks to everyone who has directly or indirectly helped us in completing our Computational Geometry Project successfully.

Ms. Raji Utekar

Ms. Archana Sonawane

Mr. Avinash Desai

Abstract

Our Computational Geometry project, conducted by the department, explores the application of the sweep-line algorithm to detect and report intersections among line segments within a two-dimensional plane. Utilizing event-driven processing, the algorithm efficiently identifies intersecting segment pairs, which are essential in computational geometry fields such as computer graphics, geographic information systems, and collision detection.

Key challenges addressed include managing overlapping segments, handling shared endpoints, and optimizing data structures for performance improvements.

The report outlines the algorithm's design and implementation, details weekly progress across various development phases, and analyzes testing results on complex datasets. These insights demonstrate the algorithm's robustness, scalability, and potential applications in dynamic computational geometry environments.

Keywords:
slope, orientation, point, segment, event, vector, intersection, sweep line

Contents

Chapter 1
Computational Geometry Course

Computational Geometry is the study of algorithms to solve geometric problems, like finding the shortest paths, detecting intersections, and analyzing shapes. It has applications in fields like computer graphics, mapping, robotics, and data analysis.

This course conducted by Savitribai Phule Pune University introduces key techniques, such as the sweep-line algorithm for finding intersecting line segments and Voronoi diagrams for identifying nearest points. By learning these methods, students will gain skills in designing efficient solutions for real-world problems involving spatial data. This foundation in computational geometry will prepare students for roles in academia and industries that use mathematical and computational tools for spatial problem-solving.

This course requires a minimum of one semester. Each student registers for this course will be given **4 credits**.

Chapter 2
Prerequisites

2.1 Orientation of three ordered pairs

2.1.1 Problem Statement

Given three points a, b and c, the task is to determine the orientation of these three points.

2.1.2 Definition: Orientation

Orientation of an ordered triplet of points in the plane can be

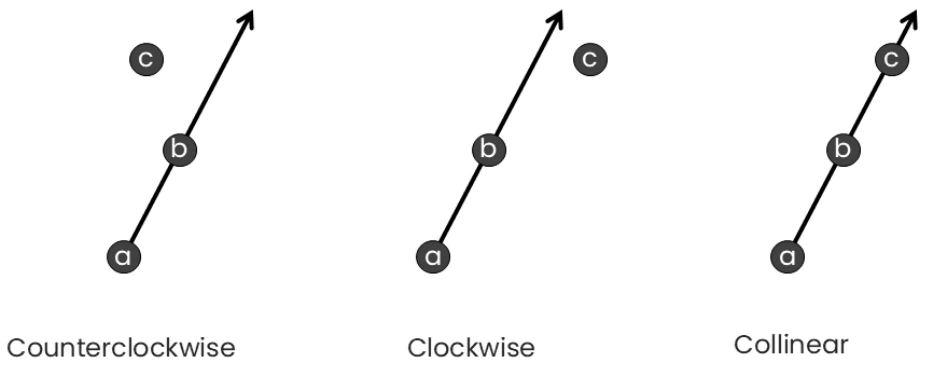

Figure 2.1: Orientation of three points a, b, c

Note:

- If orientation of (a, b, c) is collinear, then orientation of (c, b, a) is also collinear.
- If orientation of (a, b, c) is clockwise, then orientation of (c, b, a) is counterclockwise and vice versa is also true.

2.1.3 How to compute orientation?

The idea is to use the slope.

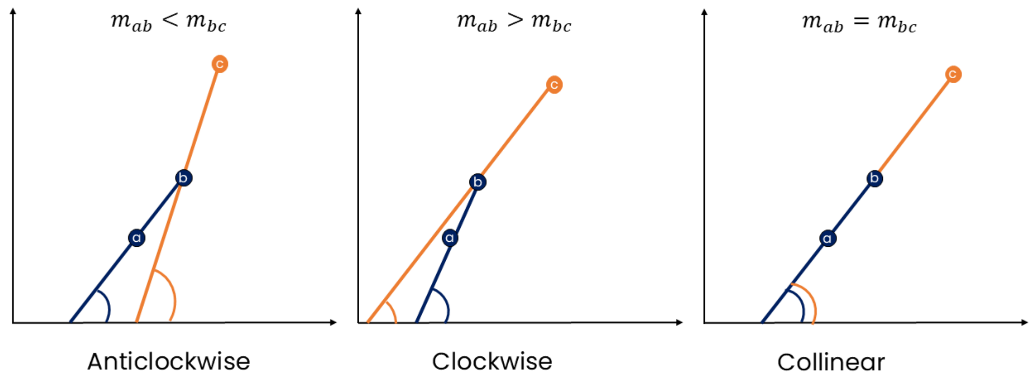

Figure 2.2: Orientation using slope

Here, slopes of ab and bc are given by:

$$m_{ab} = \frac{y_2 - y_1}{x_2 - x_1} \quad \& \quad m_{bc} = \frac{y_3 - y_2}{x_3 - x_2}$$

- **Anticlockwise:**

$$\begin{aligned} m_{ab} &< m_{bc} \\ \frac{y_2 - y_1}{x_2 - x_1} &< \frac{y_3 - y_2}{x_3 - x_2} \\ (y_2 - y_1) * (x_3 - x_2) &< (y_3 - y_2) * (x_2 - x_1) \\ (y_2 - y_1) * (x_3 - x_2) \; - \; (y_3 - y_2) * (x_2 - x_1) &< 0 \end{aligned}$$

- **Clockwise:**

$$\begin{aligned} m_{ab} &> m_{bc} \\ \frac{y_2 - y_1}{x_2 - x_1} &> \frac{y_3 - y_2}{x_3 - x_2} \\ (y_2 - y_1) * (x_3 - x_2) &> (y_3 - y_2) * (x_2 - x_1) \\ (y_2 - y_1) * (x_3 - x_2) \; - \; (y_3 - y_2) * (x_2 - x_1) &> 0 \end{aligned}$$

- **Collinear:**

$$\begin{aligned} m_{ab} &= m_{bc} \\ \frac{y_2 - y_1}{x_2 - x_1} &= \frac{y_3 - y_2}{x_3 - x_2} \\ (y_2 - y_1) * (x_3 - x_2) &= (y_3 - y_2) * (x_2 - x_1) \\ (y_2 - y_1) * (x_3 - x_2) &- (y_3 - y_2) * (x_2 - x_1) = 0 \end{aligned}$$

2.1.4 Algorithm

- Calculate slopes of line formed by points (a, b) and (b, c).
- If slopes are equal, then points are collinear
- If slope of (a,b) < slope of (b,c), then points are in counter clockwise orientation
- If slope of (a,b) > slope of (b,c), then points are in clockwise orientation

2.1.5 Code snippet:

```cpp
#include<iostream>
using namespace std;

//Function to check orientation
string Orientation(int p1[], int p2[], int p3[])
{
    int slope1 = (p2[1] - p1[1]) * (p3[0] - p2[0]);
    int slope2 = (p3[1] - p2[1]) * (p2[0] - p1[0]) ;

    if(slope1 - slope2 < 0)
        return "Anticlockwise";
    else if(slope1 - slope2 > 0)
        return "Clockwise";
    else
        return "Collinear";
}
int main(){
    int p1[] = { 0, 0 };
    int p2[] = { 4, 4 };
    int p3[] = { 1, 1 };
    cout << Orientation(p1, p2, p3) << endl;

    return 0;
}
```

2.2 Finding if two line segment intersects

2.2.1 Problem Statement

Given two line segments (p1, q1) and (p2, q2), find if the given line segments intersect with each other.

2.2.2 How orientation is useful here?

2.2.2.1 General Cases

The figure below shows general cases for segments' intersections:

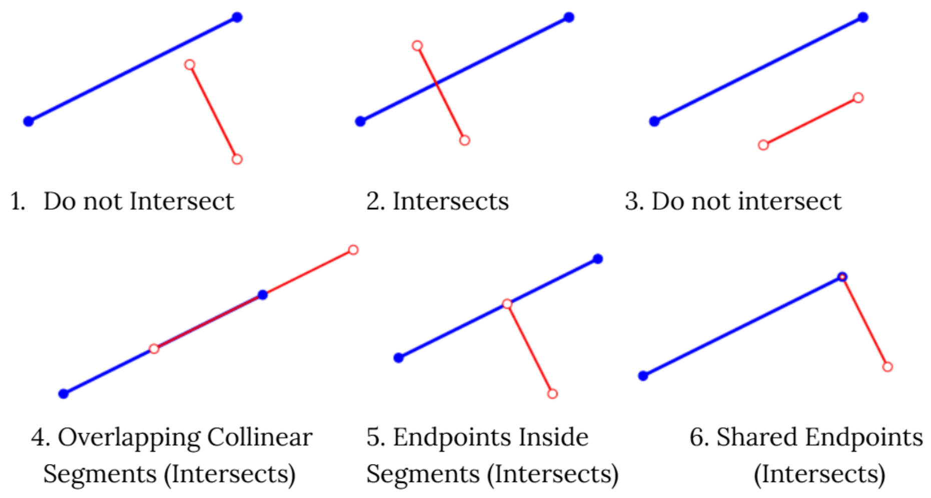
1. Do not Intersect
2. Intersects
3. Do not intersect
4. Overlapping Collinear Segments (Intersects)
5. Endpoints Inside Segments (Intersects)
6. Shared Endpoints (Intersects)

Figure 2.3: General cases

Here, we observe that two segments (p1, q1) and (p2, q2) intersect each other if following two conditions satisfies:

- Points (p1, q1, p2) and (p1, q1, q2) have different orientations.
- Points (p2, q2, p1) and (p2, q2, q1) have different orientations.

2.2.2.2 Special Cases

The figure below shows special cases for segments' intersections:

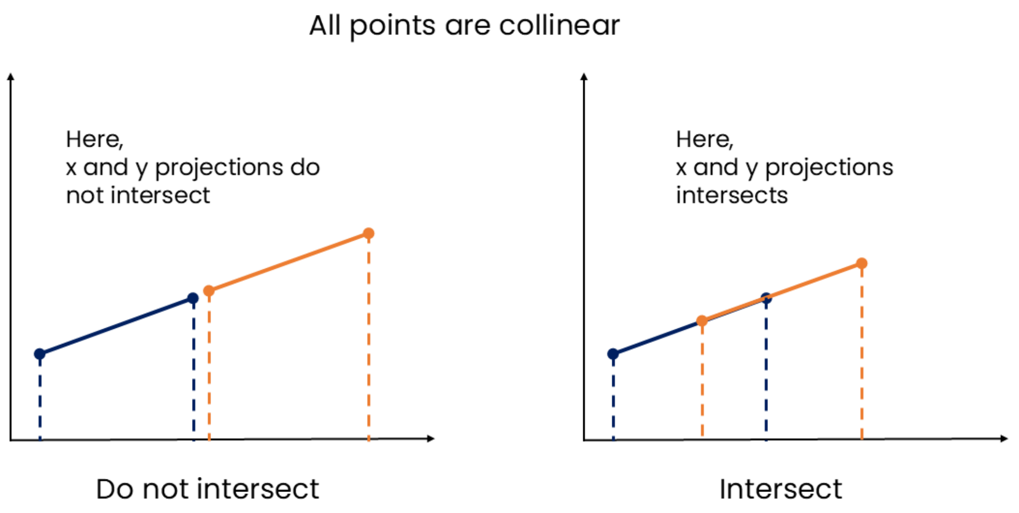

Figure 2.4: Special Cases

Here, we observe that two segments (p1, q1) and (p2, q2) intersect each other:

- Special Case:
  - If all points
    (p1, q1, p2),
    (p1, q1, q2),
    (p2, q2, p1),
    (p2, q2, q1)
    have same orientations i.e. they are parallel, then check if the x and y projections of (p1, q1) and (p2, q2) intersect

2.2.3 Code Snippet

```cpp
#include<iostream>
using namespace std;

//Struct Point
struct Point
{
    int x, y;
};

int orientation(Point p1, Point p2, Point p3)
{
    int slope = (p2.y - p1.y) * (p3.x - p2.x)
            - (p3.y - p2.y) * (p2.x - p1.x);

    if(slope == 0) //Collinear
        return 0;
    else if(slope > 0) //Clockwise
        return 1;
    else //Counter Clockwise
        return 2;
}

//Checks if Point B lies on line segment 'AC'
bool onSegment(Point A, Point B, Point C)
{
    if( (B.x <= max(A.x, C.x) && B.x >= min(A.x, B.x)) &&
        (B.y <= max(A.y, C.y) && B.y >= min(A.y, B.y)))
        return true;

    return false;
}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if(o1 != o2 && o3 != o4)
        return true;

    //Check if (p1, q1, p2) are collinear and p2 lies between line segment (p1, q1)
    if(o1 == 0 && onSegment(p1, p2, q1))
        return true;

    //Check if (p1, q1, q2) are collinear and q2 lies between line segment (p1, q1)
    if(o2 == 0 && onSegment(p1, q2, q1))
        return true;

    //Check if (p2, q2, p1) are collinear and q1 lies between line segment (p2, q2)
    if(o3 == 0 && onSegment(p2, q2, p1))
        return true;

    //Check if (p2, q2, q1) are collinear and q1 lies between line segment (p2, q2)
    if(o4 == 0 && onSegment(p2, q2, q1))
        return true;

    return false; //Doesn't fall in any above condition
}

int main(){
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};

    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {10, 0}, q1 = {0, 10};
    p2 = {0, 0}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {-5, -5}, q1 = {0, 0};
    p2 = {1, 1}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";
return 0;
}
```

Chapter 3
Intersection of n line segments Naive Approach

3.1 Problem Statement

Given n line segments (p1, q1), (p2, q2), ... ,(pn, qn), find if the given line segments intersect with each other or not.

3.2 Input

- First Line: Enter the number of line segments, **n**.
- Next n Lines: Each line contains the coordinates of the endpoints of a line segment, formatted as **x1 y1 x2 y2**, where (x1,y1) and (x2,y2) are the coordinates of the two endpoints of the line segment.

3.3 Output

List all intersecting pairs of line segments. Each intersection should be reported as a pair of segment indices, with the index starting from 1.

3.4 Example

For input:
4
1 1 4 4
2 1 2 5
3 0 3 3

4 2 6 2

The output might be:

1 2
1 3

3.5 Algorithm: `findIntersections`

Input: A vector `segments` of `Segment` objects, where each segment represents a line segment.

Output: Prints pairs of intersecting segments if any intersections are found; otherwise, prints a message indicating no intersections.

Working:

1. Initialize Variables:
   - Let `n` be the number of segments in the `segments` vector.
   - Set a boolean variable `hasIntersections` to `false`. This will be used to track if any intersections are found.
2. Check for Intersections:
   - For each segment `segments[i]` (loop variable `i` ranges from 0 to `n-1`
   - For each subsequent segment `segments[j]` (loop variable `j` ranges from `i+1` to `n-1`:
     - Call the `doIntersect()` with `segments[i]` and `segments[j]` as arguments to check if they intersect.
     - If `doIntersect()` returns `true`:
       - Print a message indicating that `Segment i+1` intersects with `Segement j+1`.
       - Set `hasIntersections` to `true`.
3. Final Output:
   If `hasIntersections` remains `false` after all pairs are checked, print "No Intersections found."

Complexity: This algorithm has a time complexity of $O(n^2)$ due to the nested loops that check all pairs of segments.

3.6 Code Snippet

```
#include<iostream>
#include <vector>
using namespace std;

//Point class having data members x and y
class Point
{
public:
  int x, y; //data members
  Point() //Default constructor
  {
      x = 0;  y = 0;
  }

  Point(int x, int y) //Parametrised constructor
  {
      this->x = x;  this->y = y;
  }
};

//Class for segment in 2D Plane
class Segment
{
public:
  //data members
  Point left, right; int id;

  Segment() //default constructor
        {
      left = { 0, 0 };
      right = { 0, 0 };
      id = 0;
  }

  Segment(Point left, Point right, int id) //parametrized
   constructor
  {
      this->left = left;
      this->right = right;
      this->id = id;
  }
};

// Function to find the orientation of ordered triplet (p, q,
    r)
int orientation(Point p, Point q, Point r)
```

```
{
  int slope = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y
   - q.y);
  if (slope == 0) //collinear
    return 0;
  return (slope > 0)
    ? 1 //clockwise
    : 2; //anticlockwise
}

// Function to check if a point q lies on segment pr
bool onSegment(Point p, Point q, Point r)
{
  return  q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
    q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y);
}

// Function that returns true if two segments 's1' and 's2'
   intersect
bool doIntersect(Segment s1, Segment s2)
{
  Point p1 = s1.left, q1 = s1.right; //segment s1 = segment
   p1q1
  Point p2 = s2.left, q2 = s2.right; //segment s2 = segment
   p2q2

  int o1 = orientation(p1, q1, p2);
  int o2 = orientation(p1, q1, q2);
  int o3 = orientation(p2, q2, p1);
  int o4 = orientation(p2, q2, q1);

  //General case
  if (o1 != o2 && o3 != o4)
      return true;

  //Special cases
  if (o1 == 0 && onSegment(p1, p2, q1))
      return true;
  if (o2 == 0 && onSegment(p1, q2, q1))
      return true;
  if (o3 == 0 && onSegment(p2, p1, q2))
        return true;
  if (o4 == 0 && onSegment(p2, q1, q2))
      return true;

  return false;
}

//Function that checks for any intersection between segments
```

```
void findIntersections(vector<Segment>& segments)
{
  int n = segments.size(); //returns the size of Segments
    vector
  bool hasIntersections = false;

  //Now to check all pairs of intersection
  for (int i = 0; i < n; i++){
      for (int j = i + 1; j < n; j++)
      {
      if (doIntersect(segments[i], segments[j]))
      {
            cout << "Segment " << i + 1 << " intersects with
    Segment " << j + 1 << "\n";
            hasIntersections = true;
      }
      }
  }

  if (!hasIntersections)
    cout << "No Intersections found." << endl;
}

int main()
{
  int n;
  cout << "Enter the number of segments: ";
  cin >> n;
  vector<Segment> segments;

  //Input the segments
  for (int i = 0; i < n; i++)
        {
      int x1, y1, x2, y2;

      cout << "Enter segment " << i + 1 << " coordinates (x1
    y1 x2 y2): ";
      cin >> x1 >> y1 >> x2 >> y2;

      segments.push_back(Segment(Point(x1, y1), Point(x2, y2)
    , (i+1)));
  }

  //Find and print intersecting segments
  findIntersections(segments);

  return 0;
}
```

Chapter 4
Sweep Line Algorithm

4.1 Introduction

The Sweep Line Algorithm is a fundamental computational geometry technique used to detect intersections between line segments in a 2D plane. By ”sweeping” a vertical line from left to right across the plane, the algorithm identifies and processes events where intersections may occur, efficiently finding overlapping or intersecting line segments. This method is widely used in computer graphics, GIS, and robotic path-finding due to its ability to handle a large number of segments with reduced computational complexity.

4.2 Problem Definition

4.2.1 Objective

Implement the sweep line algorithm to detect intersections between a set of line segments in a 2D plane. The algorithm should efficiently identify all intersecting pairs of line segments.

4.2.2 Input

1. **First Line:** Enter number of line segments, n
2. **Next n Lines:** Each line contains the coordinates of the endpoints of a line segment, formatted as x1 y1 x2 y2, where (x1,y1) and (x2,y2) are the coordinates of the two endpoints of the line segment.

4.2.3 Output

List all intersecting pairs of line segments. Each intersection should be reported as a pair of segment indices, with the index starting from 1.

4.2.4 Example

For input:

```
4
1 1 4 4
2 1 2 5
3 0 3 3
4 2 6 2
```

The output must be:

```
1 2
1 3
```

4.3 Weekly Tasks Assigned for Project

- **Week 1:** Implement the basic sweep line algorithm to detect intersections between line segments. Focus on correctly handling the event points where the sweep line intersects with segment endpoints.
- **Week 2:** Extend your implementation to handle complex cases such as overlapping line segments and segments that share endpoints. Ensure that all intersections are detected and reported accurately.
- **Week 3:** Optimize your algorithm to improve performance. Investigate and apply techniques to efficiently manage the status structure and handle a large number of line segments.
- **Week 4:** Test your implementation with a variety of complex input cases, including line segments that create multiple intersecting regions, to ensure robustness and accuracy.
- **Week 5:** Research and incorporate advanced features, such as handling special cases or optimizing the data structures used in the algorithm. Ensure that your implementation is both efficient and accurate.
- **Week 6:** Review and finalize your project. Make any necessary adjustments based on testing results, document your code and findings, and prepare for submission.

4.4 Sweep Line Algorithm to `findIntersections`

Input: A vector `segments` of `Segment` objects, where each segment represents a line segment.

Output: Prints pairs of intersecting segments if any intersections are found.

Working:

1. Initialize Variables:
    - Let `n` be the number of `segments` in the segments vector.
    - Create an empty vector `events` to store endpoint events of each segment.
2. Prepare Events for Sorting:
    - For each segment `segments[i]` (with index `i` from `0` to `n-1`):
        - Add an event for the left endpoint of `segments[i]`, marking it as a "left" event with `SegID = i`.
        - Add an event for the right endpoint of `segments[i]`, marking it as a "right" event with `SegID = i`.
    - Sort the `events` vector in ascending order based on the x-coordinates (and y-coordinates if x-coordinates are the same).
3. Sweep Line Algorithm to Find Intersections:

    (a) Create an empty vector `activeSegments` to maintain segments that are currently active in the sweep line.

    (b) For each event in `events` (loop index `i` from `0` to `2*n-1`):
    - Let `curr_index` be the segment index associated with the current event (`events[i].SegID`).
    - If the current event is a "left" endpoint:
        - Add this segment to `activeSegments`.
        - If `activeSegments` contains more than one segment:
            * For each segment in `activeSegments` before this newly added segment:
                · Check if the new segment and the segment being iterated over intersect using the `doIntersect()`.

        · If they intersect, print that `Segment it -> SegID + 1` intersects with `Segment curr_index + 1`.
- If the current event is a "right" endpoint:
    - Remove the segment with index `curr_index` from `activeSegments`.

Time Complexity: Sorting the events takes $O(nlogn)$, and processing each event within `activeSegments` involves iterating through active segments, making this approach efficient for finding intersections among multiple segments.

Note: The `doIntersect()` checks if two segments intersect based on their coordinates. This approach uses a sweep line technique with an active set to improve efficiency by only checking intersections among nearby segments.

4.5 Code implementation

```
#include <iostream>
#include <fstream> // For file handling
#include <vector>
#include <algorithm>
using namespace std;

//Point class having x and y coordinates
class Point
{
public:
    int x, y;
    Point()
    {
        x = 0;
        y = 0;
    }
    Point(int x, int y)
    {
        this->x = x;
        this->y = y;
    }
};

//Segment class with start and end point and Segment ID
class Segment
{
public:
    Point left, right;
    int id;
    Segment() {
        left = { 0, 0 };
        right = { 0, 0 };
        id = 0;
    }
    Segment(Point left, Point right, int id)
    {
        this->left = left;
        this->right = right;
        this->id = id;
    }
};

\\Event class
class Event
{
public:
    int x, y;
    bool isLeft;
    int SegID;

    Event() {
        x = 0;
        y = 0;
        isLeft = true;
        SegID = 0;
    }

    Event(int x, int y, bool isLeft, int segID)
    {
        this->x = x;
        this->y = y;
        this->isLeft = isLeft;
        this->SegID = segID;
    }

    bool operator<(const Event& other) const {
        if (x == other.x) return y < other.y;
        return x < other.x;
    }

};

// Function to find the orientation of ordered triplet (p, q,
    r)
int orientation(Point p, Point q, Point r)
{
    int slope = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.
    y - q.y);
    if (slope == 0)
        return 0;
    return (slope > 0) ? 1 : 2;
}

// Function to check if a point q lies on segment pr
bool onSegment(Point p, Point q, Point r)
{
    return q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y);
}

// Function that returns true if two segments 's1' and 's2'
    intersect
bool doIntersect(Segment s1, Segment s2)
{
    Point p1 = s1.left, q1 = s1.right;
    Point p2 = s2.left, q2 = s2.right;

    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false;
}

// Function that checks for any intersection between segments
void findIntersections(vector<Segment>& segments)
{
    int n = segments.size();
    vector<Event> events;

    for (int i = 0; i < n; i++)
    {
        events.push_back(Event(segments[i].left.x, segments[i
    ].left.y, true, i));
        events.push_back(Event(segments[i].right.x, segments[
    i].right.y, false, i));
    }
```

```cpp

    sort(events.begin(), events.end());

    vector<Event> activeSegments;

    for (int i = 0; i < 2 * n; i++)
    {
        int curr_index = events[i].SegID;
        if (events[i].isLeft) {
            activeSegments.push_back(events[i]);
            if (activeSegments.size() > 1) {
                for (auto it = activeSegments.rbegin() + 1; it != activeSegments.rend(); ++it)
                {
                    if (doIntersect(segments[curr_index], segments[it->SegID]))
                    {
                        cout << "Line " << (it->SegID) + 1 << " " << curr_index + 1 << " intersects" << endl;
                    }
                }
            }
        }
        else
        {
            vector<Event> ::iterator iter; //make an iterator iter to search in activeSegments vector
            for (iter = activeSegments.begin(); iter != activeSegments.end(); iter++)
            {
                if (iter->SegID == curr_index) //if iter's segID == curr_index
                {
                    activeSegments.erase(iter); //then erase that element from activeSegments by passing iter to it
                    break; //and break it once found
                }
            }
        }
    }
}

int main()
{
    ifstream inFile("input.txt"); // Open the input file
    if (!inFile) {
        cerr << "Error opening file!" << endl;
        return 1;
    }

    int n;
    inFile >> n; // Read the number of segments from the file
    vector<Segment> segments;

    for (int i = 0; i < n; i++)
    {
        int x1, y1, x2, y2;
        inFile >> x1 >> y1 >> x2 >> y2; // Read each segment's coordinates from the file
        segments.push_back(Segment(Point(x1, y1), Point(x2, y2), i + 1));
    }

    inFile.close(); // Close the file

    // Find and print intersecting segments
    findIntersections(segments);

    return 0;
}
```

4.6 Applications of Sweep Line Algorithm

Here are some of the key areas where it is used:

1. **Geographic Information Systems (GIS):**
    - **Line segment intersection:** Detecting intersections between roads, rivers, or other linear features.
    - **Polygon operations:** Performing operations like union, intersection, and difference on polygons representing land parcels, water bodies, or administrative boundaries.
    - **Map overlay:** Combining multiple layers of geographic data (e.g., terrain, land use, population density) to create comprehensive maps.
2. **Computer-Aided Design (CAD):**
    - **Collision detection:** Identifying overlapping objects or parts in a 3D model to prevent design errors.
    - **Boolean operations:** Performing operations like union, intersection, and difference on 3D shapes.

- **Layout generation:** Optimizing the placement of objects within a design, such as electrical components on a circuit board or furniture in a room.

3. **Robotics and Motion Planning:**
    - **Obstacle avoidance:** Planning collision-free paths for robots in environments with static or dynamic obstacles.
    - **Motion planning:** Generating smooth and efficient motion trajectories for robots performing tasks like assembly or manipulation.
4. **VLSI Design:**
    - **Circuit layout:** Optimizing the placement and routing of components on integrated circuits.
    - **Design rule checking:** Verifying that the design adheres to specific manufacturing constraints.
5. **Bioinformatics:**
    - **Genome analysis:** Identifying overlapping or intersecting regions in DNA sequences.
    - **Protein structure prediction:** Analyzing the spatial arrangement of amino acids in proteins.
6. **Image Processing:**
    - **Image segmentation:** Identifying and separating different objects or regions within an image.
    - **Feature extraction:** Extracting relevant features from images, such as edges, corners, or textures.
7. **Game Development:**
    - **Collision detection:** Detecting collisions between game objects, such as characters, projectiles, and obstacles.
    - **Physics simulation:** Simulating the physical behavior of objects in a game world.

These are just a few examples of the many real-world applications of the sweep line algorithm. Its versatility and efficiency make it a valuable tool for solving a wide range of geometric problems.

Chapter 5
Conclusion

5.1 Summary of Project Objective

In this project, we successfully developed an algorithm to detect intersections between line segments using a sweep line approach, implemented in C++ with the Standard Template Library (STL). The project focused on creating an efficient solution to a common computational geometry problem and involved the use of custom data structures and event-driven processing to optimize intersection detection.

5.2 Key Achievements

Key accomplishments included designing and implementing the code to handle segment events efficiently, leveraging sorting and active set management for optimized performance.

5.3 Challenges Overcome

In this project managing complex geometric conditions and optimizing it efficiently for large data sets was bit challenging for us. Through the use of Visual Studio's debugging tools, we were able to gain a detailed understanding of each step in the algorithm, allowing us to troubleshoot and refine our approach to ensure accurate results.

5.4 Skills Gained

1. **Algorithmic Thinking:** Learned to break down the intersection problem and apply a sweep line algorithm, optimizing for efficiency.

2. **Data Structures:**
   - Used `vectors` for dynamic storage and manipulation of events and active segments.
   - Applied `sorting techniques` to prepare event points in a specific order for efficient processing.
   - Utilized `iterators and STL algorithms` like `remove_if` for managing active segments.
3. **Coding in C++:**
   - Practiced advanced C++ programming skills, including working with STL containers and algorithms, lambda functions, and custom structures.
   - Improved debugging and code optimization, focusing on both time and space efficiency.
4. **Debugging and Development Tools:**
   Visual Studio: Gained proficiency in using Visual Studio as an IDE, especially utilizing its debugging tools to step through code execution. This helped identify logical errors, understand the control flow, and verify the behavior of complex conditions in real time.

References & Links

1. **YouTube Videos -**
   (a) (https://youtube.com/playlist?list=PL5DyztRVgtRVHE6fulR_q0W_pNd8JdMlb&si=i3haw80BKlA1gVoI)
   (b) (https://youtube.com/playlist?list=PL5DyztRVgtRVHE6fulR_q0W_pNd8JdMlb&si=i3haw80BKlA1gVoI)
   (c) (https://youtu.be/rEVlLMt-fwk?si=qvqgY127gbT_FGjK)
2. **Online articles-**
   (a) (https://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any
   (b) (https://web.mit.edu/urban_or_book/www/book/chapter7/7.2.4.html)
   (c) (https://cplusplus.com/reference/vector/vector/)