National University
of computer and emerging sciences

PROJECT REPORT

Course: Design & Analysis of Algorithm

Section: SE-5A

Group Members:

- **Ayaz Hasan** [20k-1044]
- **Hussain Tahir** [20k-0185]
- **Qazi Zain** [20k-1038]

Abstract

In the realm of algorithm design and analysis, our project explores the intricate world of Line Segments and Convex Hull computation. We delve into three distinct methods for line segment determination: via determinants, via geometric algorithms, and via gradients of line. Additionally, our study extends to the computation of Convex Hulls, a fundamental problem in computational geometry. By implementing and comparing these diverse approaches, we aim to provide insights into the efficiency, accuracy, and applicability of each method, contributing to the broader understanding of algorithmic techniques in geometric problem-solving.

1 Introduction

Our project focuses on the intricate domains of Line Segments and Convex Hull computation. For Line segments, we implement three methods: the determinant method, checkside method, and Bentley Ottmann algorithm.

Determinant Method

The Determinant Method works on the algebraic method to find the intersection point of two lines. It uses the - determinant formula for finding the intersection point of two lines in a Cartesian coordinate system. The method returns the intersection point as (x, y) if the lines intersect, or None if they are parallel or coincident.

Checkside Method

Checkside checks if the endpoints of two line segments are on different sides of the other line. It uses the orientation function to determine the orientation of three points (clockwise, counterclockwise, or collinear). If the orientations of the endpoints of one line segment with respect to the other line are different, it implies intersection.

Bentley–Ottmann Algorithm

The Bentley–Ottmann algorithm uses a sweep line approach and efficiently handles events associated with line segment endpoints to determine all intersections among a set of line segments.

Convex Hull

The convex hull, a core concept in computational geometry, defines the smallest convex shape encompassing a set of points. Widely employed in computer graphics, robotics, and spatial analysis algorithms, its efficient computation is critical for optimizing solutions in fields reliant on precise geometric relationships.

2 Programming Design

Language and Libraries

Python: Chosen for readability and extensive libraries. Tkinter: Used for a platform-independent GUI. Matplotlib: Enables dynamic visualizations.

GUI

Tkinter creates an interactive interface where users input points for Line Segments and Convex Hull. User-friendly components include labels, buttons, and interactive Matplotlib plots.

Algorithms

Line Segments

Determinant, geometric algorithms, and Bentley Ottmann.

Convex Hull

Implements Brute Force, Jarvis March, Graham Scan, and Quick Elimination, Melkman for diverse approaches.

Development Environment

Visual Studio Code provides a feature-rich Python development environment.

3 Experimental Setup

Input Generation

- Random Points
- User-Provided Points

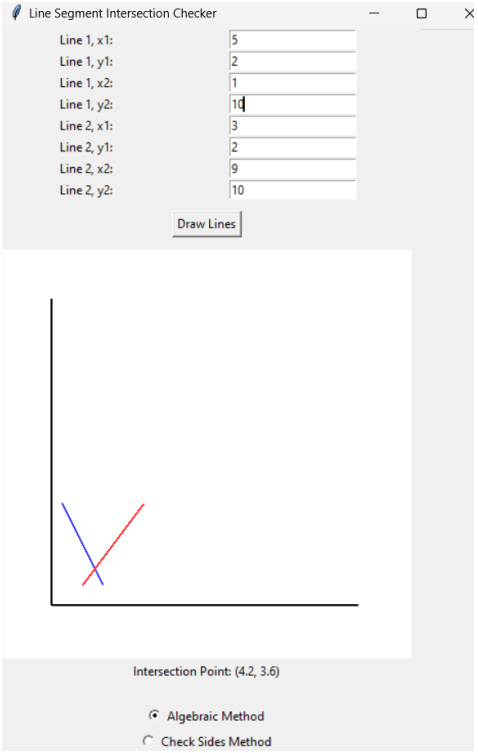
Performance Metrics

- Execution Time
- Space Complexity

4 Results and Discussion

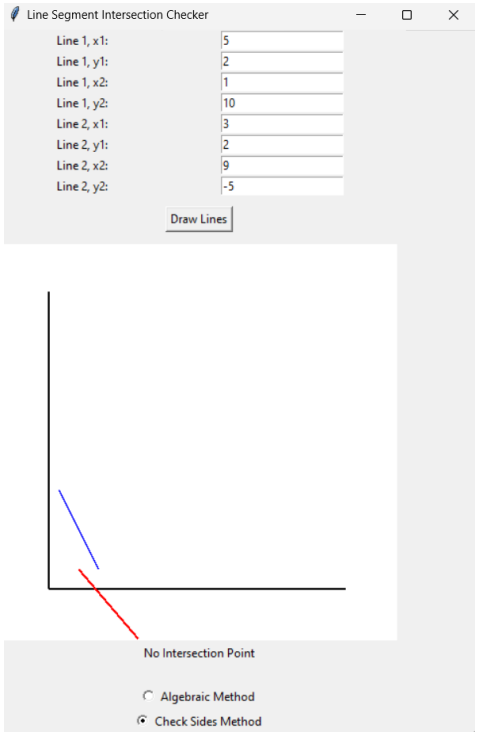
Line Segments

1) Determinant Method



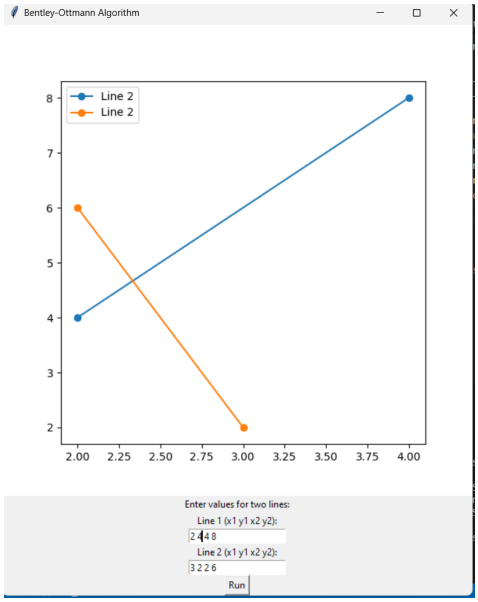
Using the determinant, the code calculates the intersection point (intersection_x, intersection_y) using the formulas derived from the algebraic method.

2) Checkside Method



The `check_sides` method determines line segment intersection by evaluating the orientation of endpoints using the cross product. If endpoints are on opposite sides of the other line, it implies an intersection. Special cases account for collinear points. The GUI displays whether lines intersect or not based on this method.

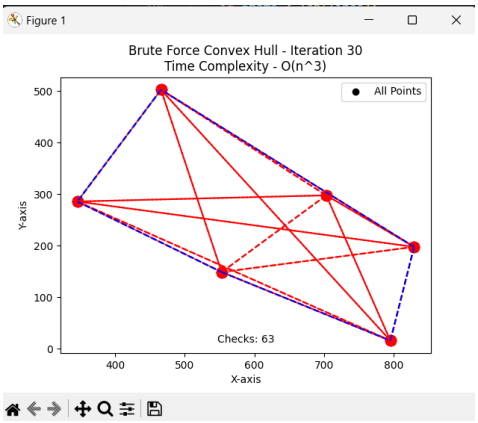
3) Bentley Ottmann Method



When the user draws two lines, it calculates the orientation of their endpoints. Based on this orientation, it checks if the lines intersect. The result is then displayed on the canvas. The algorithm considers collinear points and utilizes on-segment checks to accurately determine intersection.

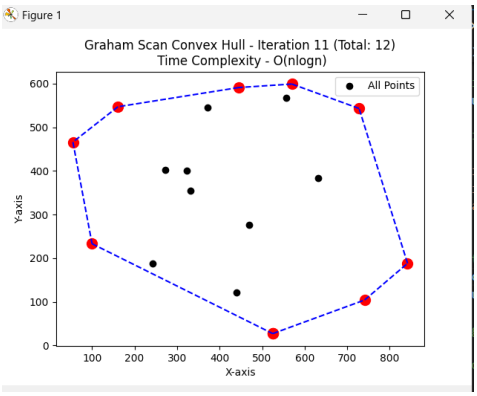
Convex Hull

1) Brute Force Method



Brute-force approach compute the convex hull of a set of randomly generated points. The algorithm iterates through all point combinations, checking their order using the Counter Clock Wise (ccw) function. The animated visualization illustrates the step-by-step construction of the convex hull, connecting valid points in each iteration and providing a dynamic representation of the algorithm's progression.

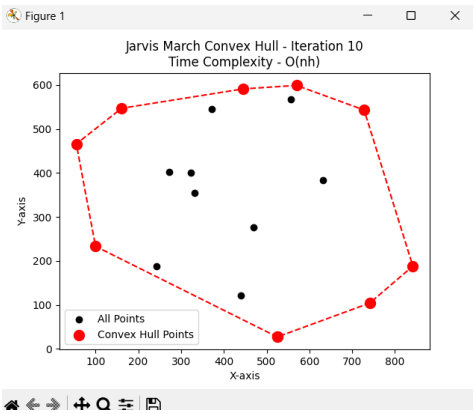
2) Graham Scan Method



Graham Scan algorithm compute the convex hull of

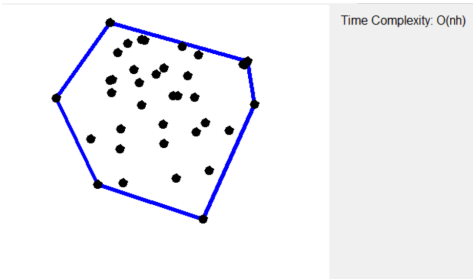
randomly generated points. It begins by selecting the lowest point and sorts the rest based on polar angles. The animated display visually illustrates the sequential construction of the convex hull, emphasizing the algorithm's efficiency with a time complexity of O(n log n).

3) Jarvis March Method



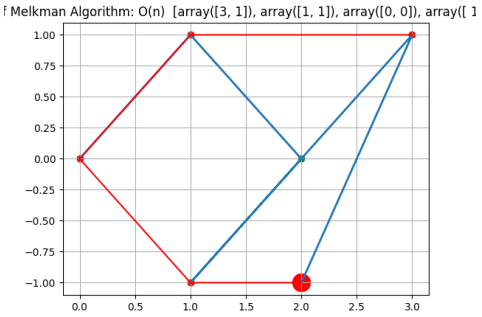
Jarvis March algorithm compute the convex hull of randomly generated points. Starting with the lowest point, the algorithm iteratively selects the next point with the smallest polar angle. The animated display visually depicts the stepwise construction of the convex hull, with the red points and connecting dashed lines representing the evolving convex hull. The time complexity of the Jarvis March algorithm is O(nh), where n is the number of points, and h is the number of convex hull points.

4) Quick Elimination Method



This Tkinter-based Python code allows users to interactively place points on a canvas using left-clicks. Upon right-clicking, the Quick Elimination convex hull algorithm is applied to compute and visualize the convex hull, displayed as blue lines. The time complexity of the algorithm is dynamically shown, providing users with a hands-on understanding of convex hull construction efficiency. Middle-clicking resets the canvas for further experimentation.

5) MelkMan Method



Melkman Convex Hull algorithm, visualizing each step of the process. It initializes the convex hull with the first three points, and then iteratively adds and removes points to maintain convexity. The end result is a visual representation of the convex hull formed by the input points. The time complexity of the Melkman algorithm is O(n), providing an efficient solution for constructing convex hulls.

5 Conclusion

In conclusion, our project has delved into the intricate realm of Line Segments and Convex Hull computation, exploring three distinct methods for line segment determination: the determinant method, checkside method, and Bentley–Ottmann algorithm. Through the implementation and comparison of these diverse approaches, we aimed to provide insights into their efficiency, accuracy, and applicability in the context of geometric problem-solving.Venturing beyond Line Segments, we

set sail into the sea of Convex Hull, an indispensable challenge in computational geometry. Our arsenal includes renowned algorithms Jarvis Marsh,Graham Scan and Quick ELimination techniques andalso the Melkman's Algorithm.

6 Conclusion

In conclusion, our project has delved into the intricate realm of Line Segments and Convex Hull computation, exploring three distinct methods for line segment determination: the determinant method, checkside method, and Bentley–Ottmann algorithm. Through the implementation and comparison of these diverse approaches, we aimed to provide insights into their efficiency, accuracy, and applicability in the context of geometric problem-solving. Venturing beyond Line Segments, we set sail into the sea of Convex Hull, an indispensable challenge in computational geometry. Our arsenal includes renowned algorithms Jarvis Marsh, Graham Scan, and Quick Elimination techniques, and also the Melkman's Algorithm.

References

- https://iq.opengenus.org/2d-line-intersection/
- https://www.hackerearth.com/practice/math/geometry/line-intersection-using-bentley-ottmann-algorithm/tutorial/
- https://core.ac.uk/display/82805947?utm$_source$ = $pdf utm_medium$ = $bannerutm_campaign$ = $pdf - decoration - v1$

Appendix

Appendix A: Glossary

GUI: Graphical User Interface. https://docs.python.org/3/library/tkinter.html

OS: Operating System. Supported Window/Linux Operating System

Appendix B: To Be Determined List

To be determined.

- Link to the GitHub repository for the list: https://github.com/ayaz892/Fast-SE-Resources-/tree/main/Semester5/ALGORITHM/PROJECT