Convex Hull and Line Intersection Algorithms

Muhammad Owais Hassan Siddiqi - 21K3402
Usman Arif - 21K3448
Sania Hasan - 21K4671

November 27, 2023

Abstract

This project explores the implementation and comparison of convex hull algorithms (Brute Force, Graham Scan, Jarvis March, Quick Hull, and Monotone Chain) and a line intersection algorithm (Slope, Parametric, and CCW). The report discusses the programming design, experimental setup, and presents results, including visualizations and execution time comparisons.

1 Introduction

Convex hull and line intersection algorithms are fundamental in computational geometry. The convex hull algorithms aim to find the minimal convex polygon enclosing a set of points, while line intersection algorithms determine whether two lines intersect. These algorithms have applications in various fields, such as computer graphics, robotics, and geographic information systems.

2 Your Programming Design

The project is implemented using Python due to its readability and extensive libraries for visualization.

2.1 Brute Force Convex Hull Algorithm

The Brute Force convex hull algorithm (`brute_force_convex_hull`) checks all possible combinations of points to find the convex hull. It iterates through each point pair and determines whether all other points lie on one side of the line formed by the pair.

2.2 Graham Scan Convex Hull Algorithm

The Graham Scan algorithm (`graham_scan`) sorts the points based on polar angles and constructs the convex hull incrementally. It starts with the point with the lowest y-coordinate (and leftmost if ties) as the pivot and sorts the remaining points based on their polar angles.

2.3 Jarvis March Convex Hull Algorithm

The Jarvis March algorithm (`jarvis_march`) begins with the bottommost point and iteratively selects the point with the smallest polar angle. This process continues until the convex hull is formed.

2.4 Quick Hull Convex Hull Algorithm

The Quick Hull algorithm (`quick_hull`) is employed for computing the Convex Hull of a set of points. This algorithm follows a divide-and-conquer approach and is known for its efficiency.

2.5 Monotone Chain Convex Hull Algorithm

The Monotone Chain algorithm (`monotone_chain`) is employed for computing the Convex Hull of a set of points. This algorithm falls under the category of convex hull algorithms and aims to find the boundary vertices that form the convex polygon encapsulating the given set of points.

2.5.1 Algorithm Overview

The Monotone Chain algorithm proceeds as follows:

1. **Sort Points:** Sort the input points lexicographically (first by x coordinate, and then by y coordinate).
2. **Build Lower Hull:** Traverse the sorted points to construct the lower hull of the convex polygon. This is achieved by iteratively adding points while maintaining convexity.
3. **Build Upper Hull:** Reverse the order of the sorted points and repeat the process to build the upper hull.
4. **Combine Hulls:** Combine the lower and upper hulls to obtain the complete convex hull.

2.6 Line Intersection Algorithm

The Line Intersection algorithm (`parametric_intersect`) determines whether two given lines intersect. It uses parametric equations and slope calculations to check for intersection points within the line segments.

2.7 System Diagram

The system diagram, shown in Figure 1, illustrates the components of the project. The main components include the graphical user interface (GUI) for inputting points, the algorithms for computing convex hulls, and the line intersection checker.

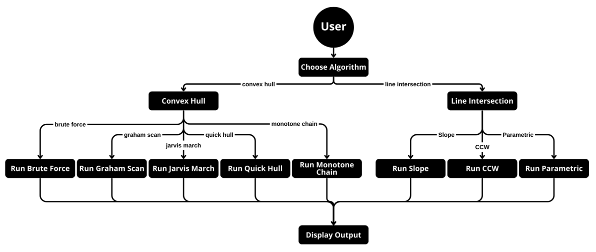


Figure 1: System Diagram

3 Experimental Setup

The project provides a user-friendly interface that allows users to choose between Convex Hull and Line Intersection algorithms. Users can further select specific algorithms within these categories. The experimental setup involves the following steps:

3.1 Algorithm Selection

- **Convex Hull:** Users can choose from a variety of convex hull algorithms, including Brute Force, Graham Scan, Jarvis March, Quick Hull, and Monotone Chain.
- **Line Intersection:** Users can opt for either the Parametric, Slope-based or CCW Line Intersection algorithm.

3.2 Point Input

Users input sets of points through the GUI. For Convex Hull experiments, these points are used as input to compute the convex hull. For Line Intersection experiments, two lines are drawn based on user input.

3.3 Algorithm Execution

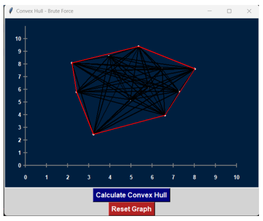
The selected algorithm is executed based on the user's choice. The system runs the chosen algorithm with the provided input to compute the convex hull or check for line intersection.

4 Results and Discussion

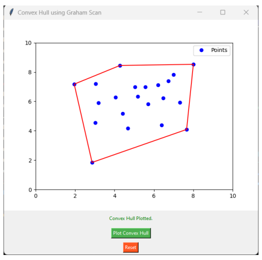
The project generates visualizations of the convex hulls and determines line intersection. Figure 2 shows sample screenshots of the project output.

4.1 Execution Time Comparison

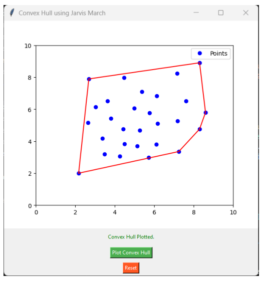
Table 1 displays the time and space complexities of the convex hull algorithms.



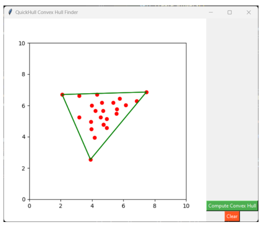
(a) Brute Force



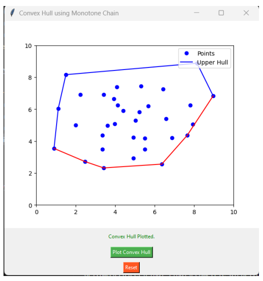
(b) Graham Scan



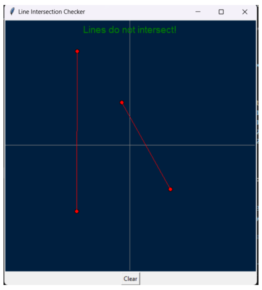
(c) Jarvis March



(d) Quick Hull

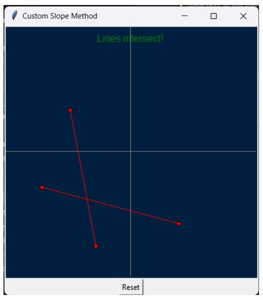


(e) Monotone Chain



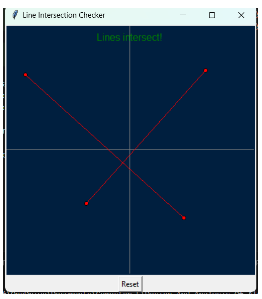
Lines do not intersect!

(f) Line Intersection (Para)



Lines intersect!

(g) Line Intersection (Slope)



Lines intersect!

(h) Line Intersection (CCW)

Figure 2: Sample Result Screenshots

Table 1: Algorithmic Analysis

Algorithm	Time Complexity	Space Complexity
Brute Force	$\Theta(n^3)$	$\Theta(n)$
Jarvis March	$\Theta(nh)$	$\Theta(1)$
Graham Scan	$\Theta(n \log n)$	$\Theta(n)$
Quick Hull	$\Theta(n \log n)$	$\Theta(n)$
Monotone Chain	$\Theta(n \log n)$	$\Theta(n)$

5 Conclusion

The project successfully implements and compares convex hull algorithms, revealing varying execution times with Graham Scan demonstrating superior efficiency. The line intersection algorithm accurately determines intersections.

References

[1] Tkinter Documentation, https://docs.python.org/3/library/tkinter.html

[2] Matplotlib Documentation, https://matplotlib.org/stable/contents.html

[3] Monotone Chain, https://iq.opengenus.org/monotone-chain-algorithm