Project Report: Line Intersection and Convex Hull Algorithms

Ameer Hamza, Aneeq Muneer, and Dinesh Dhanjee

November 27, 2023

1 Abstract

The project presents the implementation of three-line intersection algorithms and five convex hull algorithms within a web application developed using Django, HTML, CSS, and JavaScript. The application allows users to input points manually, read from a file, or generate random points. The algorithms are then executed and visualized on the website. The report discusses the programming design, experimental setup, results, and conclusions drawn from the project.

2 Introduction

The project aims to explore and implement various line intersection and convex hull algorithms. Line intersection algorithms are essential in computer graphics, computational geometry, and other applications. Convex hull algorithms find applications in pattern recognition, image processing, and geographical information systems. The web-based implementation provides an interactive platform for users to visualize and understand these algorithms.

3 Background

3.1 Convex Hull Problem

The Convex Hull problem stands as a foundational concept in computational geometry, focusing on determining the smallest convex polygon that encompasses a given set of points within the Euclidean plane. More precisely, it is the intersection of all convex sets containing the provided points. Convex hulls find widespread applications in computer graphics, pattern recognition, and robotics.

3.2 Convex Hull Problem

The challenge of the Line Intersection problem revolves around establishing whether and where multiple line segments intersect within the Euclidean plane. This problem holds crucial significance in fields like computer graphics, geographic information systems, and computational geometry.

When presented with a set of line segments denoted as L = l1, l2, . . . , lm, where each segment is defined by two endpoints, the objective is to detect all points of intersection among these segments. An intersection is recognized when two line segments share a common point.

The Line Intersection problem finds applications in diverse domains, including robotics path planning, computer-aided design, and computational biology. Algorithms are designed to offer accurate and swift solutions for pinpointing intersections within a given set of line segments.

4 Programming Design

In our pursuit of solutions for the Convex Hull and Line Intersection problems, we opted to employ the Django Web development framework with HTML, CSS, and JavaScript and designed algorithms in JavaScript programming language. We chose Django for its scalability, versatility, and the availability of endless resources catering to mathematical computations and geometry.

4.1 Choice of Programming Language

JavaScript was chosen as the programming language due to its simplicity, ease of comprehension, and the extensive array of libraries, especially those e.g. "plotly.js"

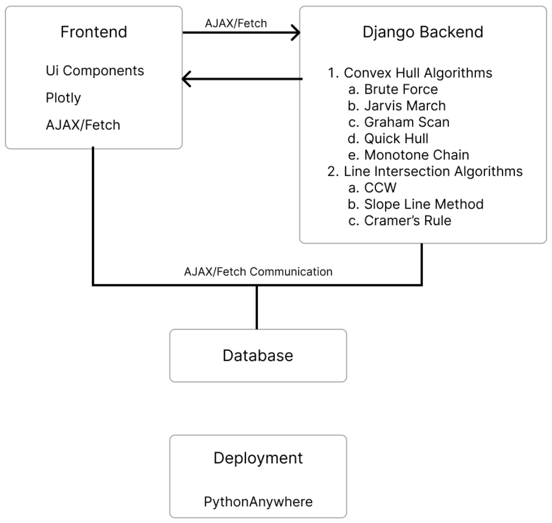

Figure 1: System Design Diagram

pertaining to computational geometry. JavaScript's expressive syntax enables a succinct and lucid implementation of intricate algorithms, contributing to improved code readability and ease of maintenance.

4.2 System Design Diagram

The system design is illustrated in the diagram below, outlining the key components and their interactions: [System Design Diagram]

4.3 Convex Hull Module

Within the Convex Hull module, we have implemented functions dedicated to solving the Convex Hull problem. This module encompasses well-known algorithms like Graham's scan, Jarvis march, Quick Hull, Brute Force, and Monotone Chain. It is designed to offer a streamlined interface for calculating the convex hull for a specified set of points.

4.4 Line Intersection Module

The Line Intersection module is tailored to address the Line Intersection problem and integrates algorithms such as the CCW, Slope line method, and Cramer's Rule. This module provides methods for identifying and reporting intersections within a set of line segments.

5 Experimental Setup

We tested the algorithms on a computer with an Intel Core i5 5th Gen processor, 8GB of RAM, and running JavaScript on a Windows system. JavaScript was chosen for its ease of use and the availability of helpful libraries for our computations. We tested on different browsers e.g. Google, Edge etc.

6 Results and Discussion

We examined the performance of four distinct algorithms—Jarvis March, Graham Scan, Quick Hull, and Brute Force—and conducted a comparative analysis of their speed. Our observations are summarized as follows:

1. **Jarvis March:** Known for its simplicity, this algorithm performed adequately on smaller datasets. It operates with a time complexity of $O(n^2)$, where n represents the number of input points. The space complexity is $O(1)$, indicating memory efficiency.
2. **Graham Scan:** Demonstrating robust performance, especially with datasets containing a significant number of points sharing the same y-coordinate, the Graham Scan algorithm has a time complexity of $O(nlogn)$ and a space complexity of $O(n)$, making it efficient for moderate-sized datasets.
3. **Quick Hull:** Notably swift, Quick Hull exhibited impressive speed, particularly on larger datasets. It boasts an average time complexity of $O(nlogn)$ and a worst-case time complexity of $O(n^2)$. The space complexity is $O(n)$, indicating memory efficiency.
4. **Brute Force:** Despite its straightforward approach, the Brute Force algorithm proved inefficient on larger datasets, operating with time complexity of

$O(n^3)$ and space complexity of $O(1)$, where n is the number of input points.

Our discussion aims to elucidate these results, facilitating the decision-making process by considering factors such as algorithm speed and its effectiveness with different data types. These insights provide a clearer understanding of when to deploy each algorithm, aiding others in choosing the most suitable one for their specific requirements.

7 Conclusion

In summary, our investigation brought to light the strengths and weaknesses inherent in various convex hull algorithms. To provide a succinct overview:

1. The simplicity of the Jarvis March algorithm is offset by its diminishing efficiency when applied to larger datasets.
2. The Graham Scan algorithm demonstrates commendable performance, particularly with datasets featuring consistent y-coordinates.
3. Quick Hull distinguishes itself by delivering robust performance on larger datasets, showcasing its versatility.
4. Although straightforward, the Brute Force algorithm proves less efficient when confronted with larger datasets.

In the majority of scenarios, Quick Hull emerged as the optimal choice, striking a balance between efficiency and adaptability. The decision ultimately hinges on specific characteristics of the data, offering practical insights for algorithm selection based on factors such as dataset size, distribution, and computational efficiency.

8 References

1. Django[1]
2. Plotly[2]
3. Monotone Chain[3]
4. Cramer's Rule[4]

[1] https://www.djangoproject.com/

[2] https://plotly.com/

[3] https://scholarworks.calstate.edu/downloads/2z10ww05b

[4] https://www.researchgate.net/publication/255981900$_{A}comparison_{b}etween_{C}ramer's_{}$
$Elimination_{M}ethod_{t}o_{s}olve_{s}ystems_{o}f_{t}hree_{o}r_{m}ore_{l}inear_{e}quations$

9 Appendix - Experimental Screenshots

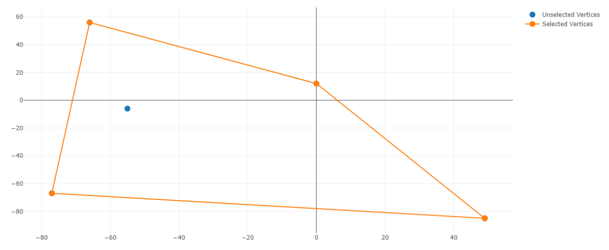

Figure 2: Brute Force

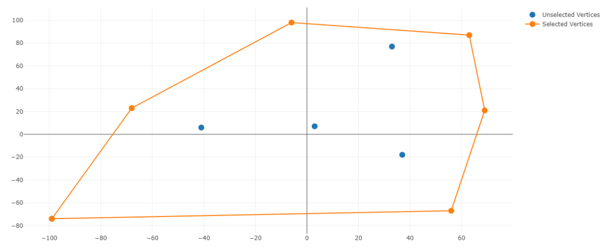

Figure 3: Jarvis March

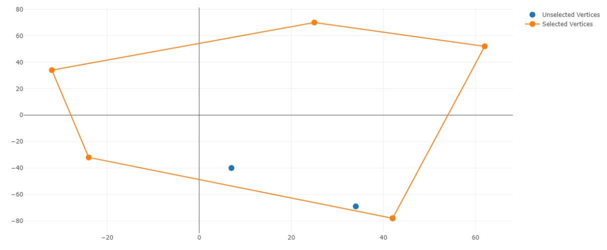

Figure 4: Graham Scan

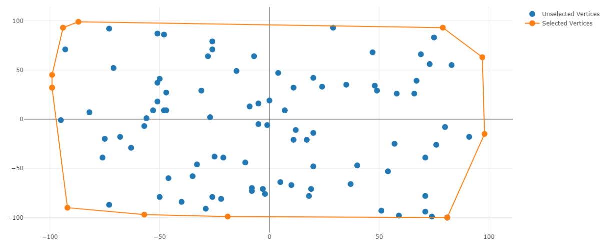

Figure 5: Quick Hull

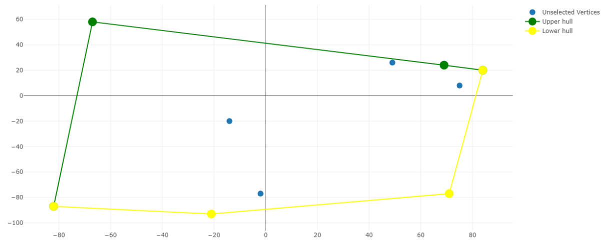

Figure 6: Monotone Chain

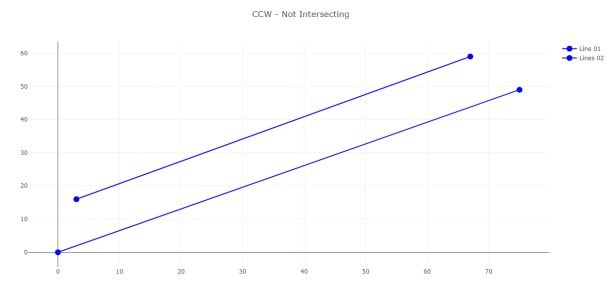

Figure 7: CCW - Not Intersecting

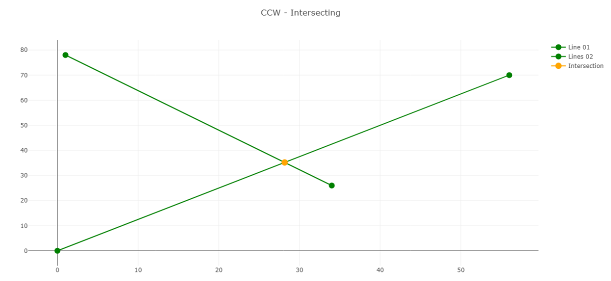

Figure 8: CCW - Intersecting

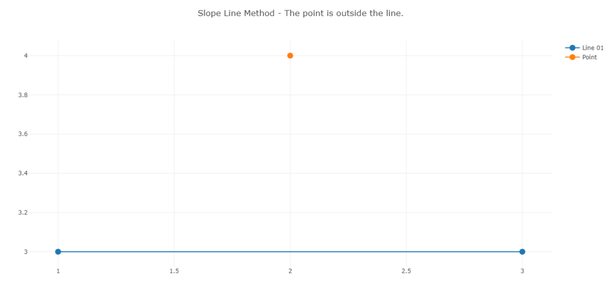

Figure 9: Slope Line Method

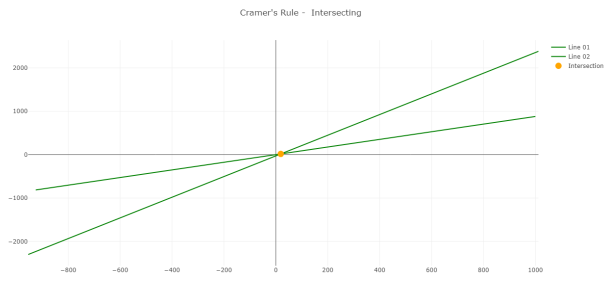

Figure 10: Cramer's Rule