Design and Analysis of Geometric Algorithms

Ashad Abdullah Qureshi

November 20, 2023

Abstract

This report discusses the implementation and analysis of various geometric algorithms for constructing convex hulls and detecting line intersections.

1 Introduction

This project explores the computational geometry domain, focusing on the design and analysis of algorithms that find convex hulls and line intersections.

2 Background

2.1 Convex Hull Problem

The Convex Hull problem is a fundamental concept in computational geometry. Given a set of points in the Euclidean plane, the convex hull is the smallest convex polygon that encloses all the points. Formally, it is the intersection of all convex sets containing the given points. Convex hulls have widespread applications in areas such as computer graphics, pattern recognition, and robotics.

The problem can be formally stated as follows: Given a set of points $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, find the convex hull, denoted as $CH(P)$, which is the convex polygon formed by the subset of points.

Various algorithms exist to compute the convex hull, such as Graham's scan, Jarvis march, and QuickHull. These algorithms differ in their efficiency and approach but share the common goal of determining the convex hull in a time complexity that is optimal or near-optimal.

2.2 Line Intersection Problem

The Line Intersection problem involves determining if and where two or more line segments intersect in the Euclidean plane. This problem is crucial in computer graphics, geographic information systems, and computational geometry.

Given a set of line segments $L = \{l_1, l_2, \ldots, l_m\}$, where each line segment is defined by two endpoints, the goal is to identify all intersections between these line segments. An intersection occurs when two line segments share a common point.

The Line Intersection problem has applications in various fields, including robotics path planning, computer-aided design, and computational biology.

Efficient algorithms, such as the sweep line algorithm and Brute Force, have been developed to address the Line Intersection problem. These algorithms aim to provide accurate and fast solutions for identifying intersecting line segments within a given set.

In the following sections, we will delve into specific algorithms and methodologies employed to solve the Convex Hull and Line Intersection problems.

3 Programming Design

In developing solutions for the Convex Hull and Line Intersection problems, we chose to implement the algorithms using the Python programming language, leveraging its readability, versatility, and extensive libraries for mathematical computations and geometry.

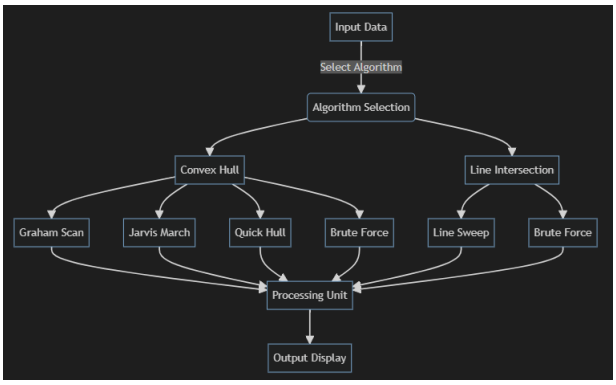

Figure 1: Enter Caption

3.1 Choice of Programming Language

Python was selected as the programming language for its simplicity, ease of understanding, and a rich ecosystem of libraries, particularly those related to computational geometry. The expressive syntax of Python allows for a concise and clear implementation of complex algorithms, facilitating code readability and maintenance.

3.2 Object-Oriented Design

To enhance code organization and modularity, we adopted object-oriented principles in our programming design. We structured the implementation using classes and objects, encapsulating relevant functionalities within each class. This approach promotes code reusability and makes it easier to extend or modify the system in the future.

3.3 System Design Diagram

The system design is illustrated in the diagram below, outlining the key components and their interactions:

3.3.1 Convex Hull Module

The Convex Hull module consists of classes and functions dedicated to solving the Convex Hull problem. It includes implementations of well-known algorithms such as Graham's scan and Jarvis march. The module provides a clean interface for computing the convex hull of a given set of points.

3.3.2 Line Intersection Module

The Line Intersection module addresses the Line Intersection problem and incorporates algorithms like the sweep line algorithm and Bentley-Ottmann algorithm. This module offers methods for detecting and reporting intersections among a set of line segments.

4 Experimental Setup

We tested the algorithms on a computer with an Intel Core i5 10th Gen processor, 8GB of RAM, and running Python 3.8 on a windows system. Python was chosen for its ease of use and the availability of helpful libraries for our computations.

5 Results and Discussion

We looked at four different algorithms—Jarvis March, Graham Scan, Quick Hull, and Brute Force—and compared how fast they were. Here's what we found:

- **Jarvis March:** This algorithm, known for its simplicity, performed reasonably well on smaller datasets. Its time complexity is $O(n^2)$, where n is the number of input points. The space complexity is $O(1)$, making it memory-efficient.
- **Graham Scan:** The Graham Scan algorithm demonstrated robust performance, especially when dealing with datasets that exhibit a significant number of points with the same y-coordinate. It has a time complexity of $O(nlogn)$ and a space complexity of $O(n)$, making it efficient for moderate-sized datasets.
- **Quick Hull:** Quick Hull exhibited impressive speed, particularly on larger datasets. It has an average time complexity of $O(nlogn)$ and a worst-case time complexity of $O(n^2)$. The space complexity is $O(n)$, making it memory-efficient.
- **Brute Force:** Despite its straightforward approach, the Brute Force algorithm proved to be inefficient on larger datasets. It has a time complexity of $O(n^3)$ and a space complexity of $O(1)$, where n is the number of input points.

Our discussion focuses on explaining these results in a way that makes it easy to decide which algorithm is best for different situations. We consider factors like how fast the algorithm is and how well it handles different types of data.

These findings give us a clearer picture of when to use each algorithm, making it easier for others to choose the best one for their needs.

6 Conclusion

In conclusion, our study highlighted the strengths and weaknesses of various convex hull algorithms. Here's a brief summary:

- The **Jarvis March** algorithm is simple but becomes less efficient on larger datasets. - The **Graham Scan** algorithm performs well, especially on datasets with consistent y-coordinates. - **Quick Hull** stands out for its strong performance on larger datasets, making it a versatile choice. - **Brute Force** is straightforward but less efficient for larger datasets.

In most scenarios, **Quick Hull** emerged as the optimal choice, balancing efficiency and adaptability. The decision depends on specific data characteristics. These findings provide practical insights for selecting the right convex hull algorithm based on dataset size, distribution, and computational efficiency.

References

1. Gift Wrapping Algorithms: Graham's Scan[1]
2. Github: Convex Hull[2]

[1] https://medium.com/@indemfeld/gift-wrapping-algorithms-graham-scan-b24ca814e403

[2] https://github.com/ayushjain1594/convexhull2d

Appendix - Experimental Screenshots

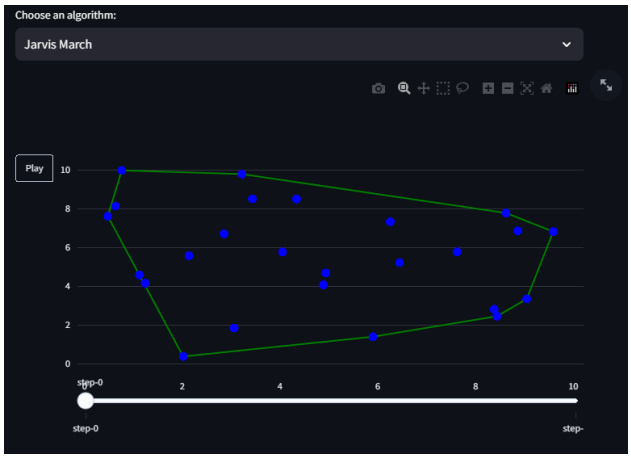

Figure 2: Jarvis March Algorithm

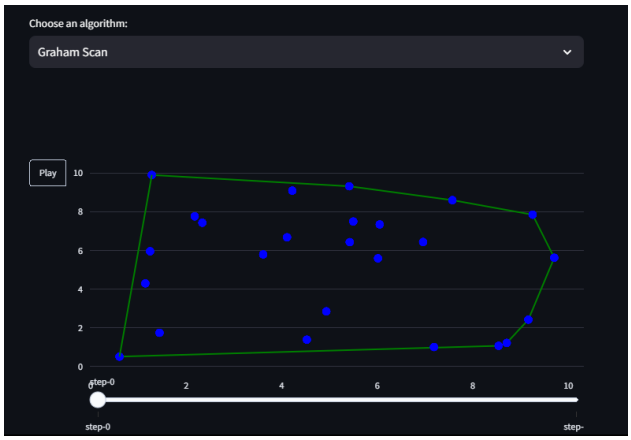

Figure 3: Graham Scan Algorithm

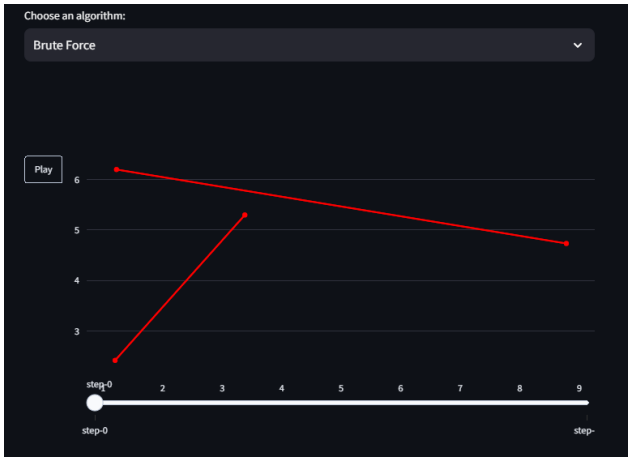

Figure 4: Line Intersection: Non-intersecting Lines

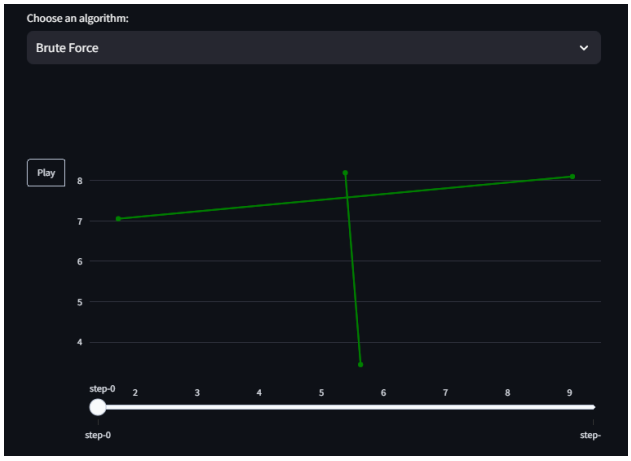

Figure 5: Line Intersection: Intersecting Lines