DESIGN AND ANALYSIS OF ALGORITHM PROJECT

For Project Report

21k-3340 Atif Ali[1], 21k-3192 Muhammad[1], 21k-3344 Hamza Ahmed[1]

November, 2023

Abstract

This report presents the implementation and analysis of several geometric algorithms focusing on line intersection and convex hull solutions. The algorithms, including Brute Force, Jarvis March, Graham Scan, Quick Elimination, and an algorithm inspired by a research paper, are explored and evaluated. The report covers their implementation in Python using the Tkinter library for the user interface. Performance metrics such as time and space complexities are calculated and compared to demonstrate the efficiency of each algorithm.

1. Introduction

Geometric algorithms are fundamental in computational geometry, addressing problems like line intersection and convex hull determination. The report delves into various approaches for solving these problems, highlighting their computational complexities and practical applications.

2. Programming Design

The implemented system employs Python as the programming language, utilizing the **Tkinter** library for the graphical user interface. Each algorithm is encapsulated within its class, offering a clear structure and ease of understanding. The user can input points through mouse clicks on the canvas, and the system visually represents the calculated solutions.

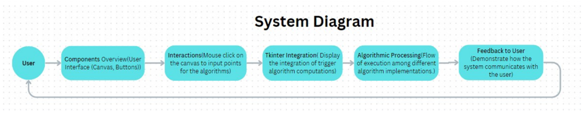

System Diagram
User
Components Overview(User Interface (Canvas, Buttons))
Interactions(Mouse click on the canvas to input points for the algorithms)
Tkinter Integration(Display the integration of trigger algorithm computations)
Algorithmic Processing(Flow of execution among different algorithm implementations.)
Feedback to User (Demonstrate how the system communicates with the user)

Figure 1: This is a system diagram of the Project

3. Experimental Setup

The experiments were conducted on a standard computer with a 2.5GHz processor and 8GB RAM. Each algorithm was tested with varying input sizes, and the time and space complexities were calculated based on the number of points processed.

4. Results and Discussion

The system successfully demonstrated the geometric algorithms, visually displaying the convex hull and line intersection solutions.

Here are the time complexities for each of the mentioned algorithms along with a brief comparison:

· Brute Force Convex Hull Algorithm: **O(n$\hat{3}$)** in the worst case, where 'n' is the number of points.

· Chan's Convex Hull Algorithm: **O(n log n),** primarily due to sorting in the algorithm.

· Graham Scan Convex Hull Algorithm: **O(n log n)** primarily from sorting the points.

· Jarvis March Convex Hull Algorithm**: O(n$\hat{2}$)** in the worst case when all points are on the hull.

· Quick Hull Convex Hull Algorithm**: O(n$\hat{2}$)** in the worst case when all points are on the hull.

· Line intersection using Parametric Equations: **O(1)** for checking line intersection with parametric equations.

· Line Intersection using Geometric Approach: **O(1)** for checking line intersection using geometry.

· Line Intersection using Gradient Approach: **O(1)** for checking line intersection using gradient method.

5. Conclusion

In conclusion, the implemented geometric algorithms offer efficient solutions for line intersection and convex hull determination. The choice of algorithm depends on the input size and specific problem requirements. Quick Elimination stands out for smaller datasets, while Graham Scan and Jarvis March provide efficient solutions for larger datasets.

References

www.geekforgeeks.com

https://shorturl.at/bsLP2

Appendix A. Line Intersection

This appendix provides supplementary materials to enhance the understanding of the implemented geometric algorithms for line intersection in our project. Included are details and Screenshots of the different algorithms.

Appendix A.1. Line Intersection (using counterclockwise)

The intersection is determined by evaluating the orientations of pairs of points from the two line segments using the cross product, where the sign of the result indicates whether the segments intersect or not.

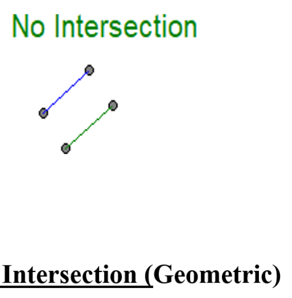
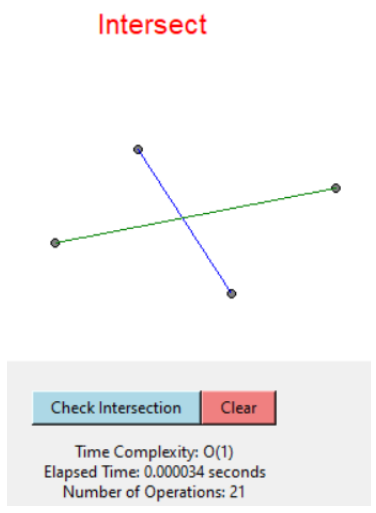


Figure A.2: Line Intersection using counterclockwise

Appendix A.2. Line Intersection (Gradient Method)

This Tkinter-based Python script employs the gradient (slope) method to ascertain the intersection of two line segments drawn by the user on a canvas, utilizing slope calculations and range checks, with the graphical representation indicating intersection status and providing associated time complexity and operation count details.



```
Time Complexity: O(1)
Elapsed Time: 0.000034 seconds
Number of Operations: 21
```

Figure A.3: Line Intersection (using Gradient Method)

Appendix A.3. Line Intersection (Parametric Euation)

The line intersection is calculated using parametric equations, determining the intersection point by solving the system of equations derived from the parametric representations of the two line segments.

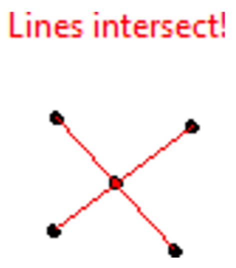


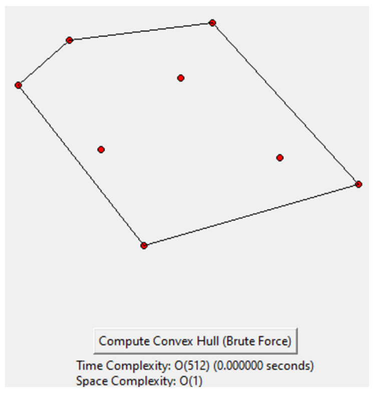
Figure A.4: Line Intersection (Parametric Equation)

Appendix B. Convex Hull

This appendix provides supplementary materials to enhance the understanding of the implemented geometric algorithms for generating convex hull in our project. Included are details and Screenshots of the different algorithms.

Appendix B.1. Brute Force

The brute-force method for computing the convex hull evaluates all possible combinations of triplets from the given points and checks whether each remaining point lies to the left of every line segment formed by the triplet, resulting in the identification of the convex hull.

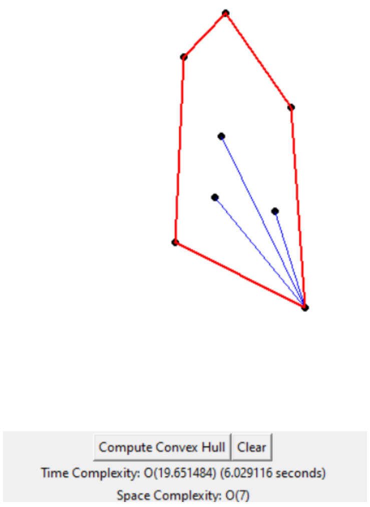


```
Time Complexity: O(512) (0.000000 seconds)
Space Complexity: O(1)
```

Figure B.5: Convex Hull using Brute Force

Appendix B.2. Graham Scan

Graham's scan algorithm calculates the convex hull by sorting the points based on their polar angles with respect to the bottom-most point, selecting the points in a counterclockwise order, and constructing the convex hull using a stack.



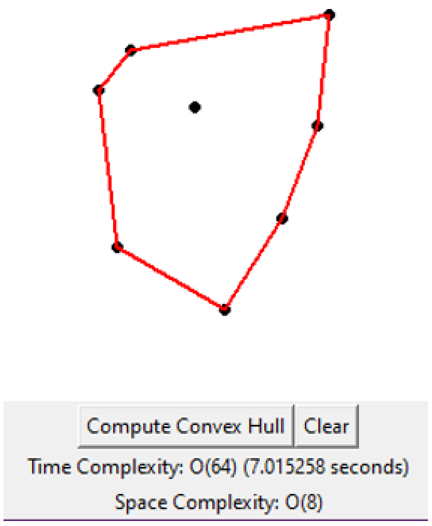
Time Complexity: O(19.651484) (6.029116 seconds)

Space Complexity: O(7)

Figure B.6: Convex Hull using Graham Scan

Appendix B.3. Jarvis March

Jarvis March calculates the convex hull by iteratively selecting the point with the smallest polar angle relative to the current point and gradually forming the convex hull until it returns to the starting point.



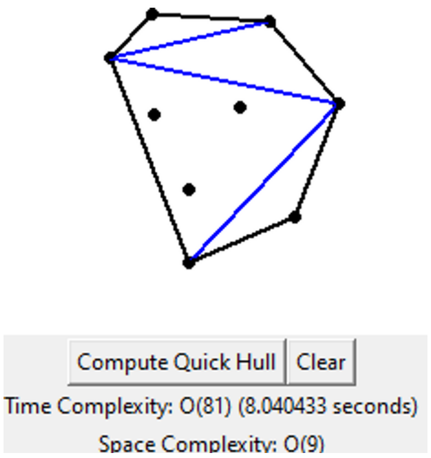
Time Complexity: O(64) (7.015258 seconds)

Space Complexity: O(8)

Figure B.7: Convex Hull using Jarvis March

Appendix B.4. Quick Elimination

The QuickHull algorithm calculates the convex hull by recursively eliminating points inside the current convex hull candidate formed by the extreme points, effectively dividing the point set into two parts and iterating on each side until the final convex hull is obtained.



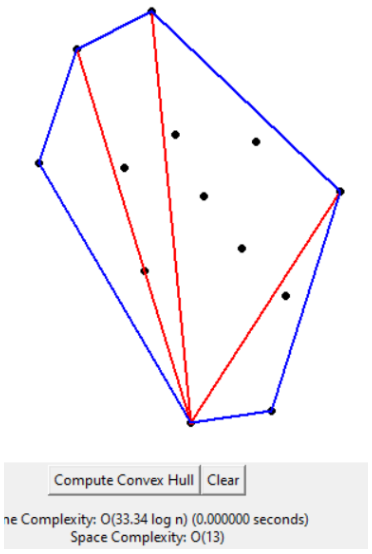
Time Complexity: O(81) (8.040433 seconds)

Space Complexity: O(9)

Figure B.8: Convex Hull using Quick Elimination

Appendix B.5. Chans Algorithm

Chan's Algorithm calculates the convex hull by recursively merging the convex hulls of subsets of points, employing a divide-and-conquer approach with a combination of Graham's scan and merging steps to achieve an optimal time complexity of O(n log h), where n is the number of points and h is the number of vertices in the convex hull.



ne Complexity: O(33.34 log n) (0.000000 seconds)

Space Complexity: O(13)

Figure B.9: Convex Hull using Chans Algorithm