Algorithm Project

Huzaifa Quaid Johar K21-3272
Saleh Shamoon K21-3333
Ahmed Saif K21-3232

Abstract

Purpose/Objective: To perform geometric algorithms using dynamic programming with visualization.
Methods/Approach: Methods provided in the requirements provided including external references (mentioned below) were used.
Results/Findings: The time and space complexities do differ a lot despite of the quick output generated by the system.
Conclusions/Implications: The conclusions drawn from the findings is that understanding the strengths, limitations, and computational complexities of these algorithms allows practitioners to select the most suitable method for specific problems, optimizing both performance and accuracy in geometric computations.

1 Introduction

In this project, we were required to implement the following problems using geometric algorithms with various Big Oh complexities . We were tasked to design programs which would either the user maps the points or inputs the number of points they would like to perform an operation on then the program would generate said number of points to apply the operations. The operations are as follows:

• For line intersection
o CCW (taught in class)
o Algebraic Method [1]
o Bounding Box[2]

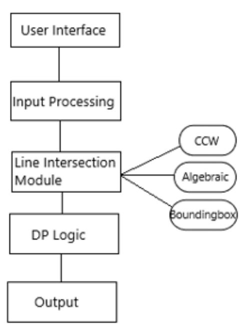
• For Convex Hull Solution
o Brute Force
o Jarvis March
o Graham scan
o Quick Elimination
o Divide And Conquer[3]

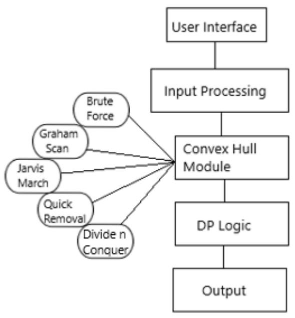
2 Programming Design

2.1 Languages Used

The line intersection algorithms were developed and visualized using Python. The Convex hull solutions were designed using HTML CSS and Java Script

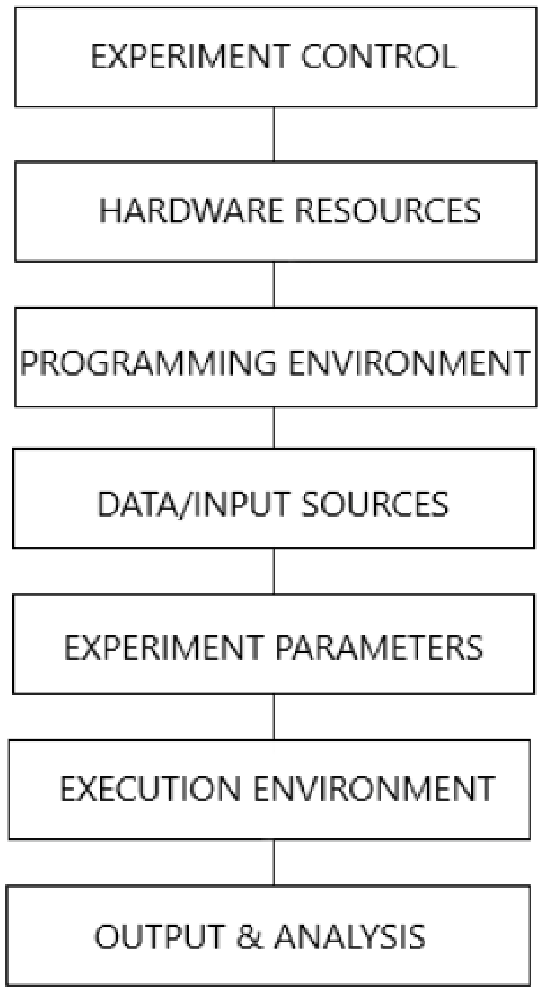
2.2 System Diagram



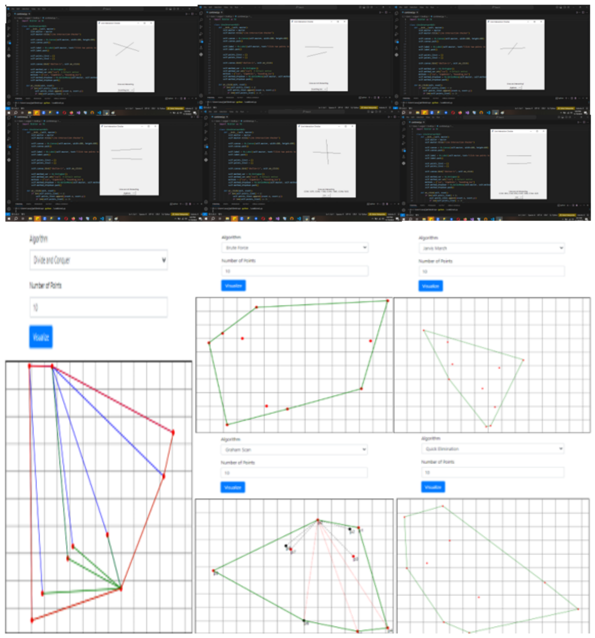


The user is presented with an interface where (with option provided) they either map the lines or give number of points as input and use the desired module and then they are presented with the output required.

2.3 Experimental Setup



3 Results & Discussion


Algorithm
Divide and Conquer
Number of Points
10

Algorithm
Brute Force
Number of Points
10

Algorithm
Jarvis March
Number of Points
10

Algorithm
Graham Scan
Number of Points
10

Algorithm
Quick Elimination
Number of Points
10

Images above demonstrating how the programs deal with the problems using different methods.

4 Conclusion

Dynamic programming algorithms offer efficient solutions for convex hull and line intersection problems. Convex hull algorithms like Graham Scan and Jarvis March efficiently determine the convex hull of points, while techniques such as bounding boxes, algebraic methods, and counterclockwise checks efficiently find line intersections. Each algorithm has trade-offs in time and space complexity, and their suitability depends on problem-specific requirements, data-set size, and constraints. Scalability and accuracy are essential considerations when selecting the most appropriate algorithm for a particular scenario.

References

[1] NA, "Geometry concepts part 2: Line intersection and its applications," *TopCoder*, 2018.

[2] NA, "Line box intersection," *3D Programming Weekly*, 2006.

[3] G. M. S. G. Editors, "Convex hulls of finite sets of points in two and three dimensions," *Programming Techniques*, 1977.

5 Appendix

5.1 Time and Space Complexities

5.1.1 Line Intersection

1. **Counterclockwise Method**: Time: Usually constant time O(1) per comparison for checking point order. Space: Minimal, typically O(1).

2. **Bounding Box Method:** Time: O(n) to create bounding boxes initially, reducing subsequent intersection checks. Space: O(n) to store bounding boxes for lines.

3. **Algebraic Method:** Time: Depends on equation solving complexity, ranging from O(1) to O(n$\hat{3}$). Space: Variable, typically O(n) to O(n$\hat{2}$) for equation-solving methods.

   Remember, these complexities are approximations and can vary based on implementation and specific cases. Each method has its strengths depending on problem requirements and data characteristics.

5.1.2 Convex Hull

1. **Brute Force:** Time: O(n$\hat{3}$) in the worst case, considering all possible combinations of points. Space: O(1) as it often operates in place without significant additional space. Jarvis March (Gift Wrapping): Time: O(nh) in the worst case, where h is the number of points on the convex hull. In the average case, O(n$\hat{2}$). Space: O(1) as it typically operates in place.

2. **Graham Scan:** Time: O(n log n) for sorting points + O(n) for the scan, totaling O(n log n) overall. Space: O(n) for storing the convex hull points.

3. **Quick Elimination:** Time: O(n log n) in the average case and O(n$\hat{2}$) in the worst case. Space: O(n) for storing intermediate data and recursive calls.

4. **Preparata Divide and Conquer:** Time: Achieves O(n log h) complexity, optimizing the standard Divide and Conquer method. Space: O(n) for recursive calls and intermediate data structures.

   Each algorithm has different time and space complexities, making them more suitable for specific scenarios based on the size and nature of the input data.