Implementation of Geometric Algorithms

K20-0467 Muhammad Minhal K20-1903 Syed Adil K20-2626 Saud Ali

Abstract

The report outlines a project involving the implementation of geometric algorithms with diverse complexities. The focus is on developing a user-friendly interface for drawing objects on the screen or utilizing file input. Time and space complexities of the algorithms are explored, and the report provides a summary of the findings.

1 Introduction

The project involves the implementation of various geometric algorithms with a wide range of complexities. Our focus is on developing a user-friendly interface that allows users to draw objects on the screen or use file input. Throughout the implementation, we explore the time and space complexities of these algorithms. This report provides a comprehensive summary of our findings.

2 Programming Design and Language

We opted Python as our primary programming language because of its widespread utilities and capabilities to code complex problems easily. It helped us in providing graphical analysis throughout the scope of the project so that the algorithms can be understood visually as well. The well-equipped Python libraries made it easier for us to grasp hold of the wide array of tasks and helped us in handling them with ease. Problems encountered throughout the scope of the project were resolved with ease as well.

3 Experimental Setup

The project scope, as mentioned, spans to three algorithms of Line segment's intersection. Furthermore, five more algorithms have been implemented involving convex hull generations from given sets of points or user-defined points.

3.1 Line Intersection via Parametric Equations

The first Line Intersection Algorithm uses parametric equations with the help of the provided coordinates to determine whether the line segments intersect or not.

- Line 1:

$$x = x_1 + t \cdot (x_2 - x_1),$$
$$y = y_1 + t \cdot (y_2 - y_1).$$

- Line 2:

$$x = x_3 + u \cdot (x_4 - x_3),$$
$$y = y_3 + u \cdot (y_4 - y_3).$$

These equations are made by two coordinates of each line segment and represented by t and u. Solving the equations requires invoking the np.linalg.solve functionality of the NumPy library. t and u values are computed, and if they fall in the range of [0, 1], the line segments intersect; otherwise, they don't.

3.2 Line Intersection via CCW Method

Before we get into understanding the algorithm, we must first explain CCW to you. CCW or Counter

Clockwise method is invoked to determine the orientation of given three points. It determines whether the line segments intersect or not by employing the cross product. It takes the slopes of the three points into consideration to see whether they form a counterclockwise turn or clockwise. Suppose three points A, B, C for the following formula:

- Formula:

$$CCW(A, B, C) = (B_x - A_x) \times (C_y - A_y) - (B_y - A_y) \times (C_x - A_x)$$

It calculates the cross product of vectors formed by these points. If CCW(A, B, C) > 0, the points create a counterclockwise turn; CCW(A, B, C) < 0 signifies a clockwise turn, and CCW(A, B, C) = 0 indicates collinearity. This method assists in determining the relative orientation of points, crucial for assessing if line segments intersect or not. This is the gist of how the algorithm works, coupled with a GUI that lets you insert coordinates and tells you whether the lines intersect or not.

3.3 Line Intersection via Slope Method

This algorithm embraces the use of slope/gradient to calculate whether the line segments intersect or not. It takes input of slope of Line 1 and its y-intercept, and the same for Line 2 as well, then computes their intersection. It's straightforward and can easily compute intersection via comparisons of slopes.

3.4 Brute Force

The Brute Force algorithm, through the 'brute-force()' function, forms a convex hull by exhaustively checking each point against the current hull. It iden- tifies extreme points based on x-coordinates, initial- izes structures for hull points, and iterates through the remaining points. Points above the current hull are added to the hull set and list. This exhaustive process determines each point's role in constructing the convex hull.

3.5 Jarvis March

Jarvis March A.K.A Gift wrapping algorithm is another algorithm to grasp convex hull from a given set of points. It supposes input of n points plot- ted on the sample space of 'tkinter' library ran- domly. The jarvismarch(points) function embod- ies the Jarvis March (Gift Wrapping) algorithm, designed to compute the convex hull of a set of points efficiently. This function begins by identi-

fying the starting point as the one with the minimum x-coordinate among the given points. It iterates through the points to construct the convex hull incrementally, adding points based on their orientations in relation to the current hull boundary. The algorithm selects the next point by considering its orientation with respect to the line formed by the starting point and the potential next point, ensuring a right turn or the farthest distance from the start point. This iterative process terminates when the next point coincides with the starting point, signaling the completion of the convex hull. The function concludes by return- ing the list of points representing the convex hull in count

3.6 Graham Scan

Graham Scan algorithm is specifically designed to compute the convex hull of a set of points. The 'grahamscan(points)' function initiates the algorithm by first identifying a starting point with the lowest y-coordinate and then sorting the remaining points based on their polar angles in a counterclockwise direction from the reference point. Through a series of operations involving point comparisons and stack manipulations, the algorithm constructs the convex hull by iteratively determining which points contribute to the hull's boundary, eliminating others, and updat- ing the stack accordingly. The resulting S stack holds the points forming the convex hull. This implemen- tation efficiently generates the convex hull by itera- tively scanning through the points, utilizing their an- gular relationships to the reference point, and build- ing the hull incrementally, resulting in a convex poly- gon encapsulating the given set of points.

3.7 Quick Elimination

This is yet another convex hull generation algo- rithm. The quick elimination(points) function imple- ments the Quick Elimination algorithm to compute the convex hull of a set of points efficiently. Initially sorting the points based on their x-coordinates, the algorithm constructs the upper and lower hulls sep- arately. It iterates through the sorted points, elimi- nating those creating clockwise turns, ensuring that each hull represents the convex boundary. Finally,it combines the upper and lower hulls, excluding the last points from both, to produce the resulting con- vex hull. This method efficiently generates the con- vex hull by iteratively eliminating points that form interior angles, ultimately providing a concise repre- sentation of the convex boundary encompassing the given points. The result is displayed on the tkinter sample space showcasing the convex hull.

3.8 Quick Hull

This is another convex hull generation algorithm. The 'quickhull(points)' function embodies the Quick hull algorithm, a highly efficient method for comput- ing the convex hull of a point set. It initiates by determining the leftmost and rightmost points, cre- ating an initial line segment for hull construction. Utilizing helper functions such as 'findsidepoints' to segregate points on one side of a line and quickhull- recursive for recursive hull expansion, the algorithm incrementally constructs the convex hull. The recur- sive method identifies the farthest point from a line formed by two existing hull points, incorporating it into the hull and subdividing the point set into sub- sets for further analysis. Functions like orientation and distance are crucial for evaluating the relative positions of points and determining farthest points from the current hull boundary. Collectively, these functions drive the divide-and-conquer strategy of the Quickhull algorithm, efficiently generating a convex hull by iteratively expanding it through recursive sub- division and point inclusion based on their distances from existing hull segments.

4 Results and Discussion

The following convex hull generation algorithms and line intersection algorithms yielded interesting results that were varying from each other. Below is the provided Results of each Algorithm alongside their name.

4.1 Line Intersection via Parametric Equations

Result:

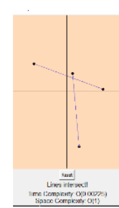

Figure 1: Parametric Equations

The overall time complexity is : 0.522 seconds

4.2 Line Intersection by CCW Method

Result:

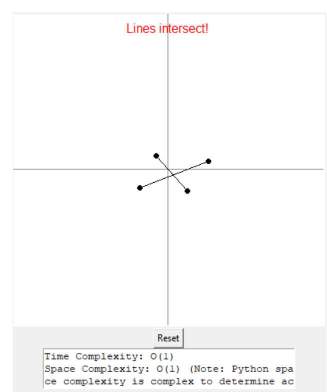

Figure 2: CCW

The overall time complexity is : 0.005250 seconds

4.3 Line Intersection via Slope Method

Result:

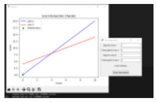

Figure 3: Slope

The overall time complexity is : 0.002014 seconds.

4.4 Brute force

Result:

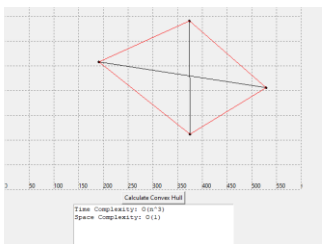

Figure 4: Brute Force

The overall time complexity is : 0.109281 seconds

4.5 Jarvis March

Result:

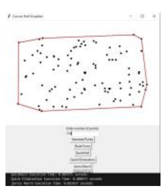

Figure 5: Jarvis March

The overall time complexity is : 0.021937 seconds.

4.6 Graham Scan

Result:

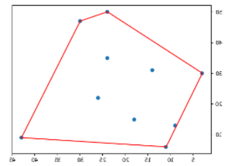

Figure 6: Graham Scan

The overall time complexity is : 0.020941 seconds.

4.7 Quick Elimination

Result:

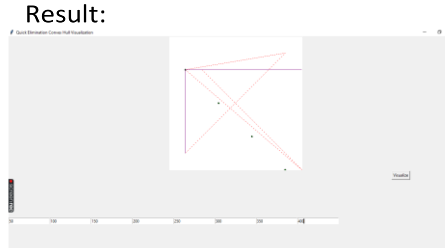

Figure 7: Quick Elimination

The overall time complexity is : 0.008973 seconds.

4.8 Quick Hull

Result:

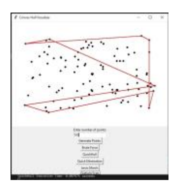

Figure 8: Quick Hull

The overall time complexity is : 0.007975 seconds.

4.9 Discussion

Conclusively out of all the convex hull generation algorithms,Quick hull stood out in terms of having the least time complexity due to its least time elapsed in performance.On the other hand,the Brute Force Convex hull algorithm performs as the worst when it comes to convex hull generation due to its enumeration of all edges factor.For the line-intersection algorithms the Slope Method stands out significantly due to its straight forward approach in analyzing intersection.The Parametric Equations Line intersection falls below due to its complex calculations.

5 Conclusion

For n set of points, in convex hull generation algorithms, the Quick hull,Graham Scan and Quick Elimination performed significantly better as for Jarvis March and Brute Force,they were not as cost effective and consumed relatively larger amount of time.The worst performance was showcased by Bruteforce. As for Line intersection algorithms the Slope Methodwas the best out of the all the Line Intersection algorithms.

6 References

https://www.geeksforgeeks.org/
https://en.wikipedia.org/wiki/Convex$_h$*ull*$_a$*lgorithmshttps* :
//www.youtube.com/watch?*v* = *B*2*AJoQSZf* 4*M*