Line Intersection Algorithms & A Comparison between Convex Hull Algorithms

1st Bilal Sohail
National University of Computer and Emerging Sciences
Karachi, Pakistan
k214554@nu.edu.pk

2nd Owais Ali
National University of Computer and Emerging Sciences
Karachi, Pakistan
k213298@nu.edu.pk

3rd Hamza Iqbal
National University of Computer and Emerging Sciences
Karachi, Pakistan
k214513@nu.edu.pk

***Abstract*—This paper investigates the implementation and performance assessment of convex hull algorithms along with methods to find intersection between line segments in the domain of computational geometry. The convex hull algorithms examined encompass classical methodologies, including the Brute Force technique, the Graham Scan, Jarvis March, Point Elimination, and Quick Hull. Executed in the Python programming language, each algorithm undergoes meticulous implementation, with a focus on the calculation of execution times. The principal objective is to quantify the computational efficacy of these algorithms in delineating convex hulls from arbitrary point sets. Through a meticulous comparative analysis of execution times, this study aims to offer valuable insights into the relative computational efficiencies of each algorithm, thereby contributing to the body of knowledge in computational geometry. The second part of the paper covers the explanation of three line intersection algorithms; one of the algorithms to identify intersection between line segments.**

I. Introduction

Within the expansive field of computational geometry, this research project delves into the intricate implementation and empirical evaluation of algorithms critical to geometric problem-solving. The spotlight is on two distinct categories of algorithms—convex hull determination and line segment intersection—each addressing fundamental challenges in their respective domains.

The computation of convex hulls stands as a foundational challenge in computational geometry, boasting diverse applications across domains such as computer graphics and geographic information systems. This research project undertakes the rigorous implementation and empirical evaluation of five distinct convex hull algorithms, executed in the Python programming language.

The algorithms under scrutiny include the classical Brute Force methodology, revered for its simplicity yet challenged by scalability; the Graham Scan, employing angular sorting for expedited convex hull determination; Jarvis March, a gift-wrapping algorithm notable for its intuitive approach; Point Elimination [1], utilizing a divide-and-conquer strategy; and Quick Hull[2], a hybrid algorithm amalgamating divide-and-conquer and incremental techniques.

Motivated by the imperative need for a nuanced understanding of the computational efficiencies of these algorithms, our research endeavor involves the meticulous implementation of each algorithm and the subsequent measurement of their execution times. This empirical approach aims to transcend theoretical considerations, offering a rigorous comparative analysis to discern the practical performance disparities among the algorithms. The outcomes of this study are poised to contribute essential insights to the computational geometry domain, guiding both researchers and practitioners in the informed selection of algorithms tailored to specific application requirements.

Additionally, we explore the Sweep Line algorithm, a pivotal methodology dedicated to identifying intersections among line segments. The Sweep Line algorithm, renowned for its efficiency in handling line segment configurations, plays a crucial role in scenarios such as computer graphics and geographical information systems.

II. Programming Design

A. Convex Hull Algorithms

For both the implementations Python programming language is used. To plot the results of the algorithms, a famous Python plotting package "Matplotlib" is used. An iterative approach has been taken for the implementation of all the algorithms except the Quick Hull algorithm, which is recursive in nature. Following is a quick overview of all the algorithms part of this paper:

1) Brute Force: This approach is not an algorithm but rather a trial-and-error approach to solving the convex hull problem. A random point is selected from the given set of points, and then by comparison with every other point in the set, it is determined whether the selected point should be part of the hull or not. Complexity of this algorithm is $O(N^3)$

2) Jarvis March: This approach helps deselect the points by starting with the smallest y-coordinate point and checks for the next point having a counter-clockwise angle with the previous point. The final result gives the set of points that should be part of the hull. Complexity of this algorithm is O(Nh)

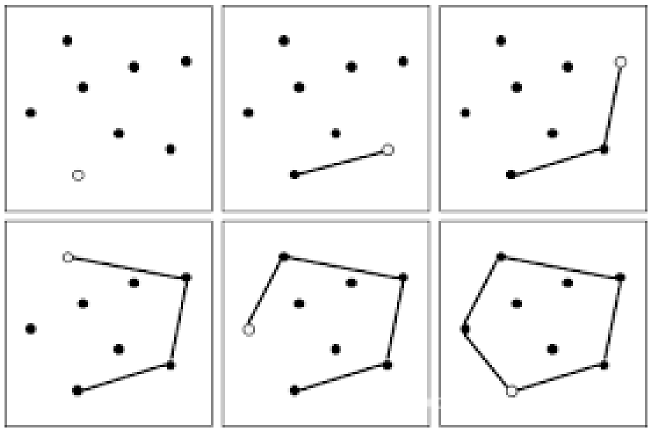
Fig. 1. Jarvis March Visualization

3) Graham Scan: Similar to Jarvis March, this approach also takes the smallest y-coordinate point as the first point and then points having counter-clockwise angles to the previous point are included in the hull. The only difference is that points not having a counter-clockwise angle are removed from the set as the iterations proceed. Complexity of this algorithm is O(Nlog(N))

4) Point Elimination (Quick Elimination): This algorithm is very different from the approaches discussed above.It takes the route of eliminating the non-hull points first and then finding the hull on the remaining points. The algorithm begins by forming a quadrilateral with extreme points of both coordinates. Points inside the quadrilateral are removed from the set of points. Afterwards, the ray scatter algorithm is used to remove points inside the quadrilateral. After the removal of points, any other convex hull algorithm can be used to find the final hull. Complexity of this algorithm is O(Nlog(N))

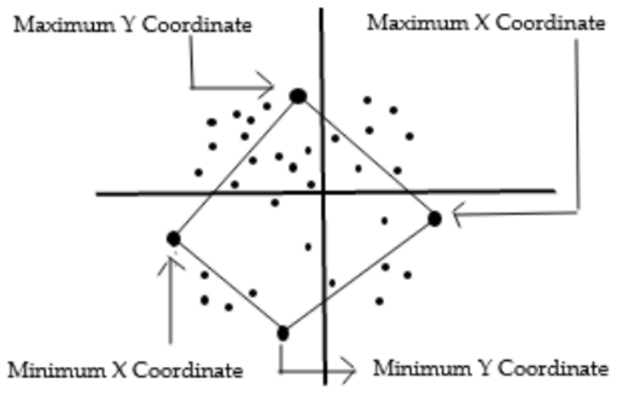

Fig. 2. Point Elimination Visualization

5) Quick Hull: This algorithms is recursive in nature and works similar to quick sort.The algorithm starts with selecting the farthest x-coordinates on left and right in the given set of points. This divides the set of points into two parts. A line segment formed between them.Then farthest point from that line segment is included in the hull-set.These three points create a temporary hull and points inside of it are removed. The above process is then repeated for all the line segements recursively to for the complete hull. Complexity of this algorithms is O(Nlog(N))

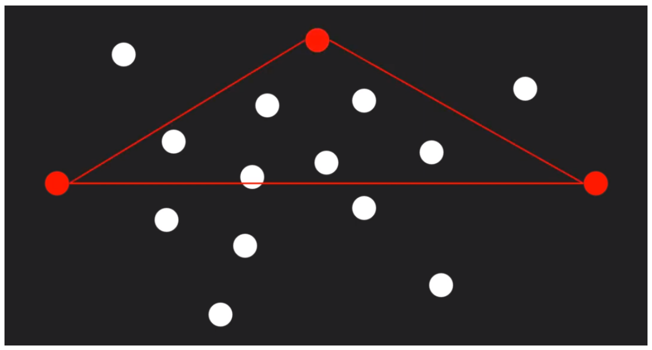
Fig. 3. Quick Hull Visualization

B. Line Intersection Algorithms

1) Counter-Clockwise Technique: The initial method employed involves determining the intersection between line segments through an assessment of their counterclockwise area from each point of one segment to the other.

2) Using Slopes: An additional method applied in this project involves determining intersections between line segments by evaluating their respective slopes. If the slopes of the two lines differ, an intersection is identified; otherwise, they are deemed non-intersecting.

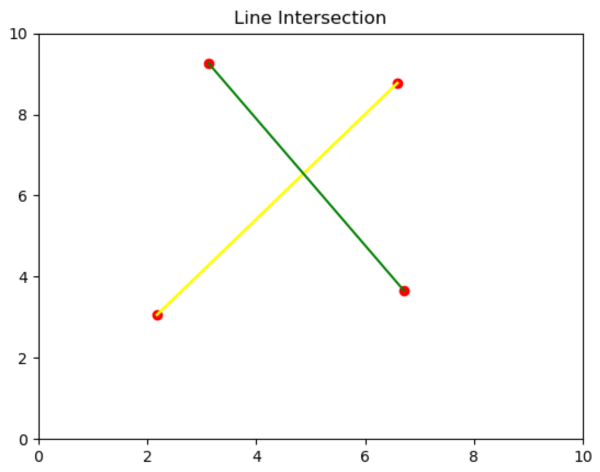

Fig. 4. Line Intersection plotting with Matplotlib

3) Using Parametric Equations: This technique involves solving parametric equations of the line segments and if the results lie between 0 and 1, intersection is identified; otherwise not.

III. Experimental Setup

A. Convex Hull Algorithms

The set of points are randomly generated points within a given range (100-500 in our experiment). The points are represented with an array of tuples, where each tuple contains the x and y coordinate of the point. Same format of array is used in all the implementations. Execution time of all the algorithms is measured with the help clock method provided in the 'time' package available for Python. For visual representation of resulting Convex Hulls is achieved through 'Matplotlib' a package for plotting available for Python.

B. Line Intersection Algorithms

Line segments are created with the help of '2DLine' object of Matplotlib.line. Line segments are created y clicking on the canvas of plotting figure window which appears when the program is executed. User draws two line segments by clicking four times. If the drawn line segments intersect, respective display is appeared on the cosole and vice-versa.

C. Visual Comparison of Algorithms

To get a visual representation of comparison between Convex Hull Algorithms, another plotting library 'Plotly' is used.

IV. Results and Discussion

TABLE I
Execution Times of Convex Hull Algorithms

Algorithm	*No. Of Points*	*Execution Time*
Brute Force	100	4.046
	200	23.621
	300	77.025
	400	196.814
	500	145.045
Jarvis March	100	1.549
	200	1.694
	300	1.756
	400	1.757
	500	1.860
Graham Scan	100	0.231
	200	0.251
	300	0.254
	400	0.240
	500	0.258
Point Elimination	100	5.797
	200	6.511
	300	9.904
	400	29.978
	500	31.003
Quick Hull	100	5.797
	200	0.000627
	300	0.008361
	400	0.0019426
	500	0.0160577

After calculation of execution times of all the convex hull algorithms and visual comparison. It is clearly visible that the best performing algorithm is Quick Hull algorithm as shown in Fig 5. This experiment also supports the idea of using this algorithm in modern computer graphics applications as it is widely used in the industry. The Line Intersection Algorithms implemented in this experiment are just for a demonstration purpose to show how same problem can be solved through various algorithms.

V. Conclusion

In conclusion, this project gave us the opportunity navigated the intricacies of computational geometry, focusing on the practical implementation and empirical assessment of algorithms crucial for addressing fundamental challenges in computational geometry. The investigation honed in line intersection algorithms, gave us insight in identifying line segment intersections with notable efficiency, particularly applicable in computer graphics and geographical information systems.

References

[1] Dr. Sajjad Waheed, Tahmina Shirin, Md. Newaz Sharif, Md. Habibur Rahaman, "A Convex Hull Algorithm using Point Elimination Technique", International Journal of Scientific Engineering Research, Volume 8, Issue 4, April-2017.

[2] C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa, "The Quickhull Algorithm for Convex Hulls", ACM Transactions on Mathematical Software, Vol. 22, No. 4, December 1996.

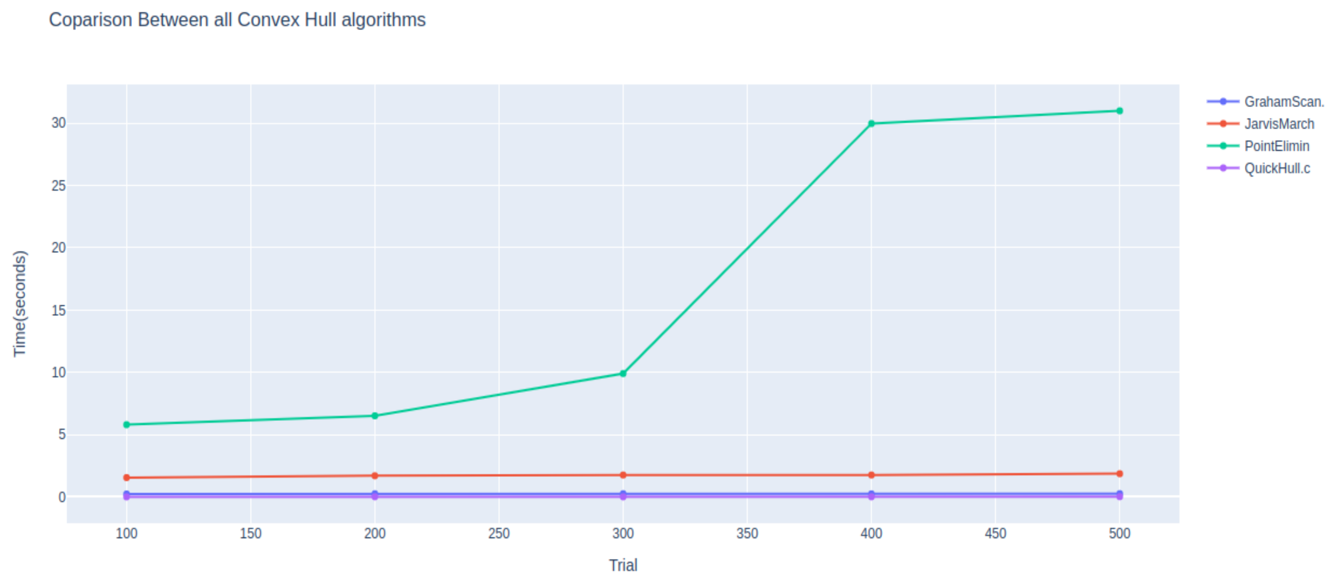

Fig. 5. Comparison of Convex Hull Algorithms as Number of points to Execution time in seconds