Geometric Algorithms: A Comprehensive Study

21K-3374 Khawaja Rabeet Tanveer
21K-4655 Muhammad Shaheer Luqman
21K-3273 Muhammad Mhustaq
21K-4613 Parshant Vijay

Abstract

This project explores the various implementations of geometric algorithms while prioritizing interaction and visualization of the steps involved. Two main problems are dealt with during this project: the algorithms behind the intersections of line segments and the creation of the convex hull. The various time and space complexities have been detailed, and a user-friendly, interactive program has been created to aid in the understanding of these seemingly daunting algorithms.

1 Introduction

The importance of Geometric Algorithms within the world of mathematics, and science in general is no secret. From graphical applications to geographical information systems, geometric algorithms are vital to computational progression. Our project aims to ease the understanding of these algorithms by allowing the user to freely input whatever set of points they please with the click of a button and see the algorithm work through it.

2 Your Programming Design

The programming language of choice was Python. Python offers an extremely wide range of utilities, making it a high-level programming language. It is arguably the easiest language present and this versatile aspect of it shines through in our project. Graphical Implementations can be quite a hassle on most programming languages, let alone coupling them with the visualization aspect that the project demands. With the vast libraries already present for Python, this is light work as such it was a rather simple and obvious choice when it came to programming languages.

3 Experimental Setup

As per the project's specifications, 3 algorithms about the intersection of line segments and 5 pertaining to the generation of the convex hull have been prepared.

3.1 CCW

An important concept to understand before any of the algorithms are dived into is the Counter Clockwise Turn **(CCW)**. CCW is a function that takes into account 3 points and returns 1 if the turn created via the connection of those points is counterclockwise, 2 if the turn is clockwise, and 0 if the points are collinear. This is achieved by the formula:

$$(P_2.y - P_1.y) \cdot (P_3.x - P_2.x) - (P_2.x - P_1.x) \cdot (P_3.y - P_2.y)$$

3.2 Intersection of Line Segments

3.2.1 CCW Method

Keeping the above information in mind, the first algorithm to detect the Intersection of Line Segments is implemented via CCW. Given 2 lines with endpoints **P1,Q1** and **P2,Q2** respectively. 4 CCW operations will be performed will the points chosen being:

$$o_1 = \text{CCW}(P_1, Q_1, P_2) \quad (1)$$
$$o_2 = \text{CCW}(P_1, Q_1, Q_2) \quad (2)$$
$$o_3 = \text{CCW}(P_2, Q_2, P_1) \quad (3)$$
$$o_4 = \text{CCW}(P_2, Q_2, Q_1) \quad (4)$$

if $(o1! = o2)$ **and** $(o3! = o4)$, the lines intersect. Furthermore, an **onsegment()** function has been implemented to deal with edge cases.

3.2.2 Slope Intersect

The second method employed to find the intersection is the slope intersect method. The input lines are checked to see whether they are parallel by creating equations from their points. These equations are then solved simultaneously to obtain the intersection points.

3.2.3 Parametric Equations

The third method implemented by research done by our group was that of Parametric Equations. Parametric equations are set up via the C of both equations being T and S respectively.

$$p_1 + t \cdot (q_1 - p_1) = p_2 + s \cdot (q_2 - p_2) \quad (5)$$

The vector representation of these points is calculated and used as a 2x2 matrix to calculate the determinant. The determinant helps realize whether the lines are parallel and if not helps in the calculation of T and S via simultaneous equations. After the calculation of T and S, they are checked to see if they fall within the range of [0,1]. If they do, the lines are intersecting.

3.3 Convex Hull

A convex hull is a set S for any points $x, y \in S$ which is the smallest possible convex set containing all the points in S. A mathematical expression of this would look like this:

$$\forall x, y \in S, \quad \forall t \in [0, 1], \quad tx + (1 - t)y \in S \quad (6)$$

A pictorial representation is also provided:

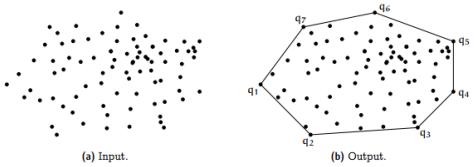

Figure 3.1: (Left)A set of points S. (Right) Its Convex Hull

3.3.1 Brute Force

The most simplistic yet resource-intensive version of the Convex Hull algorithms, brute force primarily utilizes the CCW function for its execution. Put simply, a point P on the set S of all points is chosen. A point Q and R are also chosen at random and a CCW computation is performed. If the points P and Q create a counter-clockwise turn with all other points present, the line created between points P and Q lies on the Convex Hull. This is then repeated for every other point, with Q becoming the new P.

3.3.2 Jarvis March

Jarvis March's implementation is done by choosing either the leftmost or rightmost point on the given set of points. This point is denoted as P or $current_point$. According to the choice, the points within the given set are iterated, and the ones that create a clockwise turn if the leftmost was selected or a counter-clockwise turn if the rightmost was selected are chosen and added to the convex hull. Any point that creates the desired turn is then updated to be P and the convex hull is subsequently found.

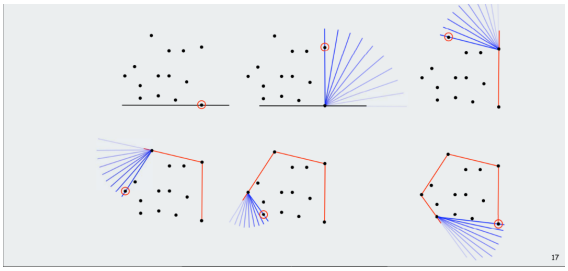

Figure 3.2: Leftmost Implementation

3.3.3 Graham Scan

Graham Scan utilizes the functionality of both Jarvis March and brute force. In Short, the points within the given set are first sorted based on their polar angles calculated from the x-axis. The sorted points are then gone through and have CCW applied to them to find out the convex hull.

3.3.4 Quick Elimination

Quick Elimination is a speedup technique used to speed up any convex hull. It works on the basis that a quadrilateral is created using random points present within the set of points given. All points present within the created quadrilateral are then ignored, and the points left along with the points used to create the quadrilateral have the algorithm run on them. The downside of this algorithm is that the method is choosing the points on the quadrilateral is not specified. This may result in a case where the points removed are minimal which will result in the algorithm slowing down.

3.3.5 Monotone Chain

Monotone Chain works on a similar concept as Jarvis March and Graham Scan. The point set is first sorted based on the x-axis and the smallest point relative to the x-axis is chosen. The algorithm then splits into 2 parts, the Upper Hull and the Lower Hull. For the upper hull, the sorted points are considered left to right and any points that create clockwise turns are ignored to create an **UPPER CONVEX HULL**. For the Lower Hull, the sorted points are considered right to left and all counter-clockwise turns are ignored creating a **LOWER CONVEX HULL**. The 2 are then concatenated to create a complete Convex Hull. A representation is shown below:

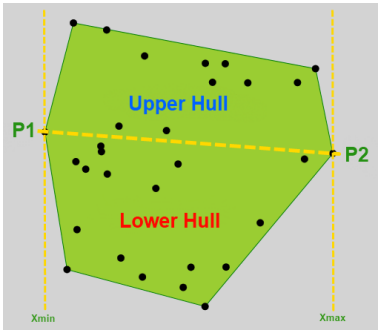

Figure 3.3: A Monotone Chain Algorithms Upper and Lower Hull

4.1 Intersection of Line Segments

Regarding the Algorithms involved in the Intersection of Line Segments, the total operation performed in each algorithm is upon a set of 4 points, $P1, Q1$ the start and endpoints of the first line and $P2, Q2$ the similar points of the second line. As such since only operations are performed upon these points and the number of points is constant, the complexity of all involved Algorithms is always **O(1)**.

The space complexities of these algorithms are also **O(1)**. This is because in each case, the input parameters for the functions performing the algorithms are 4 and the local variables used are primitive and non-scaling w.r.t input size. The constants used within the algorithm are also in numerical form, all culminating in giving the space complexity of **O(1)**.

Below are listed 3 output screenshots that show 3 different cases being run on the 3 algorithms:

4 Results and Discussion

This section consists of the outputs and time complexities of the various algorithms that have been detailed above.

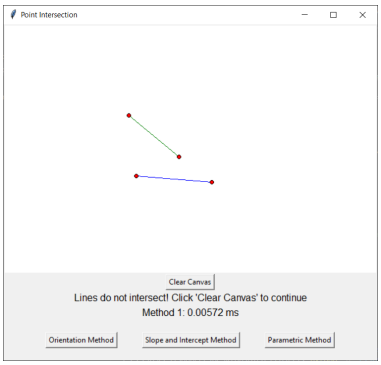

Figure 4.1: Not Intersecting

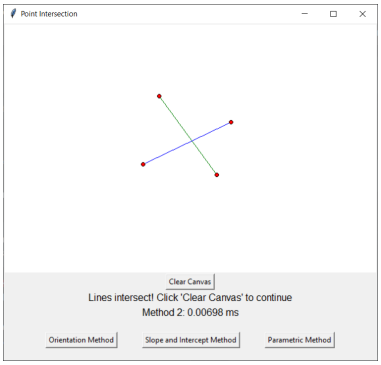

Figure 4.2: Intersecting

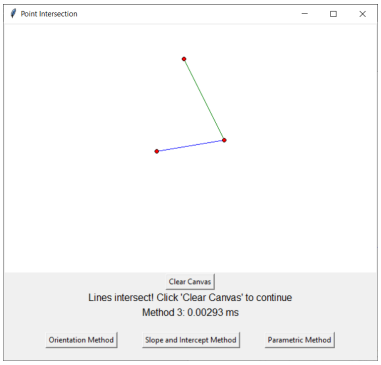

Figure 4.3: Connected on a Point

4.2 Convex Hull

In comparison to the intersection of line segments, each algorithm in a convex hull has varying time complexities and space complexities. To better represent this, all algorithms will be given their sub-sections where their relevant details will be discussed.

4.2.1 Brute Force

TIME COMPLEXITY:
i) The set of points has first been sorted to select an initial starting point. This step is optional in the brute force algorithm and yields a complexity of **O(nlogn)**.
ii)This is followed by 3 nested loops that iterate over n. These are done to choose pairs of points and decide whether the point after them will result in them lying on the convex hull or not. The complexity is $O(n^3)$.
iii) Following this, the various insertions and operations performed on the convex hull are **O(k)** where k is the number of elements that require operations.

Of the 3, $O(n^3)$ is the dominating complexity, hence the time complexity of the algorithm.
SPACE COMPLEXITY:
i) The input parameters of the algorithm are a set of points n. Hence the input space complexity is **O(n)**.
ii) The Convex Set used can at most have a size equal to its inputs. Its complexity is also **O(n)**.
iii) All other variables and operations occupy a constant complexity.

Of the 3, **O(n)** is the dominating complexity, hence the Space complexity of the algorithm.

4.2.2 Jarvis March

TIME COMPLEXITY:
i) The set of points is iterated over initially to find the leftmost point. This takes **O(n)**.
ii)This is followed by a while loop which persists as long as the original leftmost point is not reached once again. Assuming h points are added to the hull, this loop alone gives a complexity of **O(h)**. Nested within this while loop is a for loop that iterates over every point and checks whether the point in question can be added to the hull. As such the overall complexity of this Nested Loop becomes **O(nh)**.

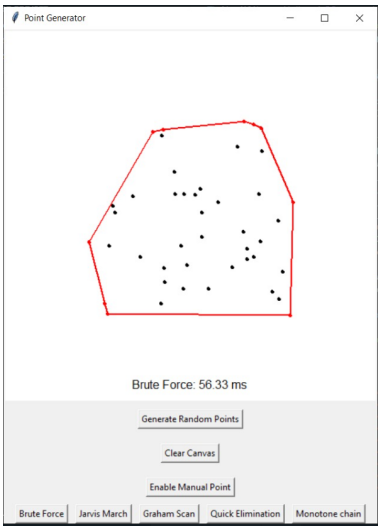

Figure 4.4: Brute Force

O(nh) is the dominating complexity, hence the time complexity of the algorithm.
SPACE COMPLEXITY:
i) The input parameters of the algorithm are a set of points n. Hence the input space complexity is **O(n)**.
ii) The Convex Set used can at most have a size equal to its inputs. Its complexity is also **O(n)**.
iii) All other variables and operations occupy a constant complexity.

Of the 3, **O(n)** is the dominating complexity, hence the Space complexity of the algorithm.

4.2.3 Graham Scan

TIME COMPLEXITY:
i) The set of points has first been sorted based on polar angles w.r.t to the lowest point on the set. This yields a complexity of **O(nlogn)**.
ii)This is followed by the Graham Scan method itself. This involves a for loop that iterates over all the sorted points for a complexity of **O(n)**. An inner while loop then works to remove the elements present in the stack if needed. The removed elements are not taken into consideration any longer, hence the iteration of the outer loop is not affected. This in turn keeps the complexity at **O(n)**.

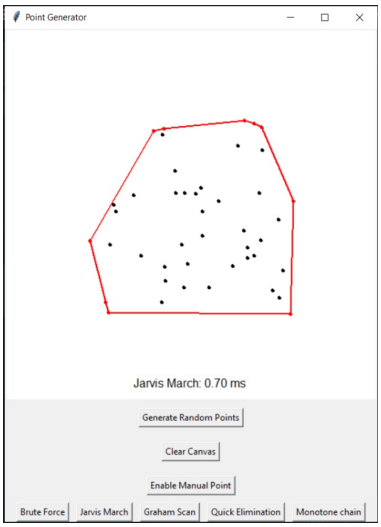

Figure 4.5: Jarvis March

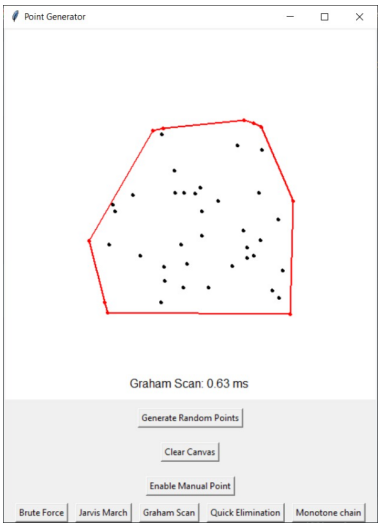

Figure 4.6: Graham Scan

Of the 2, **O(nlogn)** is the dominating complexity, hence the time complexity of the algorithm.

SPACE COMPLEXITY:

i) The input parameters of the algorithm are a set of points n. Hence the input space complexity is **O(n)**.

ii) The Candidate Points set used can at most have a size equal to its inputs. Its complexity is also **O(n)**.

iii) The stack used to store the convex hull also has a complexity of **O(n)**.

All 3 give **O(n)** as the Space complexity of the algorithm.

4.2.4 Quick Elimination

TIME COMPLEXITY:

i) The leftmost, rightmost, bottom-most, and topmost points are calculated by iterating over the points 4 times. This results in a complexity of **O(n)**

ii) A bounding box is constructed from the points in **O(1)** and then a loop is run over the remaining points to check whether they fall in the loop or not. This takes **O(n)**.

iii) This results in the number of points n being reduced to a number lesser than n, assumed to be K. Running Graham Scan on these points K will result in the complexity of Graham Scan being **O(KlogK)**.

Quick Elimination acts as a speedup for other algorithms, as such the dominating complexity often belongs to the algorithm it speeds up, in this case **O(KlogK)** is the dominating complexity, hence the time complexity of the algorithm.

SPACE COMPLEXITY:

i) The input parameters of the algorithm are a set of points n. Hence the input space complexity is **O(n)**.

ii) The Candidate Points set used can at most have a size equal to its inputs. Its complexity is also **O(n)**.

This makes **O(n)** the Space complexity of the algorithm.

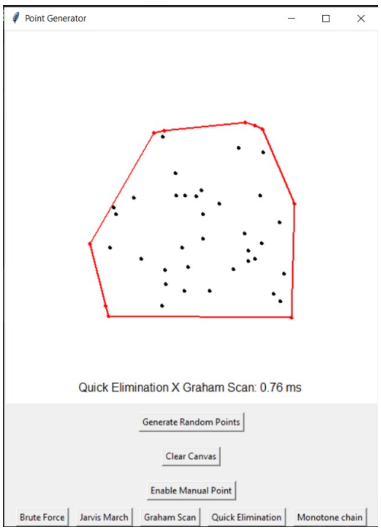

Figure 4.7: Quick Elimination

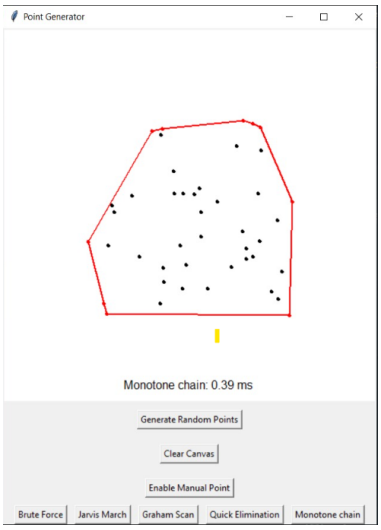

Figure 4.8: Monotone Chain

4.2.5 Monotone Chain

TIME COMPLEXITY:

i) The points in the input set are sorted based on the X-axis. This sorting takes **O(nlogn)**.

ii) The Sorted points are then traversed to create the lower and upper hulls using stack. Even in their worst complexities, this building will take a maximum complexity of **O(n)**.

iii) The concatenation of both hulls then takes O(n) since it involves combining lists of size n.

Just like Graham Scan, the sorting portion is the dominating factor for the complexity, resulting in the time complexity of **O(nlogn)**.

SPACE COMPLEXITY:

i) The input parameters of the algorithm are a set of points n. Hence the input space complexity is **O(n)**.

ii) The Candidate Points set used can at most have a size equal to its inputs. Its complexity is also **O(n)**.

This makes **O(n)** the Space complexity of the algorithm.

5 Conclusion

Bringing this report to a conclusion, we will go over the various algorithms, comparing their times taken and using graphical representations and tables to express them. In doing so, we will also determine the most time-efficient algorithms from both categories.

5.1 Intersection of Line Segments

Case 1 for the table represents the scenario where lines intersect. Case 2 for the scenario where lines do not intersect Case 3 for the scenario where lines intersect at a point.

Table 1: Intersection Algorithms (in ms)

	Case 1	**Case 2**	**Case 3**
CCW Method	0.0071	0.00687	0.00695
Slope Intersect	0.00881	0.00406	0.00793
Parametric Equations	0.00357	0.00311	0.00314

5.2 Convex Hull

Table 2: Convex Hull Algorithms (in ms)

n	Brute Force	Jarvis March	Graham Scan	Quick Elimination	Monotone Chain
10	0.8	0.1	0.14	0.22	0.11
25	10.05	0.32	0.31	0.44	0.2
50	86.49	0.62	0.58	0.82	0.42
100	536.07	1.51	1.34	1.35	0.85
500	-	9.45	7.52	6.82	3.62
1000	-	20.45	15.49	12.89	7.49

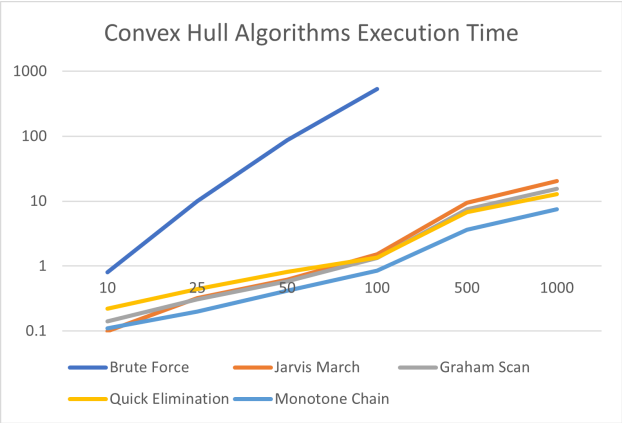

Figure 5.1: Times Taken

6 References

1. Python Algorithm Examples. (n.d.). Convex Hull - Divide and Conquer. Retrieved from https://python.algorithmexamples.com/web/divide_and_conquer/convex_hull.html

2. GeeksforGeeks. (n.d.). Convex Hull (Monotone Chain) Algorithm. Retrieved from https://www.geeksforgeeks.org/convex-hull-monotone-chain-algorithm/

3. GeeksforGeeks. (n.d.). Convex Hull using Jarvis Algorithm (or Wrapping). Retrieved from https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/

4. GeeksforGeeks. (n.d.). Convex Hull using Graham's Scan Algorithm. Retrieved from https://www.geeksforgeeks.org/convex-hull-using-graham-scan/

5. Visualgo.net. (n.d.). Convex Hull Visualization. Retrieved from https://visualgo.net/en/convexhull?slide=3

6. Quora. (n.d.). How do you find an intersection between a line and a segment if given coordinates for both lines and points on each line? Retrieved from https://www.quora.com

7. Math Stack Exchange. (n.d.). Intersection of two parametric lines. Retrieved from https://math.stackexchange.com/questions/1128106/intersection-of-two-parametric-lines