Convex Hull And Line Intersection Visualization Project Report

21K-3355 Talha Shahid 21K-3186 Ahsan Ashraf 21K-3426 Kantesh

Contents

1 Abstract

This report presents an in-depth analysis and visualization of Convex Hull algorithms, including Graham's Scan, Jarvis March, Quick Hull, and the AKT Convex Hull algorithm. The study utilizes a Python implementation with Tkinter for graphical representation. The report provides insights into the programming design, experimental setup, and detailed results and discussion of each algorithm's performance.

2 Introduction

Convex Hull algorithms are fundamental in computational geometry, with applications ranging from computer graphics to robotics. This project aims to provide a comprehensive understanding of Convex Hull algorithms through visualization and analysis.

3 Programming Design

The implementation is based on object-oriented Python, utilizing the Tkinter library for the graphical user interface. Each Convex Hull algorithm is encapsulated in a class, allowing for modular and easily extendable code.

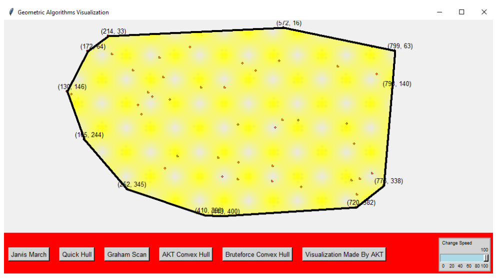


Figure 1: Tkinter-based GUI for Convex Hull Algorithms

The following code snippet illustrates the implementation of the Point2D class and the BruteForce Convex Hull algorithm:

4 Experimental Setup

The experiments involve visualizing Convex Hull algorithms on a Tkinter canvas with randomly generated points. The graphical user interface allows users to interactively control the animation speed and observe the step-by-step execution of each algorithm. The number of points, algorithms, and their execution order are configurable.

5 Results and Discussion

This section provides a detailed analysis of each Convex Hull algorithm's behavior and performance.

5.1 Graham's Scan

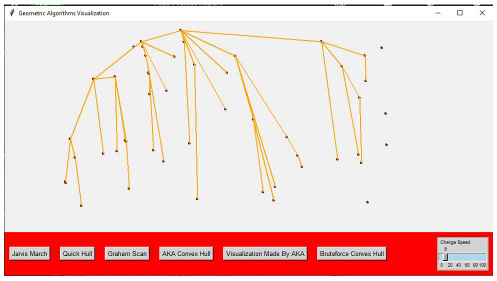


Figure 2: Tkinter-based GUI for Convex Hull Algorithms

Graham's Scan is a well-known convex hull algorithm that operates by first sorting the points based on the polar angle with respect to the lowest point. It then processes the sorted points to construct the convex hull. In your code:

- The `convex_hull_graham` method takes care of the Graham's Scan algorithm.
- It uses a custom sorting function based on polar angle to sort the points.
- The `turn` function determines the orientation of three points.
- The `_keep_left` function helps maintain the convex hull by keeping the left-turning points.

5.2 Jarvis March

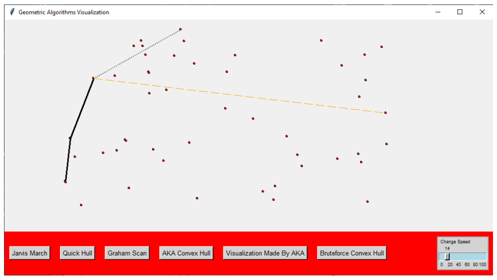


Figure 3: Tkinter-based GUI for Convex Hull Algorithms

Jarvis March, also known as the Gift Wrapping algorithm, is a simple but less efficient convex hull algorithm. It iteratively selects the point with the smallest polar angle from the current point. Your code includes:

- The `convex_hull_jarvis_march` method implements the Jarvis March algorithm.
- It iterates through points to find the next point with the smallest polar angle.
- The convex hull is constructed gradually until the starting point is reached.

5.3 Quick Hull

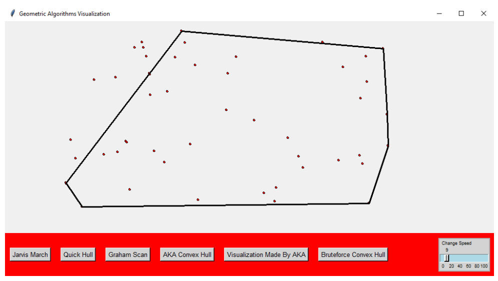


Figure 4: Tkinter-based GUI for Convex Hull Algorithms

Quick Hull is a divide-and-conquer algorithm that recursively identifies points on the convex hull. It involves selecting extreme points and partitioning the remaining points. In your code:

- The `convex_hull_quickhull` method initiates the Quick Hull algorithm.
- It starts by selecting the endpoints of the point set.
- The `find_hull` method recursively identifies points on the convex hull.

5.4 AKT Convex Hull

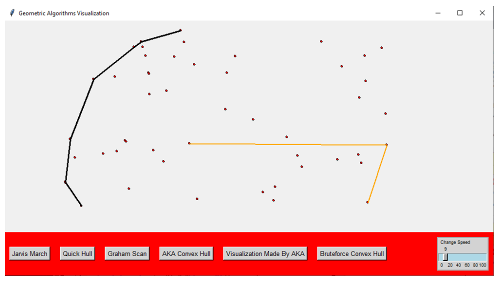


Figure 5: Tkinter-based GUI for Convex Hull Algorithms

The given code defines a class method which is named as `convex_hull_akt_algorithm` in a class named as `GeometricAlgorithms`. This method is designed to visualize the convex hull of a set of points on a canvas using the AKT algorithm. The algorithm follows these steps:

Initialization: Check if the canvas and speed parameters are provided; otherwise, return.

Delete Previous Drawings: Clear any existing geometric elements on the canvas with the tag "geometric."

Point Extraction: Extract the coordinates of points on the canvas tagged as "point."

AKT Algorithm Implementation: Define helper functions `next_to_top` and `ccw` for stack manipulation and calculating the orientation of three points. Sort the points based on their y-coordinates to separate the lower and upper halves. Initialize an empty stack. Add the first two points of the lower half to the convex hull stack.

Build the convex hull for the lower half: While the orientation of the current point with respect to the last two points in the stack is not counterclockwise, remove the last point from the stack. Add the current point to the stack. Visualize each step with a temporary line. Visualize the convex hull for the lower half. Add the first point of the upper half to the convex hull stack. Build the convex hull for the upper half using a similar process as for the lower half. Visualize the convex hull for the upper half. Connect the last point of the upper half to the first point of the lower half to complete the convex hull.

Visualization: Introduce delays to visualize each step of the algorithm on the canvas. The method utilizes the provided canvas and speed parameters to create an animated visualization of the AKT algorithm for computing the convex hull of a set of points.

5.5 Brute Force Convex Hull

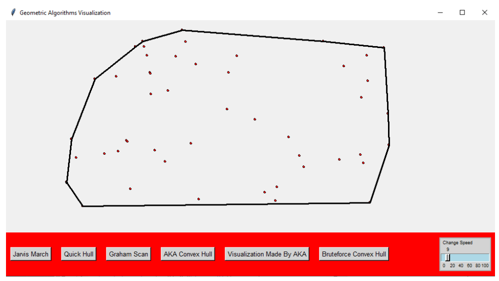


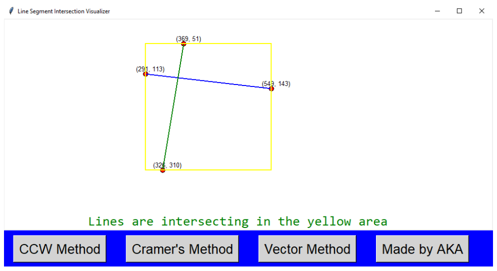
Figure 6: Tkinter-based GUI for Convex Hull Algorithms

The Brute Force Convex Hull algorithm evaluates all possible combinations of points to determine the convex hull. It checks the orientation of every triplet of points. In your code:

- The `convex_hull_bruteforce` method implements the brute-force approach.
- It checks the orientation of each triplet to determine if it is part of the convex hull.

- The `is_above` function assists in determining the orientation.

5.6 Vector Line Intersection Method



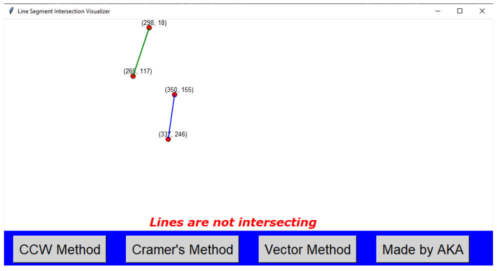
```
Lines are intersecting in the yellow area
```

Figure 7: Tkinter-based GUI for Line Intersection Algorithms

The Vector Method determines whether two line segments intersect by leveraging the orientation of four points.

- Uses orientation calculations for four points: $p1$, $q1$, $p2$, and $q2$.
- Checks for conditions where the orientations of the points differ, indicating an intersection.
- Handles special cases when points are collinear.

5.7 Cramer's Line Intersection Method



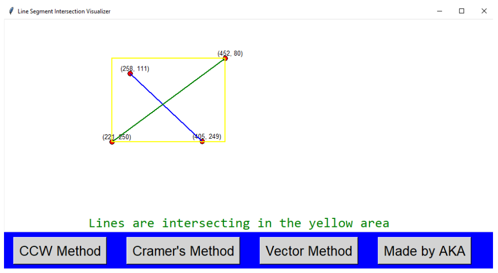
```
Lines are not intersecting
```

Figure 8: Tkinter-based GUI for Line Intersection Algorithms

Cramer's Rule is applied to determine whether two lines given by pairs of points intersect. It involves solving a system of linear equations to find a unique solution, indicating an intersection point.

- Sets up a system of linear equations using the given points.
- Uses determinants to solve for the intersection point.
- Checks if the intersection point lies within the line segments.

5.8 CCW Line Intersection Method



```
Lines are intersecting in the yellow area
```

Figure 9: Tkinter-based GUI for Line Intersection Algorithms

The Counter Clockwise (CCW) method is another approach for determining the intersection of two line segments. It uses the concept of calculating the orientation of three points to identify whether they form a counterclockwise turn.

- Utilizes orientation calculations to determine the relative position of line segments.
- Detects intersections by checking for changes in orientation between points.

6 Complexity Analysis

6.1 Time Complexity of Jarvis March

The time complexity of Jarvis March is analyzed based on the number of points in the set $|S| = n$ and the number of hull points $|H| = h$. In the average case, the algorithm iterates through the set $|S| = n$ as many times as there are hull points $|H| = h$. Therefore, the time complexity is given by $O(nh)$. In the worst case scenario, if the number of hull points $|H| = n$, then the time complexity becomes $O(n^2)$.

6.2 Time Complexity of Graham's Scan

Algorithm has a time complexity of $O(n \log n)$, making it an efficient algorithm for convex hull computation.

6.3 Time Complexity of QuickHull

The algorithm has the following time complexities: The worst-case time complexity of QuickHull is $O(n^2)$, where n is the number of input points. In an expected scenario time complexity is $O(n \log n)$.

6.4 Time Complexity of Brute-Force

A brute-force approach, attempting to intersect each edge of the first polygon with each edge of the second, works with a time complexity of $O(n^3)$, where n and m are the respective numbers of vertices of the input polygons.

6.5 Time Complexity of AKT Algorithm

The AKT Algorithm exhibits a time complexity of $O(n \log n)$, surpassing the efficiency of Graham's scan by a factor of 77 and Jarvis March by a factor of 12.

Algorithm	**Time Complexity**
Jarvis March (Average)	$O(nh)$
Jarvis March (Worst)	$O(n^2)$
Graham's Scan	$O(n \log n)$
QuickHull (Worst)	$O(n^2)$
QuickHull (Expected)	$O(n \log n)$
Brute-Force	$O(n^3)$
AKT Algorithm	$O(n \log n)$

7 Conclusion

In conclusion, the visualization and analysis of Convex Hull algorithms provide valuable insights into their performance characteristics. Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the specific requirements of the application. The Tkinter-based GUI facilitates an interactive learning experience, aiding in the comprehension of these fundamental algorithms.

8 References

8.1 Line Intersection Algorithms

8.1.1 Using Orientation and CCW

- https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect
- https://shorturl.at/pqzAK

8.1.2 Using Vector Product

- https://imois.in/posts/line-intersections-with-cross-products/
- https://shorturl.at/iwNWZ

8.1.3 Using Cramer's Method

- https://cp-algorithms.com/geometry/lines-intersection.html
- https://www.cuemath.com/geometry/intersection-of-two-lines/

8.2 Convex Hull Algorithms

8.2.1 Graham Scan

- https://shorturl.at/eptN4
- https://www.geeksforgeeks.org/convex-hull-usin
- https://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf

8.2.2 Jarvis March

- https://www.geeksforgeeks.org/convex-hull-usin

8.2.3 Brute Force

- https://www.geeksforgeeks.org/convex-hull-usin

8.2.4 AKT Algorithm

- https://shorturl.at/foy07

8.2.5 Quick Hull

- https://www.geeksforgeeks.org/quickhull-algori
- https://en.wikipedia.org/wiki/Quickhull
- https://shorturl.at/ABQV4