Geometric Algorithms

By

Syed Zeeshan Ahmed - 21K-4844
Aqib Ali - 21K-4518
Khalid Khurshid Siddiqui - 21K-4673

Supervised By

Dr. Farukh Saleem
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE
FAST National University of Computer and Emerging Sciences

Abstract

In this project, we have used GUI to help us improve our understanding of Geometric Algorithms, we have used both Python's Tkinter Library and JavaScript's P5 Library to showcase the Convex Hull through five different algorithms and Line Intersection through three different algorithms.

Introduction

It is essential to know that **Geometric Algorithms** have essential features and real-life uses so that we can develop to understand this concept better we have as per the recommendation of our Department explored the topic of **Convex Hull** through various Algorithms and also the concept of **Line Intersection** through also various algorithms.

Program Design

We have used Python's Tkinter Library, JavaScript's P5 Library, HTML and CSS(for better visualization) in our project, as JavaScript is closest to C and C++ we opted for it and it is good for plotting and explaining, we have also worked on VS Code IDE. For **Convex Hull** we have used the following Algorithms:

- Brute Force
- Graham Scan
- Jarvis March(Package Wrap)
- Quick Elimination
- Quick Hull

For **Line intersection** we have used the following algorithms:

- Bently-Ottomann Method
- Slope and Intersect Method
- Parametric method

Experimental Setup

For the **Line intersection** part the **Tkinter-based GUI** allows users to draw line segments, input coordinates, and interact with algorithms. Each algorithm is initiated through user input via mouse clicks. For **Convex Hull** GUI **Javascript's P5 Library** is used for visualization and random point generation. HTML is used for representing the Generated Animation of the Convex Hull Algorithm by P5 Library.

Conclusion

This project has greatly proved how easy it is to learn and understand such complex algorithms through visualisation. On a data set of **150 points** it requires **3sec** for **Graham Scan**, **31 sec** for **Jarvis March** and **22 sec** for **Naive or Brute Force Algorithm** on the contrary a data set of **250 points**, it requires **5sec** for **Graham Scan**, **40 sec** for **Jarvis March** and **2 minutes and 2 sec** for **Naive or Brute Force Algorithm**. Hence we can clearly see the difference b/w the time complexities of O(NH) and O(N^3).

References

The **Bently-Ottoman Algorithm** was initially developed by Jon Bentley and Thomas Ottmann in 1979. The **Quick Hull** was invented in 1996 by C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa.

Appendix

.1 Line Segment Algorithms

.1.1 Parametric Method

```
Express each line segment parametrically:
    P+t(QP)
Form a system of linear equations using the
parametric representation for two line segments.
Solve the system to find parameters
t and s using Cramer's rule.
If t and s are both in the range
[0,1], the lines intersect within their segments.
Use the parameters to find the intersection point,
if it exists.
```

.1.2 Slope and Intersect Method

```
Calculate slopes slope1 & slope2 for the line segments.
If slope 1 is different from slope 2, proceed; otherwise,
the lines are parallel, and no intersection occurs.
Calculate the potential intersection point
(intersect_x,intersect_y).
Check if the intersection point is within
the valid range of both line segments.
If the intersection point is within the range,
return 'True' otherwise, return 'False'.
```

.1.3 Bentley–Ottmann Algorithm

```
Initialize a sweep line and create events for each line segment's left and
right endpoints.
Process events in sorted order as the sweep line progresses.
Update the status structure to reflect the current set of
intersecting line segments.
When processing events, detect intersections by checking
adjacent line segments in the status structure. Generate
new events for intersection points.
Move the sweep line to the next event and continue
processing events until completion.
Output the list of intersection points found during
the algorithm's execution.
```

.2 Convex Hull Algorithms

.2.1 Brute Force

```
Iterate over distinct point pairs Create lines between points
Find the left points and check the condition Add left points to the hull
Return unique points in convex_hull
```

.2.2 Jarvis March

```
Initialize the result list with the lowest point a
Repeat until the first point is reached
    Find the next point q using the orientation function
    Set the last found point as q Append q to the result list
Return the result list as the convex hull
```

.2.3 Graham Scan

```
Find point p1 with the lowest y-coordinate and sort the remaining points
by polar angle
Initialize stack with p1 and first two sorted points
Iterate over sorted points Update stack based on polar angle
Return indices of points
```

.2.4 Quick Elimination

```
Generate random points for a rectangle
Calculate min and max coordinates then
Print (x,y)
```

.2.5 Quick Hull

```
Given a set of points in the plane.
Find the points with the minimum and maximum x-coordinates;
these will be the endpoints of the convex hull.
Divide the set of points into two subsets based on the line
formed by the two endpoints.
Recursively apply the QuickHull algorithm to the two subsets.
Merge the convex hulls obtained from the recursive calls.
The final result is the convex hull of the entire point set.
```