Algorithms Project

K213335 Syed Muhammad Ali
K213329 Syed Khizar Ali
K213216 Syed Zaki Mustafa

November 2023

1 Abstract

This project explores Convex Hull algorithms—Graham's Scan, Jarvis March, and Quick Elimination—implemented in Python. The study covers theoretical foundations, algorithmic complexities, and empirical evaluations, offering insights into their comparative efficiency for diverse applications.

2 Introduction

The project explores geometric algorithms, comparing complexities for line segment intersection and convex hull calculation, aiming to understand and assess practical performance.

3 Our Programming Design

The system employs Tkinter, itertools, random, and matplotlib for Convex Hull Algorithms, Input, and Visualization. Input manages point sets, Convex Hull Algorithms execute core functions, and the Visualization Module generates visualizations.

4 Experimental Setup

Utilizing Tkinter, the graphical user interface (GUI) enables users to draw line segments, input coordinates, and interact with algorithms. The setup employs matplotlib for visualization and random point generation. Initiation of each algorithm is triggered by user input through mouse clicks.

5 Results and Discussion

5.1 Line Intersection

The line intersection algorithm visually examines drawn segments, using the gradient method, counter-clockwise turns, and line sweep algorithm. The gradient method's time complexity for a counter-clockwise check is $O(n)$, and the Line Sweep algorithm has a time complexity of $O(nlogn)$..

5.2 Convex Hull Algorithms

1. **Brute Force:** Enables interactive point addition on a canvas with a meticulous algorithm having $O(n^3)$ time complexity, assessing all possible combinations of three points for convex hull edges.
2. **Jarvis March:** Activated by a button click, the algorithm incrementally visualizes convex hull construction. Its time complexity is $O(nh)$, with n as the input points and h as the convex hull's vertex count.
3. **Graham Scan:** Activation of the algorithm occurs through a double-click on the canvas. Utilizing the Graham Scan method, it identifies the convex hull. The time complexity is $O(nlogn)$, primarily influenced by the sorting step of the input points
4. **Quick Elimination:** Random points create a rectangle for subsequent elimination, visualizing convex hulls before and after. Time complexity, O(n) with linear elimination.
5. **Kirkpatrick-Seidel:** Activated by a specific trigger, the Kirkpatrick-Seidel algorithm systematically refines the convex hull with a time complexity of $O(n \log h)$, where n is the input points' count and h is the convex hull's vertex count, proving efficient for large point sets.

6 Conclusion

The project effectively implements and examines geometric algorithms, offering a thorough grasp of functionalities and complexities. This is achieved through interactive user interfaces, detailed algorithmic steps, and comprehensive complexity analyses.

7 References

Line-sweep algorithm: `https://www.cs.uml.edu/~kdaniels/courses/ALG_504_F12/ORourkeCH7.pdf`
Kirkpatrick-Seidel: `https://pub.ista.ac.at/~edels/Papers/1983-06-ShapeSetPoints.pdf`

Appendix

.1 Line Intersection

.1.1 Orientation

```
def orientation(p, q, r): return 0 if (val := (q.y - p.y) * (r.x - q.x) - (q.x - p.x)
* (r.y - q.y)) == 0 else 1 if val > 0 else 2
```

.1.2 Slope Method

```
def slope(p1, p2):if p1.x == p2.x: return float('inf') return (p2.y - p1.y) / (p2.x - p1
```

.1.3 Sweep line Algorithm

```
def sweep_line_algorithm(intervals):
    events = sort_and_group_intervals(intervals) active_set = PriorityQueue()
    for event in events: handle_event(event, active_set)
```

.2 Convex Hull Algorithms

.2.1 Brute Force

```
def brute_force_convex_hull(P):
    return [p for pair in combinations(P, 2) for p in pair
    if all(Line(pair[0], pair[1]).side(x) == Side.LEFT for x in P)
    or all(Line(pair[0], pair[1]).side(x) == Side.RIGHT for x in P)]
```

.2.2 Jarvis March

```
def jarvis_march_convex_hull(P):
    start, hull, current = min(P, key=lambda p: p.y), [], min(P, key=lambda p: p.y)
    while (hull.append(current), current := min((p for p in P if p != current),
    key=lambda p: orientation(current, p, min(P, key=lambda q: p.y)))) != start: pass
return hull
```

.2.3 Graham Scan

```
    def graham_scan_convex_hull(P):
    start = min(P, key=lambda p: p.y)
    sorted_points = sorted(P, key=lambda p: (atan2(p.y - start.y, p.x - start.x), p))
    return [start] + sorted_points if len(sorted_points) <= 3
    else [start] + sorted_points[:2] + [p for p in sorted_points[2:]
    if orientation(sorted_points[0], sorted_points[1], p) != 1]
```

.2.4 Quick Elimination

```
    def quick_elimination_convex_hull(P):
    min_x, max_x = min(P, key=lambda p: p.x), max(P, key=lambda p: p.x)
    min_y, max_y = min(P, key=lambda p: p.y), max(P, key=lambda p: p.y)
    bounding_box = [min_x, max_x, min_y, max_y]
    remaining_points = [p for p in P if p not in bounding_box]
    return convex_hull(remaining_points)
```

.2.5 Kirkpatrick-Seidel

```
    def kirkpatrick_seidel_convex_hull(points):
    def kirkpatrick_seidel_recursive(points, dimension):
        if len(points) <= 3:    return points
        # Apply Kirkpatrick{Seidel algorithm recursively
    return kirkpatrick_seidel_recursive(points,0)
```