Comparative Analysis on Different Convex Hull Algorithm

A.F.M Minhazur Rahman
Lecturer
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

A.S.M.Montashir Fahim:1903110
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

Tasnuva Islam Shajutee:1903111
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

Golam Safayet Noor:1903112
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

S M Golam Rifat:1903113
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

Md Golam All Gaffar Tasin:1903114
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

Humayra Hassan Heya:1903115
Department of Computer Science and Engineering
Rajshahi University Of Engineering and Technology
Rajshahi,Bangladesh

I. Abstract

Convex hull concept play a fundamental role in computational geometry, finding widespread applications in various fields such as computer graphics, robotics and geographic systems information systems. This paper presents a comprehensive comparison of different convex hull algorithms, highlighting their computational complexity, efficiency, and robustness. Three prominent algorithms are examined, including the Gift Wrapping (Jarvis March), Graham Scan, QuickHull, We analyzed their theoretical backgrounds, implementation details, and performance on various datasets. The aim of this study is to assist researchers and developers in selecting the most suitable algorithm based on specific use cases and requirements.

II. Introduction

In geometry, the convex hull or convex envelope or convex closure of a shape is the smallest convex set that contains it. The convex hull may be defined either as the intersection of all convex sets containing a given subset of a Euclidean space, or equivalently as the set of all convex combinations of points in the subset. For a bounded subset of the plane, the convex hull may be visualized as the shape or polygon of n points in a given set of points and the polygon shape is smallest but contains all the given point inside the polygon [1].

This Convex hulls have wide applications in many fields. Within mathematics, convex hulls are used to study polynomials, matrix eigenvalues, and unitary elements, and several theorems in discrete geometry involve convex hulls. They are used in robust statistics as the outermost contour of Tukey depth, are part of the bagplot visualization of two-dimensional data, and define risk sets of randomized decision rules. Convex hulls of indicator vectors of solutions to combinatorial problems are central to combinatorial optimization and polyhedral combinatorics. In economics, convex hulls can be used to apply methods of convexity in economics to non-convex markets. In geometric modeling, the convex hull property Bézier curves helps find their crossings, and convex hulls are part of the measurement of boat hulls. And in the study of animal behavior, convex hulls are used in a standard definition of the home range [1].Also A convex hull algorithm can be used for collision avoidance. As it helps particles avoid collision, it can be translated to real-life scenarios to avoid vehicle collisions in cars and airplanes.

so the solution can be the construction of convex hull by brute force method that includes that the method for determining convex hull is to construct a line connecting two points and then verify whether all points are on the same side or not. There are such n(n – 1) /2 lines with n points, and each line is compared with the remaining n – 2 points to see if they fall on the same side. As a result, the brute force technique takes

$$O(N^3)$$

. [9]

This document go through to find questions that

1.is it possible to use faster algorithm to solve it discussed in algorithmic review

2.which algorithm should be suitable for different data points discussed in results and discussion

3.real world usability of algorithms discussed in discussion and conclusion

convex hull algorithm ensures the computational fast work and uses.but for fast computation the algorithm should be timely and also space perspective efficient. the graphical comparison ensure an visualize the difference in used problems

III. Algorithmic Review

Graham's Scan Algorithm:

Proposed by Ronald Graham in 1972, this is one of the earliest convex hull algorithms. It sorts the points based on their polar angles and uses a stack to efficiently compute the convex hull. Graham's Scan is simple to implement and has a time complexity of O(nlog(n)), making it quite efficient for moderate-sized point sets.

Jarvis March (Gift Wrapping) Algorithm:

Introduced by R. A. Jarvis in 1973, this algorithm is also known as the gift wrapping algorithm. It iteratively selects the point with the smallest polar angle and adds it to the convex hull. Jarvis March has a worst-case time complexity of O(nh), where "n" is the number of input points, and "h" is the number of points on the convex hull. In the worst case, h can be n, making it less efficient than some other algorithms. [1]

QuickHull Algorithm:

Developed by C. Barber, D. P. Dobkin, and H. Huhdanpaa in 1996, QuickHull is a divide-and-conquer convex hull algorithm. It recursively partitions the point set into two subsets, one inside the convex hull and one outside, and then constructs the convex hull by connecting the two hulls of the subsets. QuickHull is efficient in practice and often performs better than Graham's Scan and Jarvis March for large point sets.

IV. Algorithm Description

A. Graham Scan:

Graham scan is a method of computing the convex hull of a finite set of points in the plane with time complexity O (nlogn). It is named after Ronald Graham, who published the original algorithm in 1972. The algorithm finds all vertices of the convex hull ordered along its boundary. The first paper published in the field of computational geometry was on the construction of convex hull on the plane. In the late 1960, the best algorithm for convex hull was

$$O(n^2)$$

. At Bell Laboratories, they required the convex hull for about 10,000 points and they found out this

$$O(n^2)$$

was too slow. The algorithm is efficient in the sense that no matter how many points are on the boundary of the convex, it runs efficiently. In 1981, A.C.Yao, proved that it is optimal in the worst-case sense. The problem with Graham algorithm is that it has no obvious extension to three dimensions. The reason is that the Graham scan depends on angular sorting, which has no direct counterpart in three dimensions. Working Graham scan Algorithm begins with finding the base point which is the point with smallest Y- coordinate. Then by using the base point, we sort all the points based on angle made by them with base point. After sorting, we choose three points in sequence and check for point that leads to clockwise rotation. We remove the points that lead to a clockwise rotation and these are the point that does not form convex hull boundaries. [1]

1) ***Pseudocode:*** GRAHAM SCAN (Q)

1.Find p0 in Q with minimum y-coordinate (and minimum x-coordinate if there are ties)

2.Sorted the remaining points of Q (that is, Q minus p0) by polar angle in counterclockwise order with respect to p0.

3.TOP [S] = 0; Lines 3-6 initialize the stack to contain, from bottom to top, first three points.

4.PUSH (p0, S)

5.PUSH (p1, S)

6.PUSH (p2, S)

7.For i = 3 to m; Perform test for each point p3... pm.

8.do while the angle between NEXT TO TOP[S], TOP[S], and pi makes a non left turn

9.do POP(S)

10.PUSH (S, pi)

11.return S [2]

2) ***Graham scan algorithm simulation:***

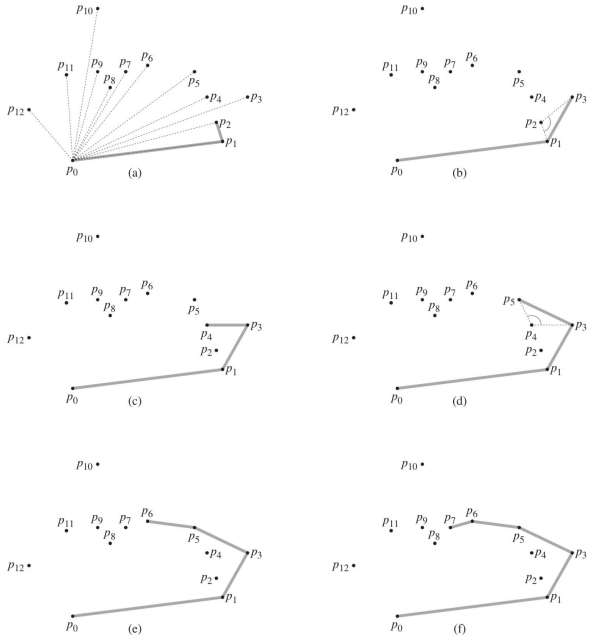

Fig 1: Graham scan algorithm simulation

B. Quick Hull algorithm:

We assume that the input points are in general position (i.e., no set of d+1 points defines a (d-1) flat, so that their convex

hull is a simplicial complex [Preparata and Shamos 1985]. We represent a d-dimensional convex hull by its vertices and (d-1)dimensional faces (the facets). A point is extreme if it is a vertex of the convex hull. Each facet includes a set of vertices, a set of neighboring facets, and a hyperplane equation. The (d-2)dimensional faces are the ridges of the convex hull. Each ridge is the intersection of the vertices of two neighboring facets. Quickhull uses two geometric operations: oriented hyperplane through d points and signed distance to hyperplane. It represents a hyperplane by its outward-pointing unit normal and its offset from the origin. The signed distance of a point to a hyperplane is the inner product of the point and normal plus the offset. The hyperplane defines a halfspace of points that have negative distances from the hyperplane. If the distance is positive, the point is above the hyperplane. [3]

1) ***Pseudocode: [4]:*** Input = a set S of n points
Assume that there are at least 2 points in the input set S of points
function QuickHull(S) is
Find convex hull from the set S of n points
Convex Hull :=
Find left and right most points, say A and B, and add A and B to convex hull
segment AB divides the remaining n-2 points into 2 groups S1 and S2
where S1 are points in S that are on the right side of the oriented line from A to B,
and S2 are points in S that are on the right side of the oriented line from B to A
FindHull(S1, A, B)
FindHull(S2, B, A)
Output := Convex Hull
end function
function FindHull(Sk, P, Q) is
Find points on convex hull from the set Sk of points that are on the right side of the oriented line
from P to Q
if Sk has no point then
return
From the given set of points in Sk, find farthest point, say C, from segment PQ add
point C to convex hull at the location between P and Q
Three points P, Q, and C partition the remaining points of Sk into 3 subsets: S0, S1, and S2
where S0 are points inside triangle PCQ, S1 are points on the right side of the oriented
line from P to C, and S2 are points on the right side of the oriented line from C to Q. FindHull(S1, P, C)
FindHull(S2, C, Q)
end function

2) quick hull algorithm simulation::

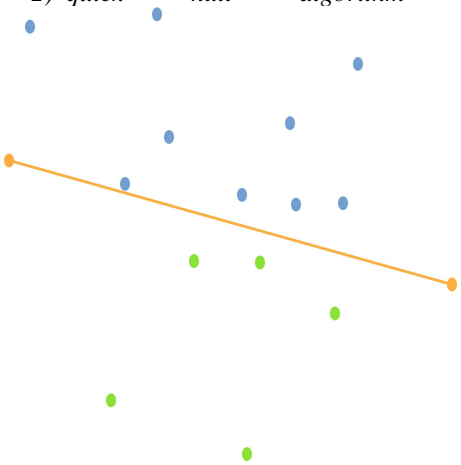

Fig 2: Quick hull algorithm simulation step 1

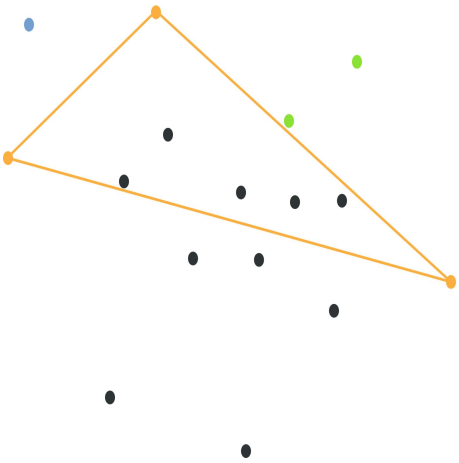

Fig 3: Quick hull algorithm simulation step 2

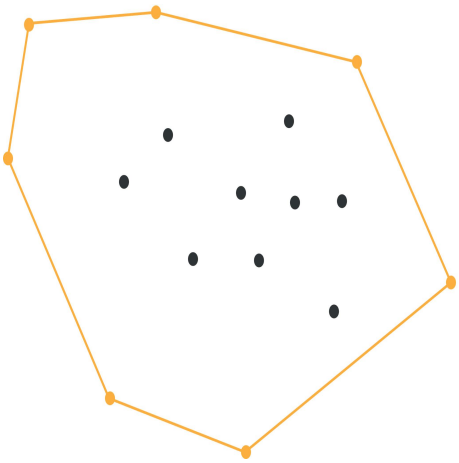

Fig 4: Quick hull algorithm simulation step 3

C. Jarvis march(Gift wrapping) Algorithm:

The gift wrapping algorithm begins with i=0 and a point

$$p_0$$

known to be on the convex hull, e.g., the leftmost point, and selects the point

$$p_i + 1$$

such that all points are to the right of the line

$$p_i = p_i + 1$$

. This point may be found in O(n) time by comparing polar angles of all points with respect to point pi taken for the center of polar coordinates. Letting i=i+1, and repeating with until one reaches

$$p_h = p_0$$

again yields the convex hull in h steps. In two dimensions, the gift wrapping algorithm is similar to the process of winding a string (or wrapping paper) around the set of points. [5]

1) ***pseudocode: [5]****:* algorithm jarvis(S) is
S is the set of points
P will be the set of points which form the convex hull. Final set size is i.
pointOnHull = leftmost point in S
which is guaranteed to be part of the CH(S)
i := 0
repeat
P[i] := pointOnHull
endpoint := S[0] ::initial endpoint for a candidate edge on the hull
for j from 0 to abs(s) do
::endpoint == pointOnHull is a rare case and can happen only when j == 1 and a better endpoint has not yet been set for the loop
if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to endpoint) then
endpoint := S[j] ::found greater left turn, update endpoint
i := i + 1
pointOnHull = endpoint
until endpoint = P[0] :: wrapped around to first hull point

2) Jarvis march algorithm simulation::

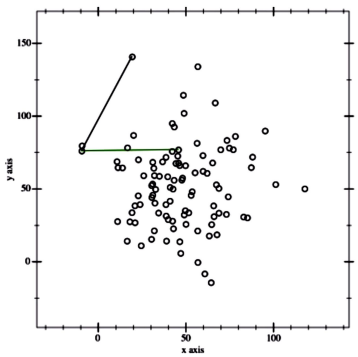

Fig 5: Jarvis march algorithm simulation step 1

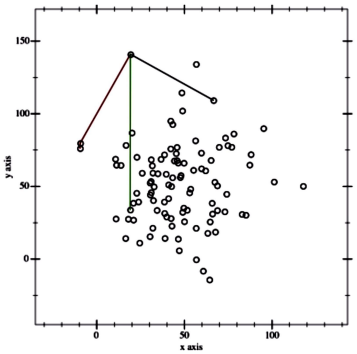

Fig 6: Jarvis march algorithm simulation step 2

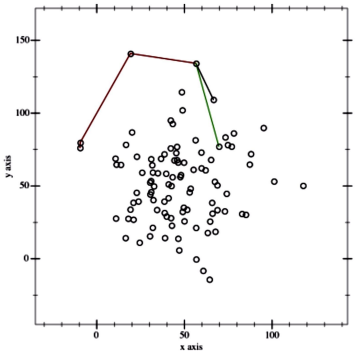

Fig 7: Jarvis march algorithm simulation step 3

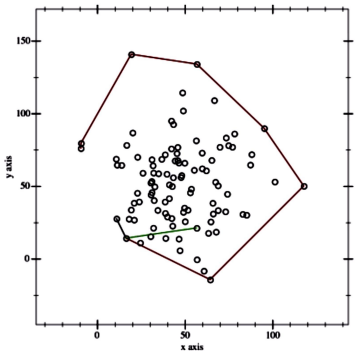

Fig 8: Jarvis march algorithm simulation step 4

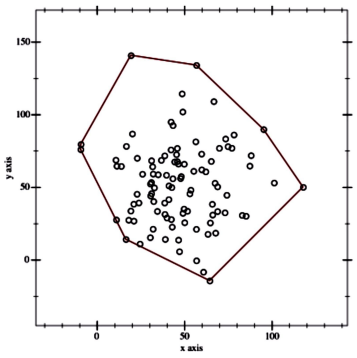

Fig 9: Jarvis march algorithm simulation step 5

V. PERFORMANCE MEASURE

For the performance measurement of this three algorithms we used randomly generated data points at three levels medium data points,large data points,worst case data points and the execution time taken as performence parameters according to data size.

1.Medium Data Points: The number of data points is limited to 3000

2.Large Data Points: The number of data points is above 5000

3.Worst case Data Points: All the data points is included in the polygon hull and it can be large or medium data points

VI. EXPERIMENTAL RESULT

A. Graham Scan algorithm

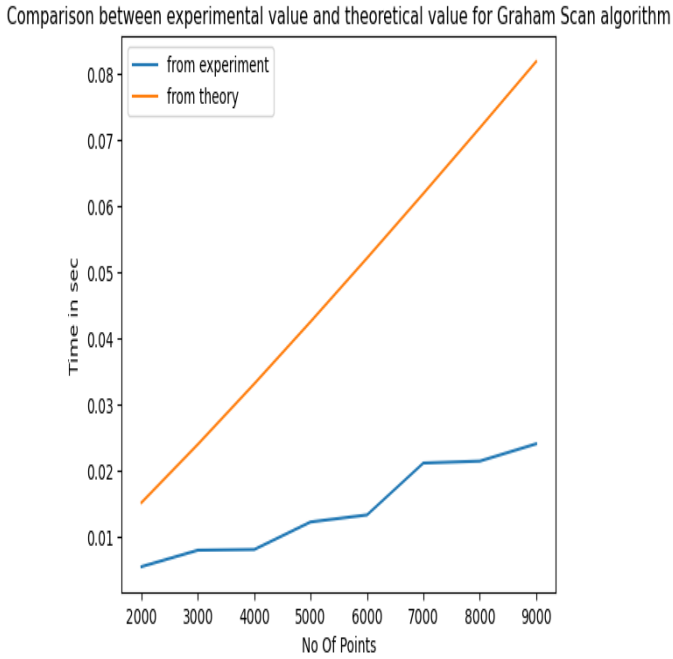

Fig 10: Graham scan algorithm theoretical and practical result

theoretically the time complexity for graham scan algorithm is O(nlogn) and the graph is upword [6].after experiment with data points we got better and improved result

B. Quick Hull algorithm

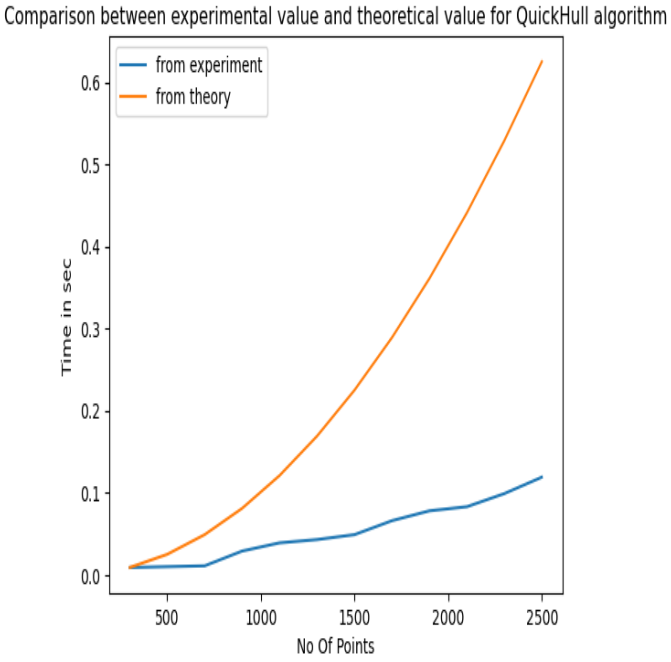

Fig 11: Quick Hull algorithm theoretical and practical result

The quick hull algorithm has a worst-case time complexity of

$$O(n^2)$$

, but an expected time complexity of 0(nlog n), when all points are hull vertices.but the experimental result is improved

C. Jarvis march algorithm

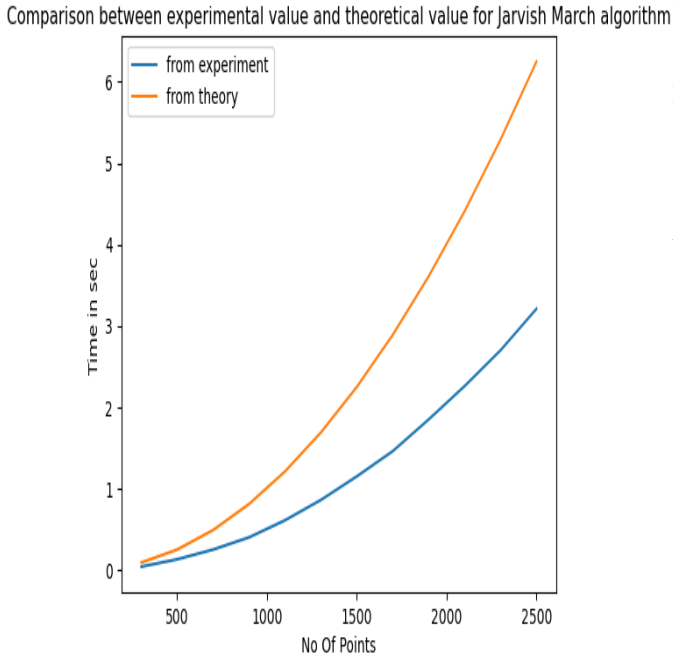

Fig 13: Jarvis march algorithm theoretical and practical result

The jarvis march algorithm has a worst-case time complexity of

$$O(n^2)$$

,when all point in hull vertices, but an average time complexity of 0(nh),where h is hull vertices nymber when all points are not hull vertices.but the experimental result is improved. [8]

D. Worst case medium data points

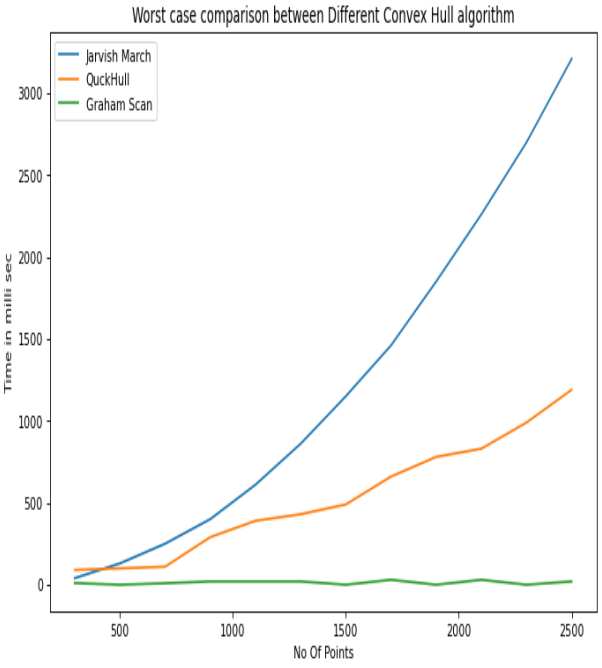

Fig 14: Worst case medium data points algorithm performance practical result

Theoretically for moderate size data points at worst case graham scan algorithm performs very much better than quick hull and jarvis march and we experimentally derived that graham scan is almost linear level where jarvis march is quadratic complexity

E. Worst case large data points

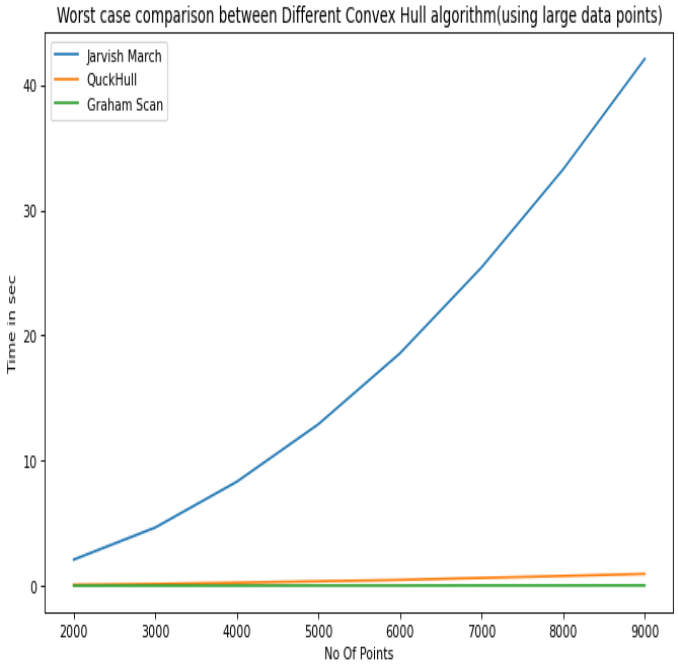

Fig 15: worst case large data points algorithm performance practical result

Theoretically for large size data points at worst case graham scan algorithm performs very much better than quick hull and jarvis march and we experimentally derived that graham scan is very fast where jarvis march is quadratic complexity

VII. Discussion

In this paper, we have presented a comprehensive comparison of three well-established convex hull algorithms: Graham Scan, Jarvis March, and Quick Hull. Our objective was to provide valuable insights into the strengths and weaknesses of these algorithms, aiding researchers and practitioners in making informed choices when selecting an appropriate convex hull algorithm for their specific applications.

Through a detailed analysis, we have highlighted the following key findings:

1. Algorithm Complexity: We discussed the time complexity of each algorithm, showcasing that Graham Scan and Jarvis March are both

$$O(n^2)$$

in the worst case, while Quick Hull exhibits an average-case complexity of O(nlog(n)). Quick Hull's performance

advantage in terms of time complexity becomes evident in larger datasets.

2. Robustness and Stability: We emphasized that Jarvis March is the most robust algorithm, guaranteeing a correct convex hull even with degenerate cases, such as collinear points. In contrast, Graham Scan and Quick Hull may require additional preprocessing or careful implementation to handle such cases.

3. Implementation Simplicity: Graham Scan and Jarvis March are relatively straightforward to implement and understand, making them suitable choices for educational purposes or when a quick implementation is needed. Quick Hull, on the other hand, demands a more complex implementation but offers superior performance for sufficiently large datasets.

4. Practical Considerations: We discussed real-world scenarios where each algorithm shines. For instance, Graham Scan is often preferred in applications where simplicity and robustness are more critical than speed. Quick Hull, conversely, is well-suited for large-scale computational geometry tasks due to its efficient average-case time complexity.If we need to work with large dataset it is important to use Graham Scan.

In conclusion, the choice of which convex hull algorithm to use should depend on the specific requirements and constraints of the problem at hand. Graham Scan and Jarvis March are dependable choices for smaller datasets and when robustness is paramount. Meanwhile, Quick Hull should be the preferred option when dealing with larger datasets and where average-case efficiency is crucial.

Ultimately, the decision should consider not only the theoretical properties of these algorithms but also the practical implications for the intended application. Further research in this area could explore hybrid approaches or optimizations that combine the strengths of these algorithms to address a wider range of use cases effectively.

VIII. Conclusion

To sum up, this comparative examination offers valuable perspectives on the performance attributes of three well-known convex hull algorithms: Graham Scan, Quick Hull, and Jarvis March. While each algorithm possesses its own strengths, the selection of the most suitable one should depend on the particular problem scenario and the nature of the input data. By conducting a comprehensive assessment of their theoretical underpinnings, implementation intricacies, and practical performance, this paper empowers researchers and professionals with the insights needed to make well-informed choices when addressing convex hull challenges within their respective applications.

References

[1] http://en.wikipedia.org/wiki/Grahamscan
[2] http://cs.indstate.edu/ jtalasila/convexhull.pdf
[3] https://www.cise.ufl.edu/ ungor/courses/fall06/papers/QuickHull.pdf
[4] https://en.wikipedia.org/wiki/QuickhullPseudocodefor2Dsetofpoints
[5] https://en.wikipedia.org/wiki/Giftwrappingalgorithm
[6] wikipedia graham scan algorithm time complexity
[7] wikipedia quickhull algorithm time complexity
[8] wikipedia jarvis march algorithm time complexity
[9] wikipedia convex hull brute force algorithm