An Algorithm to Find Convex Hull Based on Binary Tree

Muhammad Sharif, Safdar Khan, Sadaf Jameel Khan, Mudassar Raza
Department of Computer Sciences
COMSATS Institute of Information Technology, Wah Cantt
muhammadsharifmalik@yahoo.com, skhan_wah@yahoo.com, iota_labs@yahoo.com, mudassarkazmi@yahoo.com

Abstract—In this paper, a new algorithm to find convex hull is presented which keeps the points into a form of nested binary tree. There are number of methods available for finding the convex hull but all of them are time consuming because they are based on comparing distances between the points and rotation of angles, whereas the proposed technique is based on binary tree which is efficient and more accurate than the existing algorithms as it decreases the computational cost and finds precise boundary of the object. Another main advantage of the proposed technique is the efficient memory usage because of binary tree structure.

Key Words: -Convex Hull, Geo-informatics, Binary Tree.

I. INTRODUCTION

The convex hull means a smallest convex region which encloses a specified group of points. In 2D-Plane, convex hull can be viewed as if a rubber band is stretched around the points of the hull and all other points must lie under this rubber band [1]. Computing of a finite set convex hull is usually used in image processing, pattern recognition, nesting problems, geographic information system and geo-informatics [2], [3].

In this paper, a fast approach is being presented to compute convex hull which is based on binary tree. In section II, an overview of the convex hull algorithms has been given. Section III discusses the planar algorithm which is the motivation for this new technique. The proposed technique has been presented in section IV followed by comparison of quick hull and new technique in section V with the help of examples. Conclusion and references have been appendixed in section VI.

II. EXISTING ALGORITHMS

A number of algorithms are used to find the convex hull. A brief overview of two of them is follows:

- Graham's Scan Algorithm
- Quick hull Algorithm

A. Graham's Scan

This algorithm works efficiently on sorted points [2]. At random points, it computes hull in *O(n log n)* time but if the points are sorted (which make this algorithm slow), it takes the time of *O(n)*. Grahm Scan regularly checks the sorted list of points and deletes the non convex points. The rest of sorted points are considered as the convex hull of extreme points. [1].

B. Quick Hull

This algorithm is much faster than the previous algorithms [4]. It takes *O(n log n)* time to compute hull but $O(n^2)$ in worst case [1]. In first step, it locates the leftmost and the rightmost points. Then it picks the points above that line at the farthest distance. Ultimately, it discards the inner points of the triangle formed by joining the leftmost, right most and topmost points. By repeating the same process below the line, another triangle is formed by joining the rightmost, leftmost and the bottommost points, thus discarding many inner points again. After that it finds the point which is at maximum distance from the line and discards all inner points. By repeating this process until all inner points are discarded, the required convex hull is found.

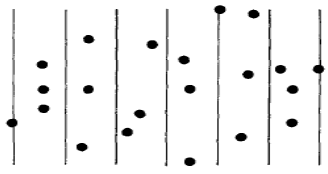
III. THE PLANAR APPROXIMATE ALGORITHM

This section represents an algorithm to compute approximate convex hull (Fig (a)). In the first step, the max and min values of the x-axis are calculated. Then the area is divided into equal size strips. Then maximum and minimum points are found according to y-axis from those strips (Fig (b)). Finally, the convex hull is constructed by taking extreme points using any convex hull algorithm Fig(c).

The running time complexities of different steps are as follows:

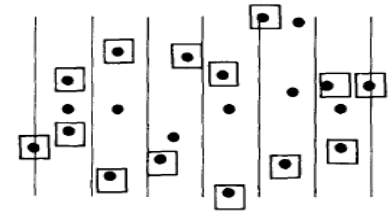
- Calculating min and max values: *O (n)*.
- Initializing strips array: *O (k)* (where k is total strips). Calculating each strip extreme points: *O(n)*.
- Calculating the convex hull: *O (k)* time.
- The total running time: *O (n + k)*.

It is an approximate algorithm and is used in places where too much accuracy is not required [2].

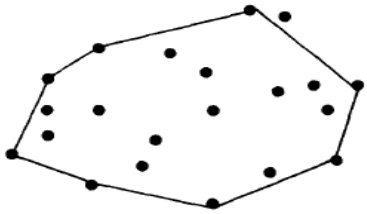


(a) Put Bin Points.

978-1-4244-4873-9/09/$25.00 ©2009 IEEE



(b) Calculate the extreme values.



(c) Build hull of extremes.

Fig. 1. The Planar Approximate Hull Algorithm. Reprinted from [2]

IV. PROPOSED TECHNIQUE TO COMPUTE CONVEX HULL

The proposed algorithm is based on binary tree which is different from all previous algorithms. In this algorithm first the tree is created according to x-axis. The root or head node has abscissa value of any point and two pointers to hold next and previous node. The new node added to the tree is placed according to abscissa of the point. If the abscissa of the point added is greater than the root, it is placed in the next pointer otherwise it goes to the previous pointer. Each node will also save minimum y-value and maximum y-value against that abscissa. If the abscissa value of the next point is equal then its y value will be checked and abscissa value will be discarded. In this way, a maximum and minimum point against an x-value is determined depending on the values of y-axis. Scan the tree using in-order traversal twice; in the first traversal take the y-min and in the second traversal take y-max and get required hull of points. This technique gives closer boundary points and eliminates all the inner points. This proposed technique will have the same time complexity as that of binary tree.

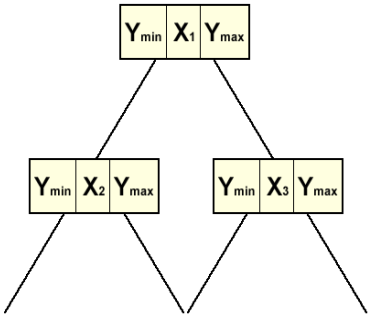


Fig. 2. The tree structure.

ALGORITHM: THE PROPOSED APPROACH

1: **given** a set of points with x (abscissa) and y (ordinate).

2: **arrange** all the points in binary tree according to x coordinate
- Make abscissa(x1) of first point the root of tree
- If x2 < root of tree
  - Then put in left
- Else
  - Put in right

3: also **save** the ordinate (y coordinate) of the point in the node with two variables i.e. min and max.
- If y(x) < 0
  - Min = y
- else
  - Max = y

4: for the same x-value just **modify** the max and min variables and discard the x-value.

5: now both y-min and y-max have a corresponding x-value.

6: **inorder** (x) : x is root of a sub tree.
- If x != NULL
- Then Inorder(left(x));
  - Print x and y-min
- Output key(x) and y-min;
- Inorder(right(x));
  - Print x and y-min

7: **inorder** (x) : x is root of a sub tree.
- If x != NULL
- Then Inorder(left(x));
  - Print x and y-max
- Output key(x) and y-max;
- Inorder(right(x));
  - Print x and y-max

8: **join** the last point of first in-order with the last point of second in-order and also join the first point of first in-order with the first point of second in-order.

V. COMPARISON OF QUICK HULL AND THE PROPOSED ALGORITHM

The following two examples elaborate the comparative procedures and results of both the Quick Hull and the proposed technique.

Example 1:

For the purpose of experiment, the following points have been randomly generated and convex hull of these numbers is constructed first by Quick Hull and then by the proposed technique.

TABLE I.
Randomly Generated Points

Points Number	Points Values
1	(1,5)
2	(4,4)
3	(3,3)
4	(3,1)
5	(6,2)
6	(1,2)
7	(8,0)
8	(1,-1)
9	(3,-2)
10	(6,-2)
11	(4,-3)
12	(1,-4)
13	(-6,-1)
14	(-4,-2)
15	(-2,-1)
16	(-2,-5)
17	(-4,4)
18	(-2,3)
19	(-4,1)
20	(-8,2)

The randomly generated points are plotted on the paper as shown in the figure below.

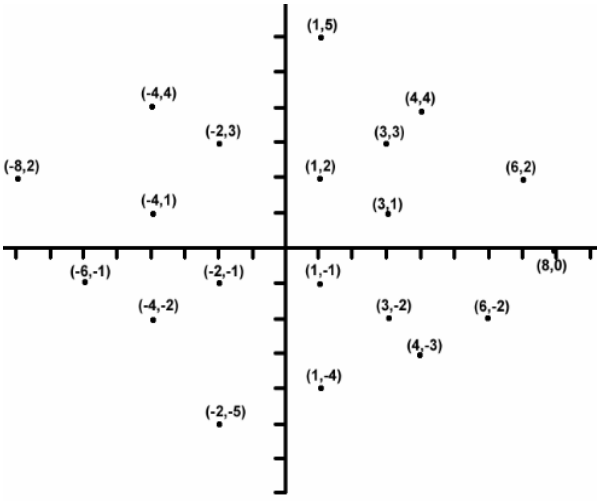


Fig. 3. Points generated randomly and drawn on graph

A. By Quick Hull

By applying Quick Hull on the points shown above in Fig. 3, convex hull is found in two steps given below:

STEP 1:

In this step left most (-8, 2), right most (8, 0), topmost (1, 5) and bottommost (-2, 5) points are found to make a quadrilateral ABCD. All the inner points of the said quadrilateral are discarded. (Fig. 4).

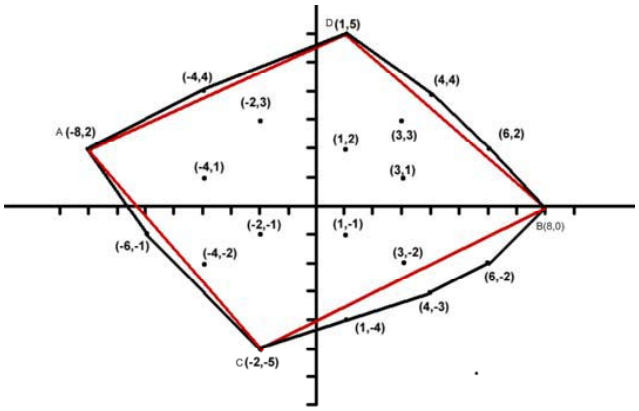


Fig. 4. Process of quick hull, mark quadrilateral points.

STEP 2:

In this step, the points having maximum perpendicular distance on each side of the quadrilateral ABCD are found and the boundary of quadrilateral is extended to those points discarding the inner points. Repetition of this step finally makes the convex hull. The result is shown in Fig 5.

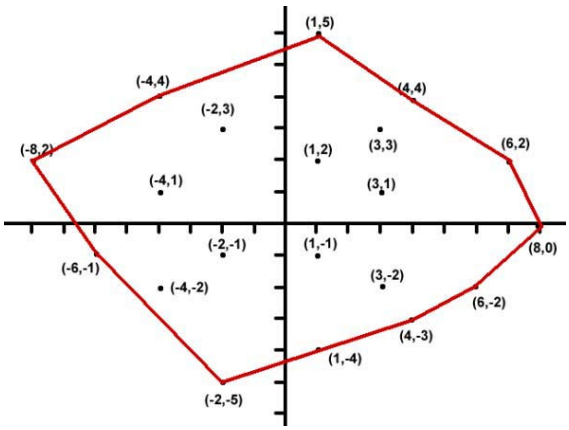


Fig. 5. Quick Hull result

B. By Proposed Algorithm

Now proposed algorithm is applied on the points shown in the Fig.3.

Any point can be taken as a head node. Let the pint (1, 2) be this head node. Let the next point be (3,-2). As abscissa 3 of this point is greater than abscissa 1 of the previous point, so it comes in the next node. If the next point is (1,5), y-max of this point contains abscissa 1, which is greater than the previously saved node, therefore ordinate 2 is replaced with 5 and as a result point (1,2) will be discarded.

The same process is applied to all the remaining points resulting in the emergence of the tree shown below in Fig. 6.

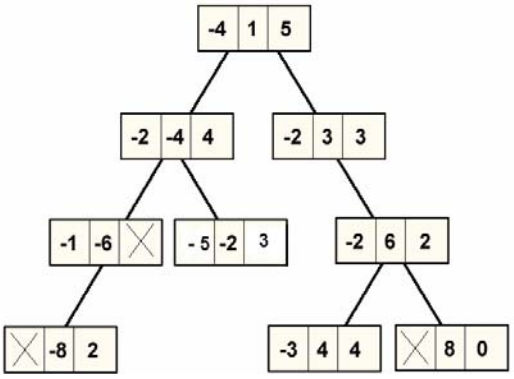


Fig. 6. The binary tree process for random points

The selected points in the above tree in fact show the required convex hull as shown in Fig. 7.

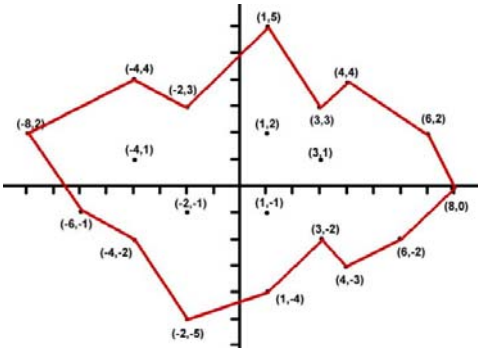


Fig. 7. The binary tree result (convex hull)

Example 2:

Instead of taking randomly generated points, a real object is taken for finding the convex hull.

A fish is taken as an object and then points are marked on it.

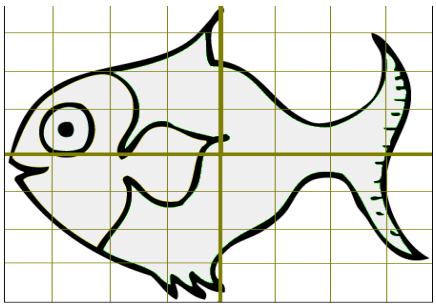


Fig. 8. The object fish to be taken as real example

To implement new proposed algorithm fixed points are taken on the fish as shown in Fig 9.

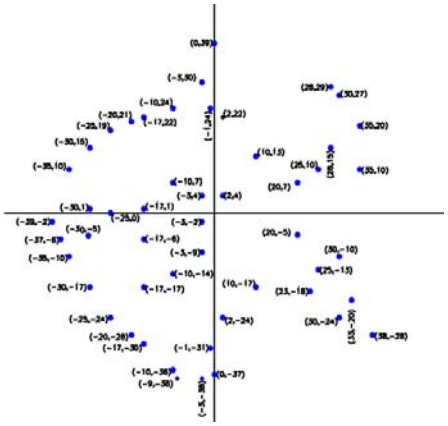


Fig. 9. Fixed number of points taken on fish, which are given to algorithm to find convex hull

First we implement the Quick Hull on the marked points as shown in the Fig. 9 and then apply the proposed technique to find convex hull.

A. By Quick Hull

By applying Quick hull, convex hull is formed in two steps:

STEP 1:

In this step, left most (-39,-2), right most (38,-28), topmost (0, 39) and bottommost (-9,-38) points are used to get the quadrilateral ABCD as shown in Fig. 10.

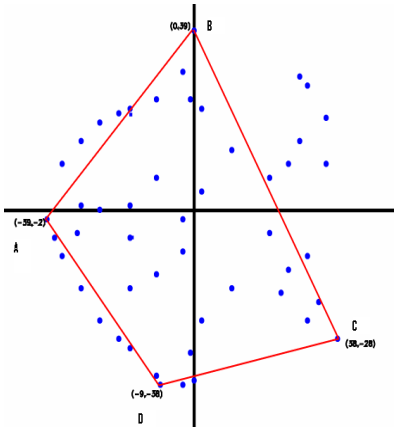


Fig. 10. Quick Hull first step result

STEP 2:

In the second step, the points having maximum perpendicular distance on each side of the quadrilateral ABCD are located and the boundary of the quadrilateral is extended to those points thereby discarding all the inner points. By repeating this step, time and again, the final convex hull is found as shown in Fig. 11.

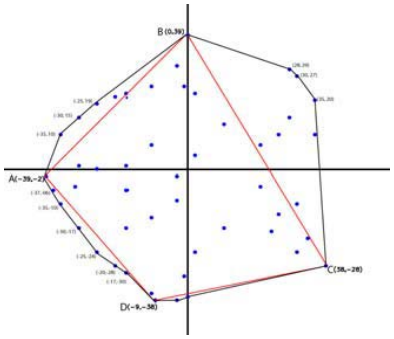


Fig. 11. Quick Hull final result

B. By Proposed Algorithm

Now proposed algorithm is applied on the points shown in Fig.9.

Take any point, let (2, 4) as head node. Let the next point be (20,-5). As the abscissa 20 of this point is greater than abscissa 2 of the previous point, so it comes in the next node. Let the next point is (2, 22), abscissa 2 of this point is already present in the head node. Check its y value and modify because y-max contains only the maximum value of ordinate and 22 is maximum than 4. In this way point (2, 4) is discarded. Now let the next point is (2,-24). As its abscissa 2 is already present and its ordinate is negative so modify the y-min value against that abscissa and that point is included in the tree node. In the same way take all points and construct the tree as shown in Fig. 12.

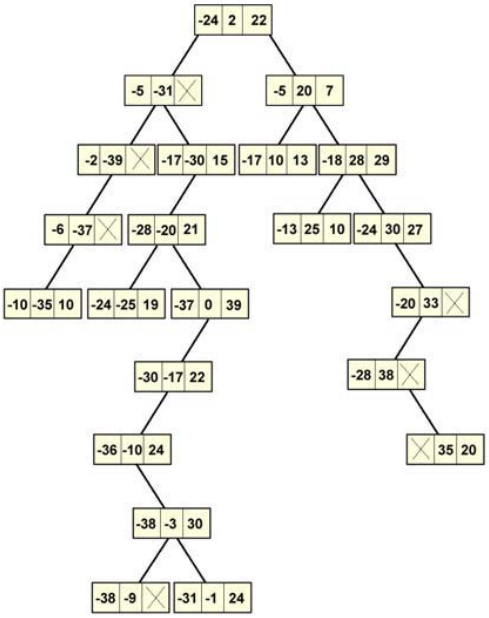


Fig. 12. The binary tree process

For finding the convex hull of the above tree, do in order traversal twice. In the first traversal, take abscissa of left most nodes and only use its y-min value to draw the lower hull of the object till the last node of traversal. In the second traversal take abscissa and use its y-max value to draw the upper hull of the object. In this way convex hull of the object is constructed. (Fig. 13).

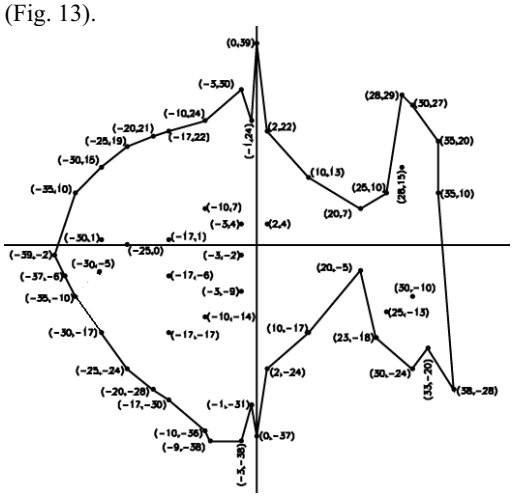


Fig. 13. The binary tree result

VI. RESULTS

We checked the proposed technique in terms of the following two parameters:

i. Accuracy
ii. Efficiency in term of processing time

A. Accuracy

For accuracy we checked both the techniques (existing and proposed) for random points on a real object and found the hull.

1. Random Points

First take the randomly generated 20 points and pass them to both: existing technique (Quick hull) and proposed technique (Tree based algorithm).

2 Real objects

Take a real object (a fish) and mark points on its boundary and body. These points are again passed to both: existing technique (Quick Hull) and proposed technique (Tree based algorithm).

3. Observation

Hulls formed by the existing and proposed techniques are same. Moreover, hull formed by new technique is more effective in terms of accuracy and precision than the existing technique because it gives accurate boundary of the object.

B. Efficiency in terms of processing time

To check efficiency in terms of processing time, points from 1000 to 500000 are generated and passed to the two existing techniques, Graham Scan and Quick hull algorithms which are the fastest among all the exiting techniques. Then

the same points are passed to the proposed Tree based algorithm technique and the processing time is noted.

1. Observation

From the table (Table II) obtained by two existing techniques and the new proposed technique, we observed that the processing time difference to calculate hull is smaller for small number of points but it gives huge difference as number of points are increased. From the Graph (Fig. 14) it is obvious that the proposed technique is very efficient in terms of processing time.

TABLE II
Results of previous and proposed techniques

Points	Graham(sec)	Quick(sec)	Tree(sec)
1000	0.001366	0.01	0.000372
5000	0.007148	0.016	0.000632
10000	0.014556	0.016	0.004051
20000	0.034326	0.015	0.000991
30000	0.051628	0.016	0.001082
40000	0.070418	0.032	0.000699
50000	0.081346	0.047	0.000718
70000	0.116849	0.063	0.000728
90000	0.148008	0.078	0.000721
100000	0.165322	0.084	0.00072
500000	0.857992	0.375	0.000729

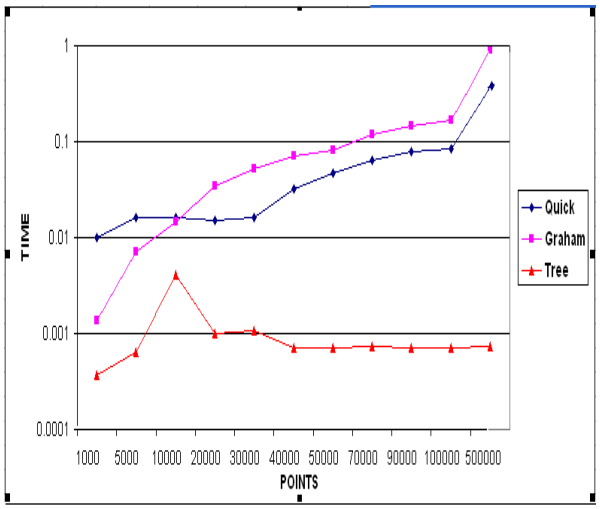


Fig. 14. Graph of previous and proposed techniques

VII. CONCLUSION

From the above two examples, it has been proved that the new algorithm technique not only finds the convex hull but also gives precise boundary of the object as compared to all previous algorithms. The new method also decreases the computational cost significantly. Another main advantage of the proposed technique is the efficient memory usage. The classical algorithm allocates the memory for all points but the proposed algorithm deals with the pointers of data items at node which are assigned to the next point in the hull without allocating extra space for all points. Therefore, this algorithm works faster than other algorithms (according to our knowledge) as tree structure is faster than earlier algorithms.

REFERENCES

[1] Chadnov, R.V. Skvortsov, A.V. , "Convex Hull Algorithms Review", The 8th Russian-Korean International Symposium on IEEE Publication Date: 26 June-3 July 2004
Volume: 2, On page(s): 112- 115 vol. 2

[2] Jon Louis Bentley and Mark G. Faust, "Approximation Algorithms for Convex Hulls" Communications of the ACM, January 1982, Volume 25

[3] Fadi Yaacoub, Yskandar Hamam, Antoine Abche, Charbel Fares, "Convex Hull in Medical Simulations: A New Hybrid Approach",. 32nd Annual Conference on IEEE Industrial Electronics, IECON 2006, Nov. 2006

[4] C. BRADFORD BARBER University of Minnesota DAVID P. DOBKIN, "The Quickhull Algorithm for Convex Hulls" ACM Transactions on Mathematical Software, Vol. 22, No. 4, December 1996, Pages 469–483.

[5] Gerth St_lting Brodal_ Rolf Fagerberg_ Riko Jacob, "Cache Oblivious Search Trees via Binary Trees of Small Height", Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. San Francisco, California. Pages: 39 – 48. Year of Publication: 2002