Advances in Computer Science and Information Technology (ACSIT)
p-ISSN: 2393-9907; e-ISSN: 2393-9915; Volume 4, Issue 1; January-March, 2017, pp. 38-41
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/Publication.html

Space Efficient Construction of Convex Hull

Deepak Kumar Singh[1], Sammita Chakravarti[2] and Arpit Kumar Singh[3]

[1,2,3]National Institute of Technology Durgapur Computer Science and Engineering
E-mail: [1]dksingh202015@gmail.com, [2]sammitachak1994@gmail.com, [3]arpitsinghnitd@gmail.com

Abstract—*The convex hulls of sets of n points in two and three dimensions can be determined with O (nlogn) operations. The presented algorithms use the "divide and conquer" technique and recursively apply a merge procedure for two non-intersecting convex hulls. Since any convex hull algorithm requires at least O (nlogn) operations, the time complexity of the proposed algorithms is optimal within a multiplicative constant. To solve the above mentioned problem, with constant space and O(nlogn) time complexity we have to solve another problem which is merging of two convex polygons represented by an array using constant space and O(n) time complexity.*

1. INTRODUCTION

An algorithm is called space efficient, or in-place if it requires no extra memory apart from the space required for storing the inputs. In-place algorithms are tricky to devise due to limited memory considerations. Quick sort and Heap sort are some well-known examples of in-place algorithms.

Several classical problems of computational geometry have been revisited with space requirements in mind. Two dimensional convex hull of points is one of them. Space efficient algorithms have many advantages over their classical counterparts. Mostly, they necessitate little memory beyond the input itself, so they typically avoid virtual memory paging and external I/O bottlenecks (unless the input itself is too large to fit in primary memory, in which case I/O efficient algorithms can be used). They also typically exhibit better locality of reference. Convex hull of a set of points is the smallest convex set that contains the points. The convex hull is a fundamental construction for mathematics and computational geometry. For example, Boardman [1993] uses the convex hull in his analysis of spectrometry data, and Weeks [1991] uses the convex hull to determine the canonical triangulation of cusped hyperbolic 3-manifolds. Other problems can be reduced to the convex hull for example, half space intersection, Delaunay triangulation, Voronoi diagrams, and power diagrams. In his review article, Aurenhammer [1991] describes applications of these structures in mesh generation, file searching, cluster analysis, collision detection, crystallography, metallurgy, urban planning. Cartography, image processing, numerical integration, statistics, sphere packing, and point location.

Bridge is a line segment joining a vertex on the left and a vertex on the right that does not cross the side of either polygon. What we need are the upper and lower bridges for the two convex polygons and using them we can merge the two convex polygon into one in O (n) time complexity and O (1) space complexity. In this paper we will be solving another problem of merging two convex polygons using constant memory (in-place). We will be using this to solve the merging step of solving the original problem. We assume that the initial set of points are given in the form of an array. We will be using system stack which would be required for recursion.

Divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (divide), until thesebecome simple enough to be solved directly (conquer). The solutions to the sub-problems are then combined to give a solution to the original problem.

2. PROBLEM DEFINITION

Construct a convex hull of given 'n' points in a two dimensional plane in O (nlogn) time and O(1) space complexity.

3. APPROACH

Without the loss of generality we assume that the points are given in an array. We are implementing a divide and conquer algorithm to compute convex hull of the 'n' points. In the process of computing the required convex hull, we will be using the concept of merging two convex polygons in a space efficient manner.

4. PROCEDURE

4.1. Pre-processing

Sort the array which contains the given 'n' points with respect to their X-coordinate. If two points have the same X-coordinate then the lower one with the lower Y-coordinate will be placed to the left of the one having higher Y-coordinate. Without the loss of generality we assume, no two points coincide. We will be using Heap Sort for accomplishing the sorting.

4.2. Algorithm Convex Hull

Input: A set S = {a_1, a_2 , a_3,….a_n} **Output** :

The Convex hull CH(S) of S.

1. Divide the array into two parts S_1 ,S_2 taking the X – coordinate of the median of the points in S as reference.

   S1= {a1,a2,a3….an/2}and S2={a(n/2+1),…an}

2. Apply recursive Algorithm CH to S1 and S2 to get CH (S1) and CH (S2).
3. Apply a merge algorithm to CH(S1) and CH (S2) to obtain CH(S) and halt.

In the dividing step, we divided S into two equal parts based on their X –coordinate to reduce the complexity of finding the median x coordinate, we have presorted the array on the basis of the x- coordinate of the points set.

As a base of the algorithm we will find the convex hull of the points using any O (n2) convex hull construction algorithm. At the base level there won't be more than five points in any subdivision. It will require about twenty five operations per sub- division and there would be a maximum of [n/5] such sub- divisions .Hence requiring no more than [n/5]*25 operations which is linear in time. For the subsequent level we will use a constant space convex polygon merging algorithm to merge convex hulls resulting from two sub-divisions into a single convex hull. The merging algorithm is taken from the Merging of two convex polygons which is explained in the next section.

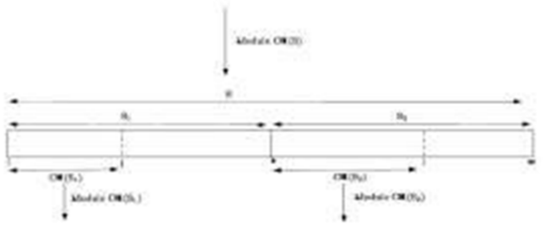

Fig. 1: Dividing array S after the divide step of algorithm Convex Hull (CH)

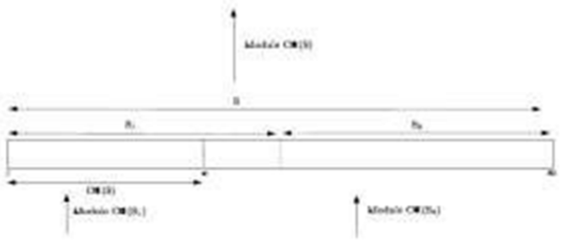

Fig. 2: Array S after the merging step of Convex Hull Algorithm

4.2. Merging Two Convex Polygons

4.2.1. Procedure to find the upper bridge of the two convex polygons

1) Start with a bridge which joins the rightmost point of a left polygon and left most point of polygon on the right. A bridge is guaranteed if you join rightmost vertex on the left to the leftmost vertex on the right.2) Keeping the left end of the bridge fixed, see if the right end can be raised. That is, look at next vertex on the right polygon going clockwise, and see whether that would be a (better) bridge. Otherwise, see if the left end can be raised while the right end remains fixed.

3) If made no progress in step 2 (cannot raise either side) then stop else repeat step 2.

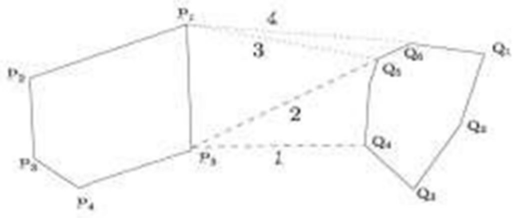

Fig. 3: Finding Common Upper Tangent of two convex polygon

For the previous in above figure-1 initially we consider the bridge-1 connecting the vertices P5 and Q4 of left and right convex polygon respectively. As mentioned P5 is the rightmost point of the left polygon and similarly Q4 is the leftmost point of the right polygon.

Now as bridge 1 is not tangent to the polygon on the right so the right end of the bridge 1 is raised to point Q5 forming the bridge 2.We check if bridge 2 is tangent to the right polygon . Since the answer is yes, we raise the left end of the bridge 2 to point P1.Now bridge 3 joining P1 and Q5 is a tangent to the left polygon.

Now we check if bridge-3 is tangent to the right polygon.

Since the answer is No, we raise the right end of the bridge 3 to point Q6.Since bridge-4 P1 and Q6 is a tangent to the right polygon, we check bridge-4 is a tangent to the left polygon. Since bridge-4 is tangent to the left polygon as well so we stop our procedure and we get bridge -4 as common upper tangent to the two polygons. Similarly we can get common lower tangent to the two polygons.

4.2.2. Merging The Two Convex Polygons represented By an Array

Let us assume the vertices of the left convex polygon in the clockwise direction be P1, P2, P3, Pn and the vertices of the right polygon in the clockwise direction be Q1, Q2, Q3 ... Qm .The two polygons will be represented in the form of an array as shown in the below figure.

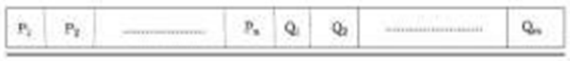

Fig. 4: The initial array before merging the convex polygons

Let Pi be the left end point of the common upper tangent and Pj be the left end point of the common lower tangent of the two convex polygons. Similarly ,Qa be the right end point of the common upper tangent and Qb be the right end point of the common lower tangent of the two convex polygons.

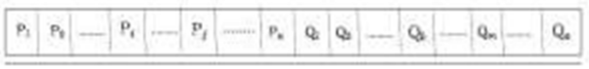

Fig. 5: Initial array with the points Pi, Pj, Qa, Qb

We will be modifying the left polygon such that the vertices Pj to Pi will appear at the starting of the array in clockwise direction .Rest of the vertices will lie between vertices Pi and Qi .This can be accomplished by performing three reverse operations on the part of the array P1 to Pn. The operations are:

a) Reverse the part of array between P1 to Pj-1. b) Reverse the part of the array between Pj to Pn.

c) Reverse the whole array containing the vertices of the left polygon.

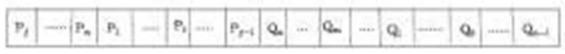

Fig. 6: Array after performing reversal operations on the polygon P and polygon Q

Similarly we will be modifying the right polygon such that the vertices from Qa to Qb will appear at the starting of the part of array containing the vertices of right convex polygon in clockwise direction. Rest of the vertices will lie after Qb.This can be accomplished by performing three reverse operations on the part of the array Q1 to Qm.The operations are:a) Reverse the part of array Q1 to Qa-1.

b) Reverse the part of the array between Qa to Qm.

c) Reverse the whole array containing the vertices of the right polygon.

Now we need to arrange the vertices such that the starting vertices of the array will represent the vertices of the merged convex polygon in clockwise order. For that we need to shift the vertices from Qa to Qb after the vertex Pi in the array.For accomplishing this we require three reverse operation as Follows:

a) Reverse the part of the array from Qa to Qb.

b) Reverse the part of the array from Pi+1 to Pj-1. c) Reverse the part of array from Pj-1 to Qa.

The final results would be as follows:

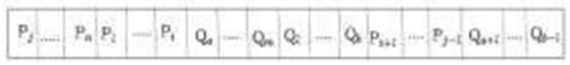

Fig. 7: Array after merging polygons P and Q

The part of the array from Pj to Qb represents the vertices of the merged convex polygon in clockwise direction.

5. COMPLEXITY ANALYSIS

5.1. Time complexity Analysis

5.1.1. Time Complexity of Convex Hull

Algorithm

In each step of the recursion we are splitting the array into two sub arrays of the equal size and recursively calling Hull module for the two sub arrays. Once we get the convex hull of the two sub arrays we merge them so as to form the convex hull of the entire array.The process of merging of two convex polygons takes linear time and constant space (See the time complexity analysis part of the algorithm of merging of two convex polygons) . Hence at each level we are subdividing the array and then merging it. This whole process requires linear time and constant space. The time complexity of the algorithm is given by the expression:

$$T(n) = 2T(n/2) + O(n)$$

which on solving using Master's theorem gives

$$T(n) = O(nlogn)$$

The time Complexity for the pre-processing step (sorting the point set on the basis of their x co- ordinate) takes O(nlogn) time. Hence the time complexity of the algorithm is O(nlogn).

5.1.2. Time Complexity of Merging of Convex

Polygon

Assume 'n' be the number of points in the left polygon and 'm' be the number of ppints on the right polygon.

1) To compute the common upper tangent of the convex polygons, the step 1 takes O(n+m) time to find out the left most point of the right polygon and right most point of the left polygon.

2) Step 2 and step 3 require O(n+m) time. Similarly to compute the lower common tangent the time complexity is O(n+m).

3)In order to obtain the required result we need to perform the three shifting operations each one requiring three reverse operations. The total time taken by this process is O(n+m).

The total time complexity of the algorithm is

O(n+m).

5.2. SPACE COMPLEXITY ANALYSIS

The entire algorithm doesn't require any extra space as the splitting and merging is done in-place. Hence the space complexity of the algorithm is O(1).

The merging step of Convex Hull Algorithm i.e. merging of two convex polygon takes constant amount of space. All the reversing operations needed to obtain the final result are being performed on the input array thereby requiring no extra space. Hence, the space complexity of the algorithm is O(1).

6. ACKNOWLEDGEMENT

We would like to express our gratitude to all the faculty members of Computer Science and Engineering Department, National Institute of Technology Durgapur for the help, support and guidance. It would not have been possible for us to complete the work without their constant support.

REFERENCES

[1] AURENHAMMER, F.1191. Voronoi diagrams, a survey of a fundamental geometric data structure. ACM Comput. Surv. 23, 345-405.

[2] BOARDMAN, J. 1993.Automating spectral unmixing of AVIRIS data using convex geometry concepts. In the 4th JPL Airborne Geoscience Workshop (Washington, D.C.). JPL, Pasadena, Calif.

[3] WEEKS, J. 1991. Convex hulls and isometrics of cusped hyperbolic 3-manifolds. Tech. Rep. TR GCG32, The geometry Center. Univ. of Minnesota, Minneapolis, Minn. Aug.

[4] T.Cormen, C.Leiserson, R.Rivest, and C.Stein. Introduction to Algorithms. 2nd edition, MIT Press, 2001.

[5] D.T.LEE. On finding the convex hull of a simple polygon. International Journal of Computing & Information Sciences 12(2):87-98, 1983.

[6] T.Chan Optimal output-sensitive convex hull algorithms in two and three dimensions Discrete Comput. Geom. !6 (1996),pp. 36-368

[7] A.M.Andrew Another efficient algorithm for convex hulls in two dimensions Inform. Process. Lett., 9(1979), pp.216-219

[8] Computational Geometry: Algorithms and Applications 3rd Edition by Mark de Berg (Author), Otfried Cheong (Author), Marc van Kreveld (Author), Mark Overmars (Author).