An Associative Static and Dynamic Convex Hull Algorithm

Maher M. Atwah
Computer Science Department
Hiram College
Hiram, OH 44234
atwahmm@hiram.edu

Johnnie W. Baker
Computer Science Department
Kent State University
Kent, OH 44242
jbaker@mcs.kent.edu

Abstract

This paper presents a new static and dynamic recursive parallel algorithm for the convex hull problem. This algorithm is a parallel adaptation of the Graham Scan and Quick Hull algorithms. The computational model selected for this algorithm is the associative computing model (ASC) which supports massive parallelism through the use of data parallelism and constant time associative search and maximum functions. Also, ASC can be supported on existing SIMD computers. The static algorithm requires $O(n)$ space, $O(\log n)$ average case running time, and $O(n)$ worst case running time. If $O(\log n)$ ISs are used the, static algorithm should have an average running time of $O(\log \log n)$.

1. Introduction

The convex hull of a finite set of a set S of n planar points is an important geometric concept. It can be defined as the smallest convex polygon for which each point in S is either on the boundary of the convex polygon or in its interior. We assume that no two points in S have the same x or y coordinates and that no three points in S lie on the same straight line as these assumptions make the algorithm easier to describe. However, the algorithm given in this paper can be easily modified to eliminate the necessity of these assumptions. The convex hull plays a central role in the field of computational geometry. This geometric concept finds practical applications in many areas including pattern recognition, image processing, engineering, computer graphics, design automation, and operations research.

Section 2 describes the associative model. Section 3 gives an associative parallel adaptation of the Graham Scan and Quick Hull algorithms. In Section 4, an associative dynamic parallel adaptation of the QuickGraham algorithm is presented.

2 The Associative Model of Computation

The associative computing model (ASC) is an extension of the general associative processing techniques developed for the associative STARAN SIMD computer in the 1970's for massively parallel computation. As our algorithm will demonstrate, ASC provides an efficient computational model for algorithms requiring massive parallelism. Details of how this model can be implemented on certain SIMD computers are given in [9]. Also, a high level language based on ASC detailed in [9] has been installed on the STARAN, ASPRO, WaveTracer, and Connection Machine CM-2.

A brief summary of the features of the ASC model is presented here. Additional information and properties of this model may be found in [8, 3]. ASC consists of an array of cells, each containing a processor and its local memory. Cell memory holds variables used for data-parallel operations. These cells are connected by bus to the instruction stream (IS) which stores a copy of the program being executed and broadcasts program instructions to all active cells. The general ASC model described in [8] allows multiple instruction streams (MASC). It is convenient to assume that variables and constants that need to be globally available to all cells are stored in the memory of the IS and may be broadcast to all active cells. The IS also has the ability to read and store a value from a specific cell. The IS variables are called sequential or global variables and PE or cell variables are called parallel variables. In addition to data-parallel execution, the ASC model supports constant time functions for associative searching and selection, logical operations, and maximum and minimum. Constant time searching permits the simultaneous examination of all active cells and the identification of all those that meet the search criteria. These identified cells are called responders and become the new set of active cells. By altering the criteria, different cells become responders. The IS has the ability to detect the presence of responders in unit time. It is also possible to access each active cell sequentially and

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

to return to the set of cells which were active preceding the search or to activate all cells. The maximum or minimum value of a parallel variable (or the cell address containing that value) can be computed for all active cells in constant time. The cells may be connected by means of an interconnection network.

3 Static QuickGraham Algorithm

An associative version of Graham Scan [5] and QuickHull [10] algorithm is presented next. One of the goals for Graham's algorithm [5] set forth by Preparta and Shamos [10] is to have an algorithm that runs on a parallel environment that allows the data to be split, preferably recursively, into smaller subproblems. This algorithm attempts to achieve this goal. This algorithm was inspired by the Quickhull algorithm [10] and by our Associative Graham Scan [1] and Associative QuickHull [2] algorithms. The algorithm is recursive and uses a divide-and-conquer approach. The original problem is divided into two subproblems, each subproblem is solved recursively, and the solutions are merged to produce the overall solution.

Given a set of S points, a partition is formed by the line segment $\overline{we}$, where w is the leftmost point in the set (i.e. one with the minimum x-coordinate) and e is the rightmost point (i.e. one with the maximum x-coordinate). The set P_A consists of the planar points that are on or above $\overline{we}$ while the set P_B consists of the points below $\overline{we}$. The algorithm finds the convex hull $CH(P_A)$ of points in P_A, the convex hull $CH(P_B)$ of points in P_B, and concatenates the two solutions. Since the algorithm to determine $CH(P_A)$ is similar to the one for $CH(P_B)$, we discuss only the first.

To determine the $CH(P_A)$, first all points that are on or below $\overline{we}$ are deleted. Then a point $r \in P_A$ is found such that r has the maximum y-coordinate among all the points p with $p \in P_A$. The usual assumption that no two points have the same x or y coordinates, can be dropped in this algorithm by selecting the point whose x-coordinate is maximal. Point r is a candidate to be a vertex of $CH(P_A)$. While it is likely that r is a convex hull vertex, it will be eliminated at a later stage if not. The technique used to eliminate r if it isn't in $CH(P_A)$ is the Graham Scan technique for eliminating on of three points, which was also used in the Associative Graham Scan algorithm [1]. The next step is to divide the problem of finding $CH(P_A)$ into the two following subproblems: (1) Delete all points that are on or below $\overline{re}$ and find $CH(P_R)$ of P_R, where P_R is the set of points in P_A above $\overline{re}$, and (2) Delete all points that are on or below $\overline{wr}$ and find $CH(P_L)$ of P_L, where P_L is the set of points above $\overline{wr}$. The solutions to the subproblems of finding $CH(P_R)$ and $CH(P_L)$ are found and joined to produce $CH(P_A)$. This is a new sequential algorithm for the convex hull. While not all details are specified in the proceeding discussion, the remaining details can be deduced from the parallel version of this algorithm given in Figure1.

The QuickGraham technique replaces the problem of computing $CH(S)$ with the problem of computing $CH(S_1)$ and $CH(S_2)$ where S_1 and S_2 are disjoint subsets of S. Moreover, two sequences of convex hull points given by $CH(S_1)$ and $CH(S_2)$ can be joined to produce the sequence of convex hull points required for $CH(S)$. Now if each of S_1 and S_2 has cardinality at most equal to a constant fraction of the cardinality of S and this holds at each level of recursion then using the same analysis as QuickHull, it can be shown that QuickGraham has $O(n \log n)$ average case running time. However, like the Jarvis March and QuickHull, the worst-case situation results in $O(n^2)$ time complexity. The QuickGraham Algorithm offers the feature of being conducive to a parallel adaptation due to its divide-and-conquer nature, but unfortunately does not use balancing division of a problem into equal-size subproblems so that worst-case complexity of $O(n^2)$ can arise.

A recursive parallel adaptation of the sequential QuickGraham is presented in Figure 1. Let S be a set of n planar points that are stored in the local memory of the PEs with at most one point per PE. Each point has its two coordinates stored in the PE variables x and y. Also, each PE has a variable called $delete$. The rest of the variables are those of the IS. These include an edge list Q which is used to store either the left or right point of the potential edge and the endpoints of potential edges of the convex hull as they are located. The list is maintained as a queue and is read from the top while new items are inserted at the bottom of the list.

Let e be the extreme point of S with the largest x coordinate. Also, let w be the extreme point of S with the smallest x coordinate. The first part of the algorithm finds all the convex hull points above $\overline{we}$ from w to e. The second part finds the convex hull points below $\overline{we}$ from w to e. Since the second part can be accomplished by a simple modification to the first part, it is omitted.

If every point selected by each pass through the WHILE loop were a point on the convex hull, an identical analysis to the Associative QuickHull could be used. While this point may fail to be in the convex hull, it is fairly unlikely this will happen very often. Therefore, for the average case, it is expected that this algorithm will have $O(\log n)$ running time and $O(n \log n)$ cost. For the same reason given for the Associative QuickHull, its worst case running time is $O(n)$ and cost is $O(n^2)$.

Since this algorithm is recursive, it can also be solved using multiple ISs. As with the Associative QuickHull, it is expected that if $O(\log n)$ ISs are a available then this algorithm should have an average running time of $O(\log \log n)$ and an average cost of $O(n \log \log n)$. In the worst case, its running time is $O(n)$ and its cost is $O(n^2)$.

Associative QuickGraham Algorithm

Input: A set S, of points given as (x, y) coordinates.
output: A linked list L, of the vertices of the convex hull.

1. All PEs are used to compute max and min, the maximum and minimum value of the x-coordinate of S , respectively.
2. Restrict the PEs to the one that satisfies $x = xmin$. This PE stores w.
3. Restrict the PEs to the one that satisfies $x = xmax$. This PE stores e.
4. The IS places w and $\overline{we}$ into FIFO list Q.
5. The IS adds w and e to L and a forward pointer from w to e.
6. While Q contains unprocessed edges
   (a) Read l and $\overline{mq}$ from top of list Q and mark it "processed"
   (b) Restrict the active PEs to those whose point p has it's x-coordinate is between the x-coordinates of m and q. Each active PE sets $delete = true$ if p lies below $\overline{mq}$.
   (c) Restrict the active PEs to those above $\overline{mq}$. If no active PEs remain, skip steps 6.d - 6.f.
   (d) All active PEs are used to locate the point p in the active PEs whose y-coordinate has the maximum value. This PE stores r. If r is already in L, skip steps 6.d - 6.f.
   (e) The IS places three edges and three points at bottom of Q
      i. q and $\overline{mr}$
      ii. m and $\overline{rq}$
      iii. If $l \neq m$ and $l \neq q$ then if $l.x < r.x$ then place m and $\overline{lr}$ else place q and $\overline{rl}$.
   (f) The IS perform the following:
      i. Deletes the forward pointer from m to q.
      ii. Adds r to L
      iii. Adds a forward pointer from m to r and another forward pointer from r to q.
7. Activate all PEs that contain $delete = false$. These PEs hold the vertices of the convex hull.

Figure 1. Associative QuickGraham

4 Dynamic QuickGraham Algorithm

In this section we introduce a dynamic convex hull algorithm. This algorithm is based on parallel adaptation of the static QuickGraham algorithm (Figure 1).

This problem can be stated as follows: Given a number of points that are moving in Euclidean space, we want to maintain for this set of points the convex hull. Each of the convex hull algorithms we have examined thus requires all of the data points to be present before any processing begins. In many geometric applications, particularly those that run in real-time, this condition cannot be met and some computation must be done as the points are being received. In other words, we call an algorithm that cannot look ahead at its input off-line, while one that operates on all the data collectively is called on-line.

A dynamic convex hull is needed [12], when a population is to be estimated using statistics [6, 7], or simulating chemical reactions. In addition, a dynamic algorithm is needed in applications such as graphics, air traffic control, and robotics.

To obtain dynamic algorithm for convex hull, we must make substantial modification to the static algorithms presented [2]. We first state the requirements for a dynamic algorithm. Following Chazelle [4], we need to support four operations:

1. Insert a point M.
2. Delete a point M.
3. Report all the vertices of the convex hull in some reasonable order.
4. Determine whether an arbitrary point M lies inside or outside the convex hull.

Note that in operation 1 the point M can be either a new point or a point that is already in the structure. Also, the operation "delete point M" always refers to a vertex of the convex hull.

If we do not need to support deletions, it is quite easy to make our algorithms dynamic since we can continue to discard points which are not extreme points of the hull. However, if points are to be deleted, it is possible that some non-hull points will later become extreme points of the hull, and in this case we can no longer eliminate any points entirely. With a dynamic algorithm, the number of processors needed is determined by the maximum size of P, where P is the number of processors available. In the discussion that follows, we use N to indicate the largest number of points that will be in P at any given time. Since our algorithm is using one point per processor N will be equal to P.

This algorithm considers only the upper convex hull; the lower hull is the same as the upper hull with minor modifications. We have to note that when a point is marked for

deletion its value is still stored in that PE and it can be retrieved by marking that PE not deleted.

Let S be the current set of n planar point that are stored in the local memory of the PEs with at most one point per PE. Each point p has its two coordinates x and y stored in the PE variables. Let e be the extreme point of S with the largest x coordinate. Also, let w be the extreme point of S with the smallest x coordinate.

A minimum of three points is needed for the algorithm to work. As the first point or points initially are entered they are stored one per processor in the array, the static algorithm given in [2] is used to compute the upper hull. When a point is deleted, its value is simply replaced by **null** in its own processor. New points being entered are assigned to a processor with a **null** value. As long as the total number of points does not exceed N, there will be no overflow.

After that, whenever a points M is inserted into P the following steps takes place:

1. If M is below $\overline{we}$ then mark it for deletion.
2. If there is a point p equal to M then replace M by **null**.
3. If step 1 & 2 fails then
   (a) Find the greatest lower bound point (call it glb) and the lowest upper bound point (call it lub) from the set of the current convex hull points.
   (b) If M is below $\overline{glb, lub}$ then mark this point for deletion.
   (c) Else
      i. Activate all PE such that the x-coordinate is less than the x-coordinate of M.
      ii. For each active PE, assign $area$ = area of triangle wpM where p is point (x, y) held in that PE.
      iii. Restrict the active PEs to the one storing the maximum area. This PE is called L and marked $extreme$.
      iv. All the PEs that are below $\overline{LM}$ are marked for deletion.
      v. Activate all PE such that the x-coordinate is greater than the x-coordinate of M.
      vi. For each active PE, assign $area$ = area of triangle Mpe where p is point (x, y) held in that PE.
      vii. Restrict the active PEs to the one storing the maximum area. This PE is called G and marked $extreme$.
      viii. All the PEs that are below $\overline{MG}$ are marked for deletion.

Note that the greatest lower bound point for a point M is a convex hull point with its x coordinate value is smaller than the x coordinate of M but larger that all the x coordinate values of all the convex hull points that are to the left of M see Figure 2). Also, the lowest upper bound point for a point M is a convex hull point with its x coordinate value is larger than the x coordinate of M but smaller that all the x coordinate values of all the convex hull points that are to the right of M see Figure 2).

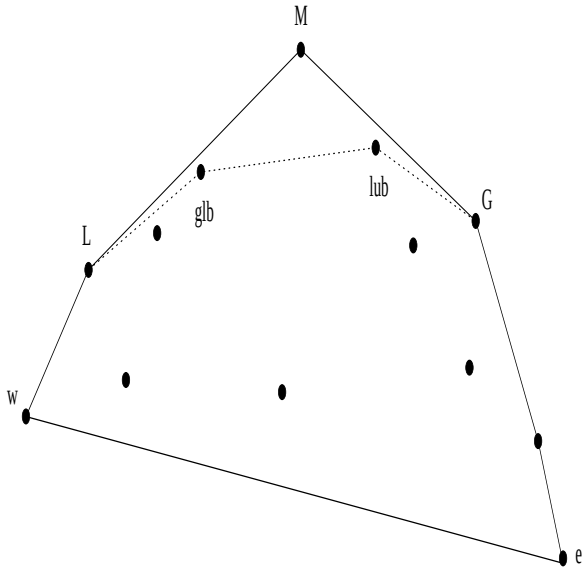

Figure 2. Dynamic Convex Hull: After inserting a point M.

Figure 2 gives an example of inserting a point M. The dotted lines represent the old part of the convex hull and the solid lines represent the new hull after M has been inserted. Notice that glb and lub points are marked for deletion since they fall below $\overline{LM}$ and $\overline{MG}$, respectively.

All the above steps require constant time. So, inserting a point to the upper convex hull cost $O(1)$ time.

Point queries are very simple with this scheme and it is performed as follows:

1. If M is below $\overline{we}$ then it is outside the upper hull
2. Else
   (a) Find the greatest lower bound point (call it glb) and the lowest upper bound point (call it lub) from the set of the current convex hull points.
   (b) If M is below $\overline{glb, lub}$ then it is inside the upper convex hull, otherwise it is outside the upper convex hull.

As point insertion, point query cost $O(1)$.

Deleting a point is different than inserting a point since, if points are deleted, it is possible that some non-hull points will later become extreme points of the hull. So, if there is a request to delete a point M from S the following steps take place (M is a vertex of the convex hull):

1. Replace M by **null**.
2. Find the greatest lower bound point (call it glb) from the set of the current convex hull points.
3. Activate all the PE such that the x-coordinate is greater than the x-coordinate of glb.
4. If any one of these PEs was marked for deletion then mark it not deleted.
5. Run the static algorithm Figure 1 on the set of active PEs to recompute the upper hull.

All the steps cost $O(1)$ except step 5, which in the worst case will cost $O(h)$, where h is the current number of the vertices of the convex hull.

Reporting all the vertices of the convex hull in clockwise (or counterclockwise) order cost $O(h)$, where h is the current number of the vertices of the convex hull. Point reporting has the same cost as sorting. First, we activate all the PEs that are marked $extreme$. Then, we pick the point that contain the smallest x-coordinate and so on until all the extreme points are reported.

This algorithm cannot freely mix insertions and deletions with queries and reports. Queries and report requests cannot be entered after a series of insertions and deletions until the new hull has been completely calculated, and all queries and reports must be completed before a batch of insertions and deletions can be entered.

5 Conclusion

A parallel static and dynamic convex hull algorithm designed for the associative computing model is presented in this paper. This static algorithm is similar to the QuickHull algorithm except or algorithm is simpler and doesn't require a complex and expensive merge step. In the dynamic algorithm, point insertion and query takes $O(1)$ time and point deletion and reporting take $O(h)$ time, where h is the number of the vertices of the convex hull. In the worst case, h equals n and time reporting and deletion is $O(n)$. One advantage of this algorithm is that it did not require the use of network operations, which are known to be much slower than local operations [11].

References

[1] M.M. Atwah, J.W. Baker, and S.G. Akl. An Associative Implementation of Grahams Convex Hull Algorithm. *Proc. of the Seventh IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 273-276, Washington D.C., October 1995.

[2] M.M. Atwah, J.W. Baker, and S.G. Akl. An Associative Implementation of Classical Convex Hull Algorithms. *Proceedings of Eighth IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 435-438, Chicago, IL, October 1996.

[3] J. Baker and M. Jin. Simulation of Enhanced Meshes with MASC, a MSIMD Model. *Proc. of the 11th International Conference on Parallel and Distributed Computing Systems*, pages 511-516, November 1999.

[4] B. Chazelle, Computational Geometry on a Systolic Chip, *IEEE Trans. Comp.*, C-33(9), 1984, 774-785.

[5] R.L. Graham. An Efficient Algorithm for determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, vol. 1, pages 73-82, 1972.

[6] R. A. Jarvis, On the Identification of the Convex Hull of a Finite Set of Points in the Plane, *Information Processing Letters*, 2, 1973, 18-21.

[7] M. H. Overmars and J. Van Leeuwen, Dynamically maintaining configurations in the plane, *Proc. 12th Annual SIGACT Symp.*, Los Angeles, CA. May 1980.

[8] J. Potter, J. Baker, A. Bansal, S. Scott, C. Leangsuksun and C. Asthagiri. ASC: An associative computing paradigm. *IEEE Computers*, Vol. 27(11), pp 19-25, November 1994.

[9] J. Potter. *Associative Computing: A Programming Paradigm for Massively Parallel Computers.* Plenum Publishing, New York. 1992.

[10] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag. Second Edition. New York. 1985.

[11] J. H. Reif, Editor, Asynchronous PRAM Algorithms, *Synthesis of Parallel Algorithms*, 22, 957-997, Morgan Kaufman Publishing, San Mateo, CA 1993.

[12] M. I. Shamos, Computational Geometry, Ph.D. dissertation, Yale University, 1978.