SOME PERFORMANCE TESTS OF CONVEX HULL ALGORITHMS

D.C.S. Allison
M.T. Noga

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

CS830001

Abstract: The two-dimensional convex hull algorithms of Graham, Jarvis, Eddy, and Akl and Toussaint are tested on four different planar point distributions. Some modifications are discussed for both the Graham and Jarvis algorithms. Timings taken of FORTRAN implementations indicate that the Eddy and Akl-Toussaint algorithms are superior on uniform distributions of points in the plane. The Graham algorithm outperformed the others on those distributions where most of the points were on or near the boundary of the hull.

Key Words: Convex hull, computational geometry, sorting, distributive partitioning.

1. Introduction

No problem in the field of computational geometry has received more attention during the last few years than the computation of the *convex hull*. This problem may be stated as follows: Given a set S of N points in two dimensional Euclidean space, determine those points in S which are the vertices of the minimum-area convex polygon that will entirely contain S. There are a number of important applications where it is necessary to compute the convex hull, for example in character recognition [22], statistics [17], and computer graphics [28]. Research has been directed towards producing algorithms which have good expected case behavior on several of the standard distributions of points in the plane. Among these are the Graham algorithm [13], which requires an explicit sort step, the gift-wrapping approach of Jarvis [16] (see also [10]), and the divide and conquer schemes of Eddy [11] and Akl and Toussaint [3,4]. We will focus our attention on several of the recent modifications [2,6,30], including one of our own [24], which lead to faster and cleaner implementations of both the Graham and Jarvis algorithms. These will be tested against coded versions of the Eddy and Akl-Toussaint algorithms which were obtained from the authors. All algorithms compute the *ordered convex hull* of the set [25], the sequence of vertices in the order in which they appear along the boundary of the hull.

2. Graham Algorithm

Historically, the Graham algorithm [13] represents one of the first attempts to produce a computationally efficient solution to a geometric problem. Before giving Graham's algorithm we first examine an algorithm due to Sklansky [31] to determine the convex hull of a simple non-intersecting polygon P. This algorithm is the precursor of the Graham algorithm.

Sklansky's algorithm is based upon systematically removing all concave vertices of P. Starting at an arbitrary vertex point on the polygon, a counterclockwise scan is made in which each vertex point j is tested for concavity. Each test may be accomplished in constant time by checking if j is on the left-hand side of a directed line from its two neighboring points i and k; Fig. 1. If j is found to be concave (on the left-hand side) it is immediately deleted from P and the scan temporarily backtracks clockwise one point. (This is because removal of a point from P may make the previous vertex point concave.) The algorithm terminates when all points have been examined at least once, and the last point examined was convex. An easy induction argument suffices to prove that this algorithm requires O(N) time to complete.

Unfortunately, Sklansky's procedure as described above is incorrect (see [20]). There are some polygons on which it will fail to produce the correct answer. However, for many polygons it will work, and one such class is the *star-shaped polygons*. A polygon P is star-shaped if and only if there exists a point z, interior to P, such that for all vertices $v \in P$, the line zv is contained in P. That is, there is at least one point z inside P that can see all of the vertices of P.

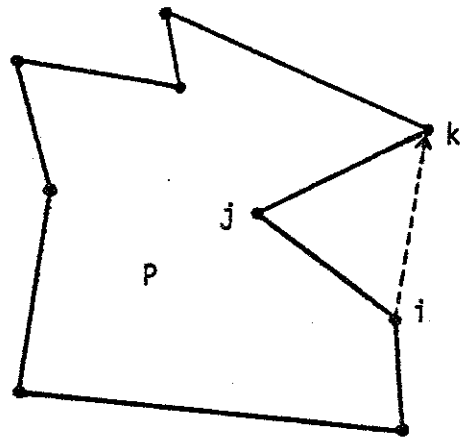

Fig. 1. A simple non-intersecting polygon P. Vertex j will be deleted since it is on the left-hand side of a directed line segment from i to k.

Relating this back to the convex hull of a set of points S, Graham simply ordered the points by polar angle about an interior point z of the hull. The reordered points of S implicitly define a star-shaped polygon and consequently the remainder of Graham's algorithm is nearly identical to Sklansky's. A minor difference is that as a result of the ordering step there may be subsequences of points that have the same polar angle (lie along the same ray). In this case only the outermost points, those with the greatest amplitude, are retained. We give a summary of the Graham algorithm and its worst-case analysis below.

Step 1: Convert the points of S to (r,θ) polar coordinates about any point z which is interior to the convex hull of S. Graham stated that z can be computed in at most O(N) time (but usually much less time) by testing 3 element subsets of S for collinearity and taking the centroid of the first triangle found.

Step 2: Order the points by increasing polar angle; Fig. 2. This requires O(NlogN) time in the worst-case [15].

Step 3: If it is the case that $\theta_i = \theta_{i+1}$, delete the point closest to z. Additionally, any point with an amplitude $r_i = 0$ can also be deleted. All of these points can be eliminated in O(N) time by using a linked-list data structure.

Step 4: The remaining points of S can now be considered the vertices of a star-shaped polygon. For each sequence of 3 consecutive vertices i, j, and k in S, j is

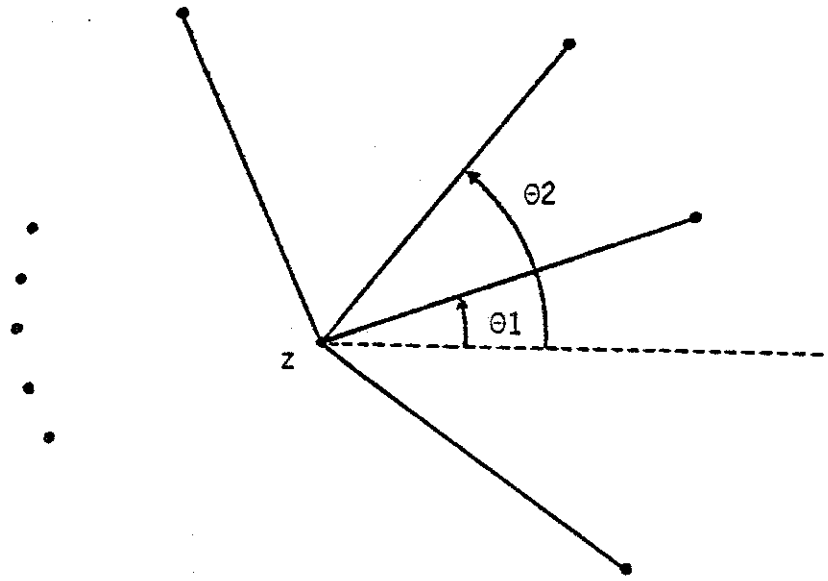

Fig. 2. Ordering the points by polar angle.

checked to see if it is concave with respect to points i and k. j is deleted from S whenever it is concave resulting in a new sequence of points which in turn must be checked for concavity. As explained earlier this step may be carried out in O(N) time.

The worst-case running time of the algorithm is clearly O(NlogN).

3. Implementation of the Graham Algorithm

To implement the Graham convex hull algorithm one might be tempted to use trigonometric functions in converting S to (r,θ) coordinates. The problem is that on modern computers trigonometric functions are very time consuming operations. Typically, for every sine function call, several floating point multiplications can be carried out in the same amount of time.

A way to avoid using trigonometric functions is to set z equal to the bottommost point of the set [6]. The bottommost point is the point with minimum y-coordinate. (If there is more than one such point, the minimum x-coordinate point is chosen.) The points may then be ordered by slope [24] as:

if $(x_i \neq x_z)$ or $(y_i \neq y_z)$

then $\theta_i := (-x_i - x_z)/(|x_i - x_z| + (y_i - y_z))$ (1)

else $\theta_i := -2$

Each θ_i will be in the range from $-1 \leq \theta_i < 1$ (-1 for the smallest angles) except when i has the same coordinates as z. In this case we set θ_i to a value smaller than -1 (a convenient choice is -2). The compu-

tation for each point takes at most three comparisons (1 for the absolute value), two subtractions, and one division. An extra array of size N is needed to store the angular values.

The ordering process should be carried out by use of a pointer sort [28]. In this way we avoid an exchange in each of the arrays representing x, y, and θ. Any one of the distributive sorting algorithms [5,19,21,23] can easily be modified for pointer sorting. This will insure that the Graham algorithm will have O(N) or near O(N) expected case time complexity over a wide range of planar point distributions [9,24]. The amount of extra array storage required to implement the sort described in [5] is approximately $2\frac{1}{2}N$.

Once the points are ordered, a circular doubly linked-list can be formed using the pointers from the sort step. The list pointers require 2N array locations. By making one pass through the entire list, all points coincident with the bottommost point and innermost points along identical rays can then be eliminated.

The linked-list is also available to delete points, if necessary, in the final step of the algorithm. Each of the three point concavity tests may be carried out by using the vector scalar product [6,8,31] as follows:

$$\textit{if } (x_i - x_j) \cdot (y_k - y_i) > (y_j - y_i) \cdot (x_k - x_i), \qquad (2)$$

then keep j in the convex hull, else delete j from any further consideration as a possible vertex point of the hull. No trigonometric functions are required, only 2 multiplications, 4 subtractions, and 1 comparison.

At the completion of the algorithm the linked-list will contain the convex hull in standard form. (A polygon P is in standard form if its

vertices occur in counterclockwise order beginning with the vertex that has least y-coordinate. All vertices must be distinct with no three consecutive vertices collinear [30].) This canonical representation is designed to simplify the implementation of other geometric algorithms which require convex hull preprocessing. Not including the sort step, a total of 5N array locations is required to implement the algorithm (including 2N to hold the x,y coordinates of the set S).

4. The Jarvis Algorithm

Imagine that each point of a planar set is represented by a small peg which has been inserted into a flat piece of wood. The convex hull of the set may be identified by attaching a long section of string to the bottommost peg and then subsequently wrapping it around in a counterclockwise fashion taut against the pegs until they have all been encircled (lasso style). Each peg where the string changes direction is a vertex point of the hull.

For computational purposes, this gift-wrapping approach was first formulated in 2-dimensions by R.A. Jarvis [16]. Later Akl [2] reported on two problems that may arise if the original algorithm is applied. These involved a bad choice of the initial origin and the necessity of including an intermediate step which eliminates points on the interior of the hull as the algorithm progresses. The algorithm below includes the modifications suggested by Akl to remedy these problems.

Step 1: Determine the bottommost point of the set and use this point as the initial origin (as suggested in [2]).

Step 2: A radius arm extended parallel to the x-axis is affixed to the initial origin and swung in a counterclockwise direction until another point of the set is encountered; Fig. 3. This point must be on the convex hull.

Step 3: The last point found may then be used as the origin for a new scan of the radius arm whereby the next point is identified.

Step 4: Any point found to lie in the region enclosed by a line from the initial origin to the last point found can be deleted since these points are interior to the hull (this step must be included [2]).

Step 5: Repeat steps 3 and 4 until the initial origin is reencountered.

The complexity of the algorithm can be anywhere from linear to quadratic (O(N) to $O(N^2)$) depending upon the number of points that are expected to be on the hull and the distribution of the points involved [16,24,30]. Concerning specific distributions, several results from stochastic geometry [26,27] can be used to analyze the average time used by the Jarvis algorithm. These are summarized in table 1 below:

Table 1 - Average time complexity of the Jarvis algorithm

Distribution	Average time complexity
Uniform in a convex polygon	O(NlogN)
Uniform in a circle	$O(N^{4/3})$
Normal in the plane	O(NlogN)
Uniform on the boundary of a circle	$O(N^2)$

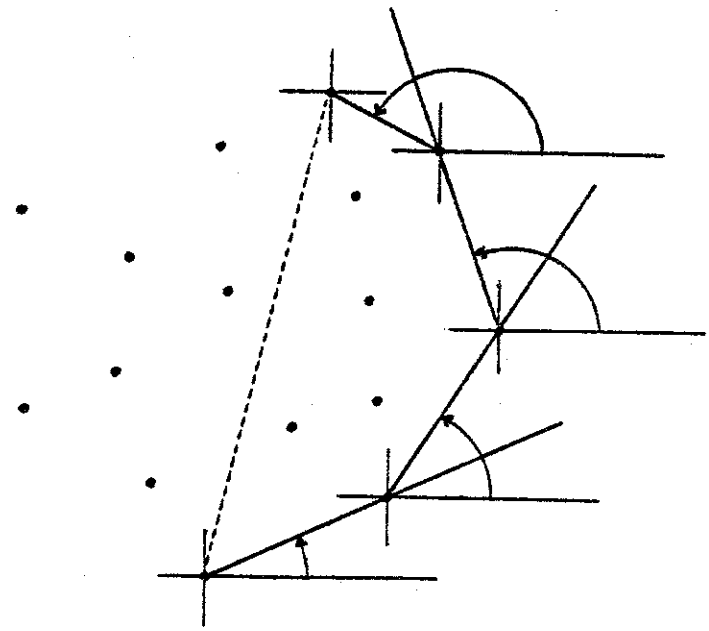

Fig. 3. The gift-wrapping approach.

5. Implementation of the Jarvis Algorithm

We have found that a few of the ideas used in the Graham algorithm may be usefully applied in the Jarvis implementation. Tenatively, we assume that all points in S are on the hull. As the algorithm progresses points will be deleted from S and, if the points are on the hull, their indices stored in an auxilary array H. At the completion of the algorithm, H will identify the ordered convex hull in standard form. The deletion of points from S may be handled by using a doubly linked list data structure (similar to the Graham implementation).

After initializing the linked-list the bottommost point z is located and placed into H; we may then delete z from any further consideration.

In anticipation of the point deletion phase, we evaluate and store the angular displacement of rays emanating from the first hull vertex z to the other points of S. If during the process of evaluating the angles a point coincident with z is found then it is deleted from the list of possible hull vertices; no evaluation of an angle is necessary. The angular value θ_i of each point may be computed as:

$$\theta_i = -(x_i - x_z)/(|x_i - x_z| + (y_i - y_z)). \qquad (3)$$

This formula is identical to the one used in the ordering step of the Graham algorithm (formula (1)).

Clearly, the point L which generates the minimum angle is the next vertex on the hull (for equal angles we pick the one furthest from the origin). This point may be placed into H and deleted from S. Any point with an angle identical to that of the last hull vertex L found may also be deleted.

The last point found L can be considered the origin of a coordinate system in which the remaining points of S fall into one of quadrants 1, 2, 3, or 4; Fig. 4. The point which makes with L the angle of minimum value is the next point on the hull. The computation may be carried out most efficiently by noting that quadrant 1 angles < quadrant 2 angles < quadrant 3 angles < quadrant 4 angles [16]. Assume inductively that as we work through the points of S, i indexes the point temporarily accepted as having the minimum angle, and that j indexes the point being considered as a replacement for i. Three cases arise when comparing i and j. (i) If j is in a higher numbered quadrant than i, then j may be immediately rejected as the 'next point' candidate; no evaluation of an angle is necessary. (ii) If i and j are in the same quadrant, then $angle_j$ is computed and compared with $angle_i$. In the case that $angle_i > angle_j$, we set i := j and $angle_i := angle_j$. (If $angle_i = angle_j$ take the point that is farthest from L.) (iii) If j is in a quadrant of lower value than i, we compute $angle_j$ and set i := j and $angle_i := angle_j$. Quadrant determination can be made by examining the sign of the quantities $(x_j - x_L)$ and $(y_j - y_L)$. Angular displacement may be determined by using variations of formula (1). For quadrants 1 and 3 we set

$$angle_j := -(x_j - x_L)/((x_j - x_L) + (y_j - y_L)),$$

and for quadrants 2 and 4 we set

$$angle_j := -(y_j - y_L)/((x_j - x_L) + (y_j - y_L)).$$

Any point found to be on or to the right side of a line from the initial origin (bottommost point) to the new hull vertex i may be deleted from the linked-list. Steps 3 and 4 are performed in an iterative fashion by

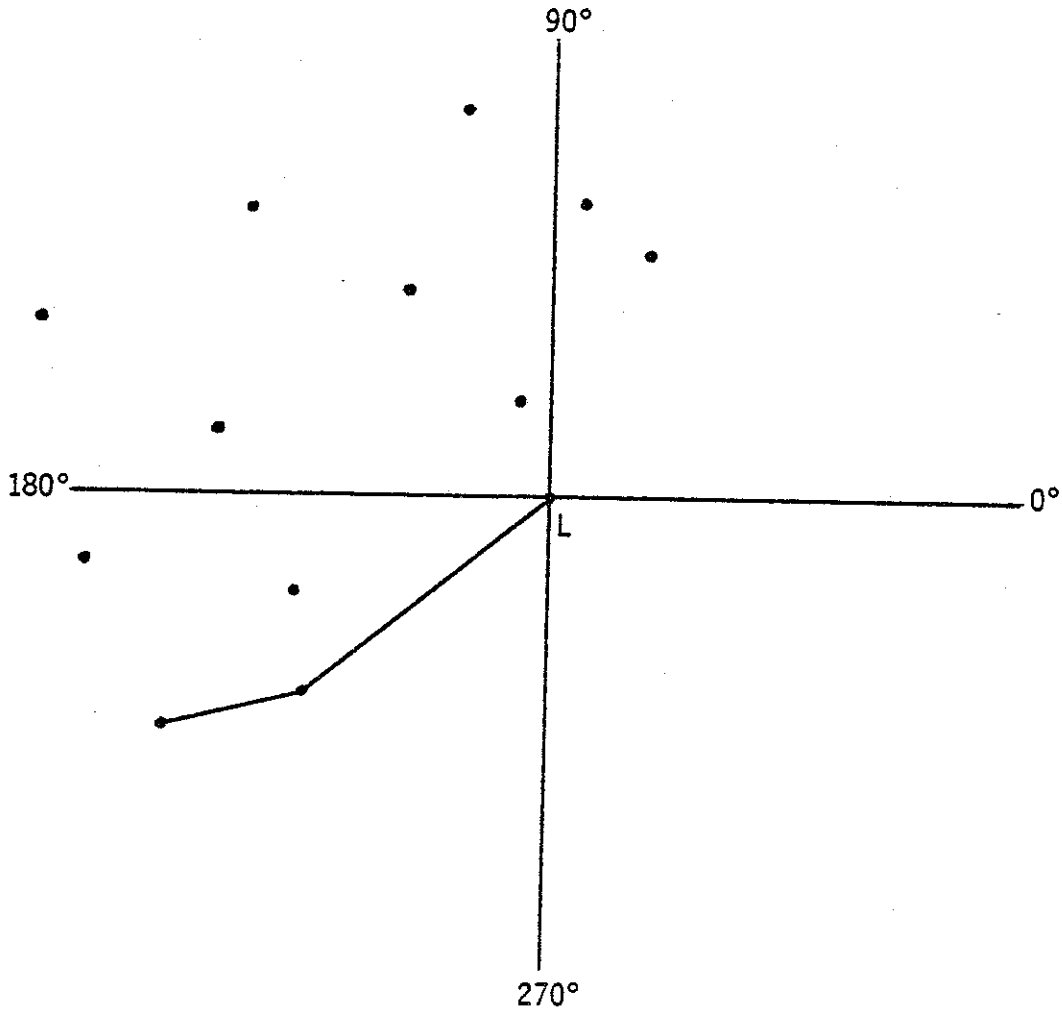

Fig. 4. Quadrant method for determining the next point on the hull.

setting L := i and repeating the procedure until the number of remaining points in the list is either one or zero. (If one point remains it must be on the hull and is added to H.)

Once the basic algorithm is complete the list pointers and array H can be used to reform a linked-list that contains the convex hull in standard from. The total storage required to implement the algorithm is 6N.

6. The Eddy Algorithm

One of the most useful tools in the design of algorithms is the divide and conquer technique. The basic idea is to split a problem of size N into M smaller subproblems all of approximately the same size. Each subproblem may then be further resplit (in a recursive fashion) until it is more profitable to apply a direct method to solve each subproblem. Often it is necessary to do further work building up the results of the subproblems to arrive at the total solution of the problem.

Several divide and conquer schemes have been devised for convex hull computation. The Eddy algorithm (W. F. Eddy [11]) is probably the best known of these schemes. The algorithm makes use of triangles composed of points already known to be on the hull (extreme points) to quickly eliminate those points interior to the boundary of the hull.

The first step is to locate two points that are certain to be on the hull. Practical choices are the bottommost and topmost points of the set S; Fig. 5. The points are then partitioned into two lists, one to the left of line BT, and the other to the right of BT. During the partitioning process we keep track of the highest points above and below

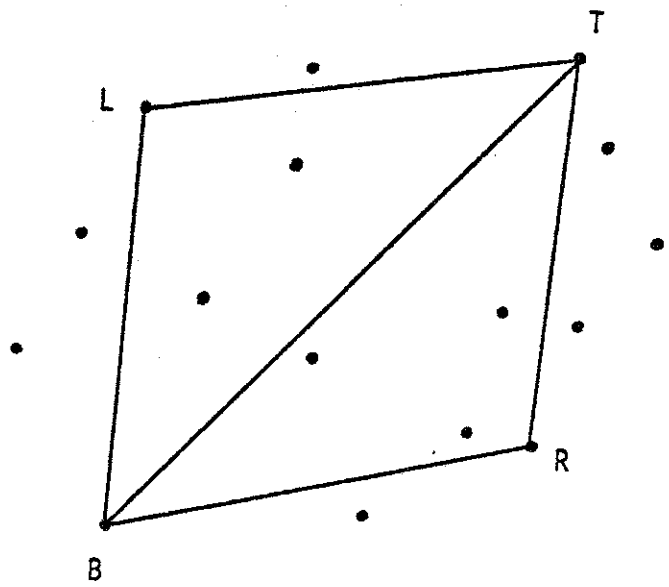

Fig. 5. Partitioning process of the Eddy algorithm.

line BT. These points, which are obviously on the hull, are labeled as L and R in Fig. 5. We now concentrate on the list to the left of line BT; the procedure will be identical for the list to the right of BT. Any points inside (or on the perimeter) of the triangle BLT are not on the hull and can be eliminated. The remaining points are placed into two lists, one above line BL, and the other above line LT. The triangle elimination step may then be carried out recursively on each of these new lists. The recursion bottoms out and backtracks whenever a sub-list of the original set of points is empty.

It may be noted that this algorithm is similar to the sorting algorithm Quicksort [28]. The key to the speed of Quicksort is the partitioning of an unsorted list into two sublists of approximately the same size such that all elements in the first list are smaller than those of the second list. This phase only requires a number of operations proportional to the size of the original list. It is also possible in the case of the Eddy algorithm to partition a list of N points in O(N) operations. The only numerical calculation required is the determination of whether a point is above, below, or on a given line.

Because the Eddy algorithm is operationally analagous to Quicksort, the worst case running time is $O(N^2)$. However, like Quicksort, an extremely pathological circumstance must present itself for this to occur [9]. Depending on the distribution of points, performance is most likely to be O(N) or O(NlogN). Table 2 gives the average time complexity for several specific distributions of points in the plane.

Table 2 - Average time complexity of the Eddy algorithm

Distribution	Average time complexity
Uniform in a convex polygon	O(N)
Uniform in a circle	O(N)
Normal in the plane	O(N)
Uniform on the boundary of a circle	O(NlogN)

Implementation details can be found in [12]. The computer program given there requires storage space of approximately 7N.

7. Akl-Toussaint Algorithm

Since the area of the triangles used to delete points in the Eddy algorithm shrinks quite rapidly, it is questionable whether it is necessary to carry out the recursion to such a deep level. S. G. Akl and G. T. Toussaint [3,4] have developed a convex hull algorithm similar to Eddy's where divide and conquer techniques are used for only the first few steps.

The first step involves finding the four extreme points xmin, ymin, xmax, and ymax. Any points which fall inside the quadrilateral region formed by these points may then be eliminated; Fig. 6.

Next, we find the extreme point k in each of the four extremal regions whose coordinates (x_k, y_k) maximize the quantity

$$m_1 x_k + m_2 y_k$$

where,

$m_1 = +1$ for regions 2 and 3,

$m_1 = -1$ for regions 1 and 4,

$m_2 = +1$ for regions 1 and 2,

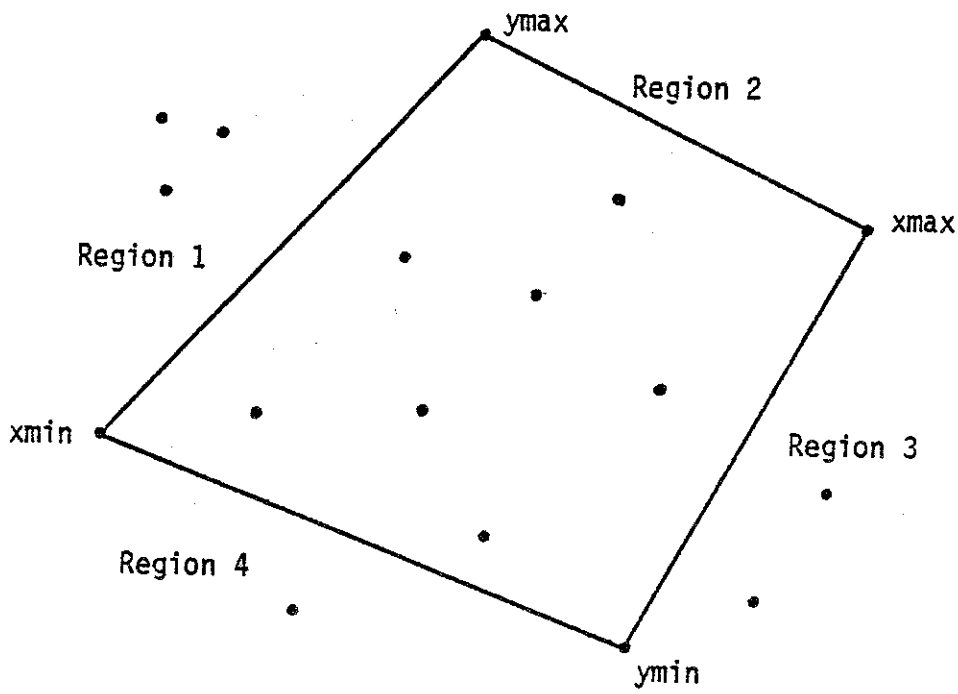

Fig. 6. Point deletion process of the Akl-Toussaint algorithm.

$m_2 = -1$ for regions 3 and 4.

This will allow all points falling inside each of the four triangles ijk where i,j ε {xmin,ymin,xmax,ymax} to be removed from any further consideration as possible extreme points of the hull.

For the final step, a variation of the Graham algorithm is applied to the remaining points in each region.

Akl and Toussaint [32] have shown that the average time complexity of their algorithm is largely dependent upon the percentage of points eliminated in the first step. For a uniform distribution of points inside a square, the time complexity will be O(N). Intuitively, the algorithm should be fast for any uniform distribution of points spread over a convex region. The worst-case time complexity is O(NlogN). This behavior results whenever all (or most) of the points are passed to the Graham algorithm in the final step.

Akl and Toussaint [1] use a liberal amount of storage in their implementation. Two arrays of size N hold the original coordinates. Twelve arrays of size N are used to store the coordinates and indices of points in the four extremal regions. Three arrays of length N are used to implement the Graham algorithm. Finally, one array of length N stores the final list of those indices (from the original coordinate arrays) which are on the hull.

The total array storage required is 18N.

8. Performance Evaluation and Discussion

In this section we report on a performance evaluation study between the Graham, Jarvis, Eddy, and Akl-Toussaint convex hull algorithms. All of the algorithms were coded in FORTRAN and run on an IBM 3032 (FORTX,OPT=2). The Graham and Jarvis algorithms were implemented using the ideas presented in sections 3 and 5. A coded version of the Eddy algorithm was obtained from Collected Algorithms from the Association of Computing Machinery [12]. As mentioned in the previous section, an implementation of the Akl-Toussaint algorithm was received for testing from the authors.

The following planar point distributions were chosen for testing:

(a) Uniform in a square,

(b) Uniform in a circle,

(c) Uniform in an annulus (inner radius 9/10 of outer radius),

(d) Uniform on the boundary of a circle.

Timings were recorded for sample sizes of 100, 250, 500, 1000, 2000, and 4000 points. 100 runs were made for sample sizes of 100, 250, 500, and 1000 points, 50 runs for 200 points, and 25 runs for 4000 points. The only exception was in the case of the Jarvis algorithm where only 1 run was made on distribution (d) for all sample sizes. On this distribution, performance was expected to be $O(N^2)$. In the tables which follow all times are for 100 runs and are given in seconds.

Table 3 - Uniform in a circle

N	Graham	Jarvis	Eddy	A-T
100	.44	1.00	.43	.41
250	1.22	3.31	1.02	.97
500	2.30	7.98	1.90	1.97
1000	4.80	19.36	3.80	3.96
2000	9.62	48.92	7.22	8.20
4000	19.96	95.84	14.44	17.24

Table 4 - Uniform in a square

N	Graham	Jarvis	Eddy	A-T
100	.47	.94	.43	.41
250	1.18	2.62	.97	.91
500	2.39	6.02	1.91	1.73
1000	4.78	13.21	3.65	3.37
2000	9.78	29.02	7.36	6.54
4000	19.96	60.60	14.48	12.88

Table 5 - Uniform in an annulus

N	Graham	Jarvis	Eddy	A-T
100	.45	1.93	.65	.77
250	1.17	6.65	1.51	2.05
500	2.24	16.37	2.84	4.35
1000	4.69	52.63	5.59	9.50
2000	9.44	104.86	10.82	20.74
4000	19.76	262.48	21.60	45.72

Table 6 - Uniform on the boundary of a circle

N	Graham	Jarvis	Eddy	A-T
100	.41	6.90	1.25	.88
250	1.01	42.50	3.54	2.30
500	2.01	164.00	7.80	4.82
1000	4.02	674.00	16.78	10.28
2000	8.16	2615.00	35.86	21.60
4000	17.28	9357.00	75.52	46.36

Examination of tables 3, 4, 5, and 6 indicates that the results of the performance testing are: (1) Contrary to previous reports [14,16], the Graham algorithm is competitive with several of the divide and conquer methods, and is to be strongly recommended on point distributions similar to (c) and (d), where most of the points are expected to be on or near the boundary of the hull. (2) The Jarvis algorithm is a poor performer relative to the other algorithms and cannot be recommended. (3) The Eddy and Akl-Toussaint algorithms were designed to be fast for uniform distributions of points in the plane. The data in tables 3 and 4 indicate that these algorithms perform exceptionally well for these distributions and are therefore highly recommended in these situations.

9. Conclusion

All convex hull algorithms are intricately linked to the problem of sorting. Finding the convex hull is really a two-dimensional sorting problem, because the task of any convex hull algorithm is firstly to discard those points that are not on the hull, and secondly to order the remaining points exactly as they appear in sequence along the boundary of the convex polygon. Among the algorithms we have examined, the Graham algorithm requires an explicit sort step, the Eddy algorithm is operationally similar to Quicksort, and the Akl-Toussaint algorithm is a combination of these two methods. Even the Jarvis algorithm is related to straight selection sorting.

It has been shown [33] that, for inputs of size N, finding the convex hull takes O(NlogN) comparisons in the worst-case. The proof is related to the one for sorting algorithms, which also requires at least

O(NlogN) comparisons in the worst case. The implication is clear. Convex hull algorithms are ultimately related to sorting algorithms and vice-versa. Until a faster sorting algorithm is developed, it is unlikely that any new convex hull algorithm will exhibit a dramatic increase in speed over any of its predecessors.

10. Acknowledgements

The authors would like to thank Godfried Toussaint and Selim Akl for words of encouragement, reference material, and computer programs.

REFERENCES

1. S.G. Akl, *Personal communication.*

2. S.G. Akl, *Two remarks on a convex hull algorithm,* Info. Proc. Lett. 8, no. 2 (1979), 107-108.

3. S.G. Akl and G.T. Toussaint, *Efficient convex hull algorithms for pattern recognition applications,* Proc. Fourth International Joint Conf. on Pattern Recognition, Kyoto, Japan (1978), 1-5.

4. S.G. Akl and G.T. Toussaint, *A fast convex hull algorithm,* Info. Proc. Lett. 7, no. 5 (1978), 219-222.

5. D.C.S. Allison and M.T. Noga, *Usort: an efficient hybrid of distributive partitioning sort,* B.I.T. 22, (1982), 136-139.

6. K.R. Anderson, *A reevaluation of an efficient algorithm for determining the convex hull of a finite planar set,* Info. Proc. Lett. 7, no. 1 (1978), 53-55.

7. J.L. Bentley and M.I. Shamos, *Divide and conquer for linear expected time,* Info. Proc. Lett. 7, no. 2 (1978), 87-91.

8. A. Bykat, *Convex hull of a finite set of points in two dimensions,* Info. Proc. Lett. 7, no. 6 (1978), 297-298.

9. L. Devroye and T. Klincsek, *Average time behavior of distributive sorting algorithms,* Computing 26, no. 1 (1981), 1-7.

10. D.R. Chand and S.S. Kapur, *An algorithm for convex polytopes,* JACM 17, (1970), 78-86.

11. W.F. Eddy, *A new convex hull algorithm for planar sets,* ACM Trans. on Math. Soft. 3, no. 4 (1977), 398-403.

12. W.F. Eddy, *Algorithm 523 CONVEX, a new convex hull algorithm for planar sets,* Collected Algorithms from ACM, (1977), 523P1-523P6.

13. R.L. Graham, *An efficient algorithm for determining the convex hull of a finite planar set,* Info. Proc. Lett 1, no. 1 (1972), 132-133.

14. P.J. Green and B.W. Silverman, *Constructing the convex hull of a set of points in the plane,* The Computer Journal 22, no. 3 (1978), 262-266.

15. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, (1978).

16. R.A. Jarvis, *On the identification of the convex hull of a finite set of points in the plane*, Info. Proc. Lett. 2, no. 1 (1973), 18-21.

17. M.G. Kendall, *Discrimination and classification*, Proc. International Symposium on Multivariate Analysis, Edited by P.R. Krishnaiah, Academic Press, New York, (1966).

18. J. Koplowitz and D. Jouppi, *A more efficient convex hull algorithm*, Info. Proc. Lett. 7, no. 1 (1978), 56-57.

19. J.S. Kowalik and Y.B. Yoo, *Implementing a distributive sort program*, Journal of Information and Optimization Sciences 2, no. 1 (1981), 28-33.

20. D. McCallum and D. Avis, *A linear algorithm for finding the convex hull of a simple polygon*, Info. Proc. Lett. 9, no. 5 (1979), 201-206.

21. H. Meijer and S.G. Akl, *The design and analysis of a new hybrid sorting algorithm*, Info. Proc. Lett. 10, no. 4-5 (1980), 213-218.

22. G. Nagy and N. Tuong, *Normalization techniques for handprinted numerals*, Comm. ACM 13, (1970), 475-481.

23. M. van der Nat, *A fast sorting algorithm, a hybrid of distributive and merge sorting*, Info. Proc. Lett. 10, no. 3 (1980), 163-167.

24. M.T. Noga, *Convex Hull Algorithms*, Masters Thesis, Dept. of Comp. Sci., Virginia Polytechnic Institute and State University, Blacksburg, VA, (1981).

25. F.P. Preparata and S.J. Hong, *Convex hulls of finite sets in two and three dimensions*, Comm. ACM 20, no. 2 (1977), 87-93.

26. H. Raynaud, *Sur l'enveloppe convexe des nuages des points aleatoires dans R!*, Appl. Prob. 7, 1970, 35-48.

27. A. Renyi and R. Rulanke, *Zufallige konvexe Polygone in einem Ringgebeit*, Z. Wahrscheinlichkeits 9, (1968), 146-157.

28. R. Sedgewick, *Implementing quicksort programs*, Comm. ACM 21, no. 10 (1978), 847-857.

29. M.I. Shamos, *Geometric Complexity*, Proc. Seventh Annual Symp. on Theory of Computing, (1975), 224-233.

30. M.I. Shamos, *Computational Geometry*, Ph.D. Thesis, Dept. of Comp. Sci., Yale Univ., New Haven, CT, (1978).

31. J. Sklansky, *Measuring concavity on a rectangular mosaic*, IEEE Trans. on Computers C-21, no. 12 (1972), 1355-1362.

32. G.T. Toussaint, S.G. Akl, and L.P. Devroye, *Efficient convex hull algorithms for points in two and more dimensions*, Tech. Rep. No. SOCS 78.5, School of Comp. Sci., McGill Univ., (1978).

33. A.C. Yao, *A lower bound to finding convex hulls*, JACM 28, no. 4 (1981), 780-787.