An optimal parallel algorithm for determining the intersection type of two star-shaped polygons

Subir Kumar Ghosh* Anil Maheshwari†

1 Introduction

A frequently occurring problem in computation geometry is to determine whether two geometric objects intersect. If they intersect, the problem is to compute their common parts. The intersection problem may be to compute the intersection or simply to detect an intersection between two objects. If the given objects are simple polygons of total n vertices, it is possible to detect the boundary intersection in $O(logn)$ time using $O(n)$ processors in the CREW PRAM computation model [2]. If the given objects are convex polygons, the common intersection region can be computed in $O(logn)$ time using $O(n/logn)$ processors in the EREW PRAM computation model [3].

In this paper, we consider the following problem. Given two star-shaped polygons with their respective star points, determine whether the boundaries of two star-shaped polygons intersect, or one contains the other, or they are disjoint. We present an optimal parallel algorithm for this problem that runs in $O(logn)$ time using $O(n/logn)$ processors in the EREW PRAM computation model, where n is the total number of vertices of the polygons.

One of the steps of the algorithm is to compute the visibility polygon of a star-shaped polygon from a point. The best known algorithm for this problem is given by Atallah and Chen [1] and it is optimal. By exploiting the properties of star-shaped polygons, we design a simple optimal parallel algorithm for this problem (in Section 3).

Two points a and b are said to be ***internally visible*** if both a and b belong to polygon A and the line segment joining a and b lies inside A. If a is an exterior point of a polygon A and b is a boundary point of A, then a and b are said to be ***externally visible*** if the line segment joining a and b lies in the exterior A. If the line segment joining two points touches the boundary of A, they are still considered to be visible. The ***visibility polygon*** of A from a point b is the set of all points of A visible from b. A polygon A is ***star-shaped*** if there exists a point $a \in A$ such that for any point $b \in A$, b is internally visible from a; a is called a star point of A. A ***ray*** is defined as the half-line drawn from a point a through another point b and is denoted by *Ray* $[a, b)$. A ***wedge*** is the region of the plane bounded by two rays drawn from the same point. Given any three distinct points $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ and $p_k = (x_k, y_k)$, let $S = x_k(y_i - y_j) + y_k(x_j - x_i) + y_j x_i - y_i x_j$. If $S > 0$,

*Computer Science Group, Tata Institute of Fundamental Research, Bombay - 400 005, India, E-Mail :ghosh@tifrvax.bitnet

†Computer Systems and Communication Group, Tata Institute of Fundamental Research, Bombay - 400 005, India, E-Mail : manil@tifrvax.bitnet

then $p_ip_jp_k$ is a *left turn*. If $S < 0$, then $p_ip_jp_k$ is a *right turn*. If $S = 0$, then the three points are collinear.

2 An algorithm for determining the intersection type

Let A and B be two given star-shaped polygons. Let a_0 and b_0 be the star points of A and B respectively. We assume that the vertices of A and B are given in clockwise order with their respective x and y coordinates. If A or B is not known to be a star-shaped polygon or their star points are not given, we use the parallel algorithm in [3] for computing the kernel of a simple polygon to settle the issues. However, the algorithm in [3] assumes the CREW PRAM model. So assuming that star points a_0 and b_0 are given, our algorithm runs in the EREW PRAM model. The symbol A (respectively, B) is used to denote the region of the plane enclosed by A (respectively, B) and $bd(A)$ (respectively, $bd(B)$) denotes the boundary of A (respectively, B). We join a_0 and b_0 by a line and the line segment a_0b_0 may or may not intersect $bd(A)$ or $bd(B)$. This leads to the following four cases.
Case 1: a_0b_0 intersects both $bd(A)$ and $bd(B)$.
Case 2: a_0b_0 intersects only $bd(A)$.
Case 3: a_0b_0 intersects only $bd(B)$.
Case 4: a_0b_0 does not intersect $bd(A)$ and $bd(B)$.

Consider *Case 1*. Since a_0b_0 intersects both $bd(A)$ and $bd(B)$, either $bd(A)$ intersects $bd(B)$ or A and B are disjoint. Take a_0 as the reference point. Let a_0b_0 intersect $bd(A)$ and $bd(B)$ at a' and b' respectively. Define b_{min} (respectively, b_{max}) to be a vertex of B such that all vertices of B lie to the right (respectively, left) of $Ray\,[a_0, b_{min})$ (respectively, $Ray\,[a_0, b_{max})$) (Figure 1). The chain formed by the vertices of B from b_{min} to b_{max} in counterclockwise order will be referred to as $chain(b_{min}, b_{max})$. Let a_{min} (respectively, a_{max}) be the point of intersection of a_0b_{min} (respectively, a_0b_{max}) and $bd(A)$. Note that if a_0b_{min} or a_0b_{max} does not intersect $bd(A)$, $bd(A)$ intersects $bd(B)$. So, we assume for the rest of the algorithm for this case that $bd(A)$ intersects both a_0b_{min} and a_0b_{max}. The chain formed by vertices of A from a_{min} to a_{max} in clockwise order will be referred to as $chain(a_{min}, a_{max})$. It is a straightforward task to compute a', b', b_{min}, b_{max}, a_{min}, a_{max} in the CREW model in $O(logn)$ time using $O(n/logn)$ processors. It can also be computed in $O(logn)$ time using $O(n/logn)$ processors using the standard simulation of CREW PRAMs on the EREW PRAMs ([7]).

By testing for intersection between $chain(a_{min}, a_{max})$ and $chain(b_{min}, b_{max})$, it can be decided whether A and B are intersecting or disjoint. By ignoring A, compute the external boundary of B visible from a_0 by the algorithm in Section 3; call it $vchain(b_{min}, b_{max})$.
Lemma 1: (Ghosh [4]) $chain(a_{min}, a_{max})$ intersects $chain(b_{min}, b_{max})$ if and only if $chain(a_{min}, a_{max})$ intersects $vchain(b_{min}, b_{max})$.

Obtain the merged list by merging the vertices of $chain(a_{min}, a_{max})$ and $vchain(b_{min}, b_{max})$ by their relative polar angles with respect to a_0. If we draw rays from a_0 through every vertex of the merged list, these rays divide the plane into wedges. In every wedge, there is an edge of $chain(a_{min}, a_{max})$ and an edge of $vchain(b_{min}, b_{max})$. By checking for intersection between the pair of edges in each wedge, it can be decided whether A and B are disjoint or intersecting. The merging of $chain(a_{min}, a_{max})$ and $vchain(b_{min}, b_{max})$ can be done by the algorithm in [5]. Moreover, for every vertex a_i (respectively, b_i) of the merged list the algorithm gives the vertex of B (respectively, A) in the merged list which precedes a_i (respectively, b_i). It means that the pair of edges in each wedge is known. Using the standard simulation of CREW PRAMs on EREW PRAMs ([7]), the pair of edges in each wedge can be stored with their respective vertices before checking the intersection.

Consider *Case 2*. Since a_0b_0 intersects only $bd(A)$, either $bd(A)$ intersects $bd(B)$ or A is contained in B. As a_0 belongs to both A and B, a_0 is chosen as the reference point (Figure 2). Let us denote the intersection point of a_0b_0 and $bd(A)$ as both a_{min} and a_{max}. Further, let us denote the point of intersection of $bd(B)$ and $Ray[a_0, b_0)$ as both b_{min} and b_{max}. So, $chain(a_{min}, a_{max}) = bd(A)$ and $chain(b_{min}, b_{max}) = bd(B)$, both traversed in the clockwise direction. By ignoring A, we compute the internal visible boundary of B from a_0; call it $vchain(b_{min}, b_{max})$. Once $chain(a_{min}, a_{max})$ and $vchain(b_{min}, b_{max})$ are obtained, merging the vertices of both chains with respect to their relative polar angles at a_0 and detecting the intersection between them follow as in Case 1.

Consider *Case 3*. Since a_0b_0 intersects only $bd(B)$, either $bd(A)$ intersects $bd(B)$ or B is contained in A. As b_0 belongs to both A and B, b_0 is taken as the reference point. The rest of the computation is same as stated in Case 2 except the roles of A and B are interchanged.

Consider *Case 4*. Since a_0b_0 does not intersect both $bd(A)$ and $bd(B)$, we extend a_0b_0 in either direction and denote the intersection point with $bd(A)$ as a' and that of $bd(B)$ as b'. If the length of $a'a_0$ is less than that of $b'a_0$, either $bd(A)$ intersects $bd(B)$ or A is contained in B. This is same as Case 2. If the length of $a'a_0$ is greater than that of $b'a_0$, then either $bd(A)$ intersects $bd(B)$ or B is contained in A. This is same as Case 3. Now we formally state our algorithm ***Intersection_type*** for determining the type of intersection of two star-shaped polygons A and B.

Theorem 1: The intersection type of two star-shaped polygons can be determined in $O(logn)$ time using $O(n/logn)$ processors in the EREW PRAM computational model, where n is the total number of vertices of the polygons.

Remarks : The above algorithm can be used to detect the intersection type of a star polygon and a simple polygon, and computing the intersection region of a star polygon and a convex polygon.

3 An algorithm for computing the visibility polygon

In this section we present a simple parallel algorithm for computing $vchain(b_{min}, b_{max})$ from a_0 in $O(logn)$ time using $O(n/logn)$ processors in the EREW PRAM computation model. Observe that an edge of $vchain(b_{min}, b_{max})$ is either partially or totally an edge of $chain(b_{min}, b_{max})$, or a segment b_iz where a_0, b_i and z are collinear and z is a point on $chain(b_{min}, b_{max})$. So, the task of computing $vchain(b_{min}, b_{max})$ from $chain(b_{min}, b_{max})$ is to construct all such segments b_iz.

Let $chain(b_{min}, b_{max}) = (b_1, b_2, ..., b_k)$ where $b_1 = b_{min}$ and $b_k = b_{max}$. A vertex $b_i \in chain(b_{min}, b_{max})$ is said to be a *left vertex* if b_{i-1} and b_{i+1} are to the left of $Ray[a_0, b_i)$ and $b_{i-1}b_ib_{i+1}$ is a left turn. In Figure 3, left vertices are $b_{11}, b_{13}, b_{15}, b_{17}$ and b_{19}. Extend a_0b_i from b_i to $chain(b_{min}, b_{max})$ and let z be the point of intersection. Then the segment b_iz and the portion of the chain between b_i and z (excluding b_i and z) are called as *left segment* and *left chain* respectively. A vertex $b_i \in chain(b_{min}, b_{max})$ is said to be a *right* *vertex* if b_{i-1} and b_{i+1} are to the right of *Ray* $[a_0, b_i)$ and $b_{i-1}b_ib_{i+1}$ is a left turn. In Figure 3, right vertices are b_3, b_5, b_7 and b_9. Analogously, we define a *right segment* and a *right chain*. In the following lemma we show that if all left and right chains are removed from $chain(b_{min}, b_{max})$, the resulting chain is $vchain(b_{min}, b_{max})$.

Lemma 2: A vertex b_i is visible from a_0 if and only if b_i does not belong to a left or right chain.

Now we state the procedure for locating left and right chains in $chain(b_{min}, b_{max})$. For every vertex b_i in $chain(b_{min}, b_{max})$ let α_i be the clockwise angle subtended at a_0 by b_i with respect to a_0b_{min}. Let $S_{max}(i)$ denotes the maximum of $\alpha_1, \alpha_2, ..., \alpha_i$. We say a left vertex b_i is *proper* if $\alpha_i = S_{max}(i)$. In Figure 3, proper left vertices are b_{11}, b_{13} and b_{19}. Analogously, let $S_{min}(i)$ denotes the minimum of $\alpha_i, \alpha_{i+1}, ..., \alpha_k$. We say a right vertex b_i is *proper* if $\alpha_i = S_{max}(i)$. In Figure 3, proper right vertices are b_7 and b_9.

Lemma 3: A left vertex or a right vertex is visible from a_0 if and only if it is proper.

Corollary 1: The proper vertices are in the sorted angular order with respect to a_0.

For every vertex b_i in $chain(b_{min}, b_{max})$ store the angle α_i in an array. The proper vertices can be located by using the parallel prefix algorithm [6]. The remaining task is to construct the left and right segments for every left and right proper vertices respectively. Since the given polygons are star-shaped, no two left and right chains overlap. This property helps in computing left and right segments. Consider three consecutive proper vertices b_i, b_j and b_m where $i < j < m$. If b_j is a left vertex, then the closest point to a_0 among the points of intersection of *Ray* $[a_0, b_j)$ with $chain(b_{min}, b_{max})$ excluding b_j lies on edge b_sb_{s+1} where $j < s < m$. Moreover, *Ray* $[a_0, b_j)$ does not intersect any edge between b_j and b_m except b_sb_{s+1}. Therefore every vertex between b_j and b_s lies to the left of *Ray* $[a_0, b_j)$ and every vertex between b_{s+1} and b_m lies to the right of *Ray* $[a_0, b_j)$. If b_j is a right vertex, then the closest point to a_0 among the points of intersection of *Ray* $[a_0, b_j)$ excluding b_j with $chain(b_{min}, b_{max})$ lies on edge b_sb_{s+1} where $i \le s < j$. For each vertex of $chain(b_{min}, b_{max})$, its previous and next proper vertices can be located by parallel prefix algorithm and then all left and right segments can be computed by checking the intersection of each edge with the corresponding ray. During the computation of left and right segments, mark the vertices that belong to left or right chains and delete them by compacting the array to obtain $vchain(b_{min}, b_{max})$

Theorem 2: The visibility polygon of an n-sided star-shaped polygon from a point can be computed in $O(logn)$ time using $O(n/logn)$ processors in the EREW PRAM computation model.

References

1. M.J. Atallah and D.Z. Chen, *An optimal parallel algorithm for the visibility polygon of a simple polygon from a point*, Proc. of the fifth ACM Annual Symposium on Computational geometry (1989) 114-123.

2. M.J. Atallah, R. Cole and M.T. Goodrich, *Cascading divide-and-conquer: A technique for designing parallel algorithms*, SIAM Journal on Computing 18(1989) 499-532.

3. R. Cole and M.T. Goodrich, *Optimal parallel algorithms for polygon and point-set problems*, Proc. of the fourth ACM Annual Symposium on Computational geometry (1988) 201-210.

4. S. K. Ghosh, *A linear-time algorithm for determining the intersection type of two star polygons*, Lecture Notes in Computer Science, Springer Verlag **181**(1984) 317-330.

5. T. Hagerup and C. Rüb, *Optimal merging and sorting on the EREW PRAM*, Information Processing Letters **33**(1989) 181-185.

6. R.M. Karp and V. Ramachandran, *A survey of parallel algorithms for shared-memory machines*, Technical Report UCB/CSD 88/408, University of California, Berkeley, 1988.

7. U. Vishkin, *Implementation of simultaneous memory address access in models that forbid it*, Journal of Algorithms **4**(1983) 45-50.

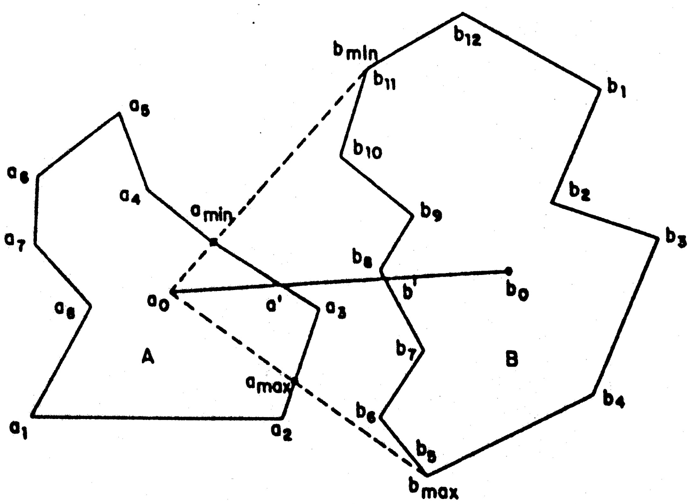

Figure 1

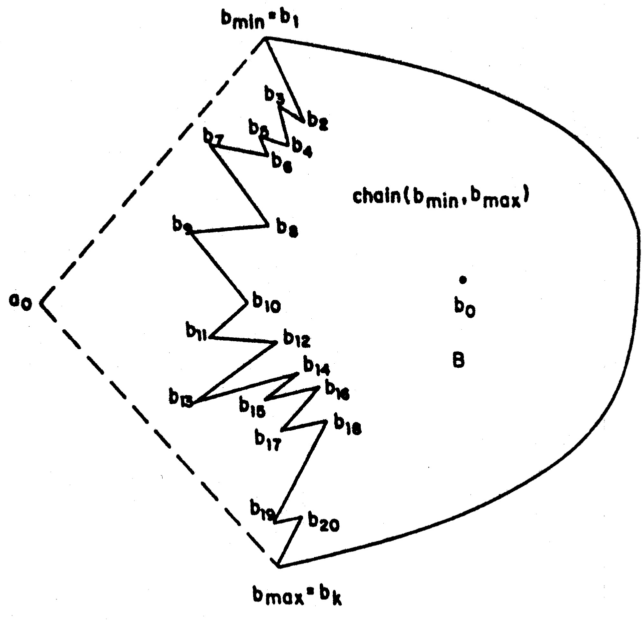

Figure 3

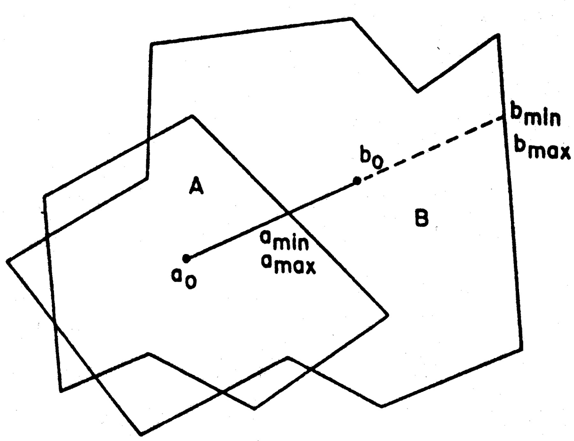

Figure 2