A Simple Linear Algorithm for Intersecting Convex Polygons

Godfried T. Toussaint
School of Computer Science
McGill University
Montreal, Quebec, Canada H3A 2A7

ABSTRACT

Let P and Q be two convex polygons with m and n vertices, respectively, which are specified by their cartesian coordinates in order. A simple O(m+n) algorithm is presented for computing the intersection of P and Q. Unlike previous algorithms, the new algorithm consists of a two-step combination of two simple algorithms for finding convex hulls and triangulations of polygons.

Key words: *Algorithms - Complexity - Computational geometry - Convex polygons - Intersection*

1. Introduction

Let $P = \{p_1, p_2,..., p_m\}$ and $Q = \{q_1, q_2,..., q_n\}$ be two convex polygons whose vertices are specified by their cartesian coordinates in clockwise order. It is assumed that the polygons are in *standard* form, i.e., the vertices of each polygon are distinct and no three consecutive vertices are collinear [7]. A common problem in computer graphics, image processing and many sub-problems in computational geometry is that of computing the intersection of P and Q which is itself another convex polygon of at most m+n vertices. The brute force method converts each edge of each polygon to a half-plane and then inserts each half-plane into a tentative intersection. This leads to an algorithm with O(mn) time complexity. If a divide and conquer strategy is used instead, then an O((n+m) log (n+m)) complexity may be achieved. An alternate O((n+m) log (n+m)) time algorithm was proposed by Kundu [14] where he first sorts all the half-planes by the angles of their defining bounding lines and then uses a linear time procedure for computing their intersection. These procedures do not exploit the convexity properties of the polygons. Several linear time algorithms that do make use of convexity have recently been proposed for this problem [6]-[9]. The algorithms in [7]-[9] are relatively cumbersome to program due to the large number of cases that arise when intersecting the trapezoids that result with the “slab” method employed in [7]. The simplest and most elegant of the above algorithms is the one due to O’Rourke et al. [6]. Here two “bugs”, one of the boundary of P and the other of the boundary of Q, “chase” each other in an alternating fashion as each tries to cross the “forward line of sight” of the other.

In this paper we present a new simple algorithm for constructing the intersection of P and Q in O(m+n) time in the worst case. Unlike the previous algorithm of [6]-[9] the new algorithm is a combination of existing simple procedures for computing convex hulls and triangulations of polygons. Because of this it may be a little slower in practice than the algorithm O’Rourke et al. [6], depending on which convex hull algorithm is employed. On the other hand little new specialized code is needed if the convex hull and triangulation codes are already available. Furthermore, the algorithm presented here is conceptually transparently clear and affords an easy proof of correctness.

2. Preliminary results

In this section we present an informal description of the algorithm and some preliminary results. A detailed description of the algorithm and a proof of correctness is included in section three. We assume that the interiors of the polygons P and Q intersect. If this is not true there is no intersection polygon to construct and the intersection then is either a line segment, a point, or the empty set. In any case, determining whether the interiors of P and Q intersect can be easily performed in $O(\log(m+n))$ time [1], [2]. In order to simplify the description of the algorithm and to prevent the essential aspects from drowning in a sea of detail we further assume that no three vertices in $P \cup Q$ are collinear and all vertices in $P \cup Q$ are distinct. This implies that if P and Q intersect then so do their interiors. It also implies that if the boundaries of P and Q intersect the intersection points do not coincide with vertices of P or Q. Special case tests can be included for the "singularities" that arise when this assumption is not made and these are similar for all the algorithms outlined above [6]-[9]. A clear exposition on handling these cases is given by O'Rourke et al. [6].

Consider then two polygons P and Q whose boundaries intersect and construct the convex hull of their union (Fig. 1). Let the boundaries of P and Q intersect at k intersection points $I_1, I_2,..., I_k$ indexed in clockwise order. The boundaries of P, Q, and the convex hull of P and Q, $CH(P \cup Q)$, partition the plane into $2k + 1$ bounded regions: the convex intersection region (I_1,..., I_2,..., I_k,...), k regions where P and Q lie outside $P \cap Q$ (P_{st} associated with I_s and I_t, and Q_{uv} associated with I_u and I_v) and k "pockets" $K_1, K_2,..., K_k$ where a pocket K_v is associated with I_v and is in the region inside $CH(P \cup Q)$ but outside $P \cup Q$. With each pocket K_v we associate a bridge which is an edge of $CH(P \cup Q)$, denoted by $B_v(p_{i_v}, q_{j_v})$ and which joins vertex p_{i_v} of P with vertex q_{j_v} of

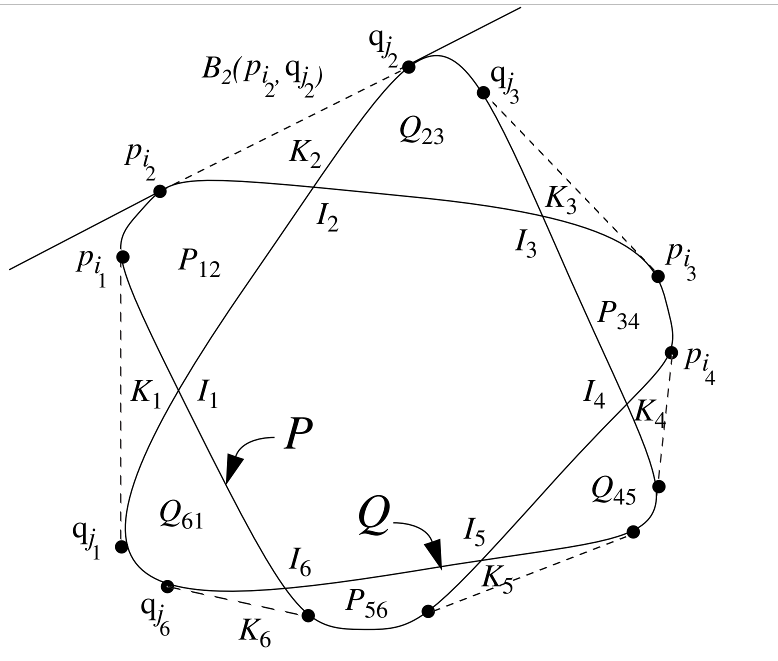

Fig. 1

Q. The algorithm for computing $P \cap Q$ can now be described informally as the following three-step procedure: first construct the convex hull of $P \cup Q$, then for each bridge B_i find its corresponding intersection I_i, and finally "merge" the corresponding polygonal chains that connect adjacent intersection points.

We now prove some lemmas that we will need to prove the correctness of the algorithm described in section three. Let $L(u, v)$ denote the directed line through u and v in the direction u, v and let $RH(u, v)$ denote the closed half-plane to the right of $L(u, v)$.

The following lemma has been proved by Guibas et al. [4] using a powerful new framework involving convolutions (a special case of fiber products) of polygons. We include an alternate elementary proof here for completeness.

Lemma 2.1: If P and Q intersect there exists a unique mutual one-to-one correspondence between the bridges of $CH(P \cup Q)$ and the intersection points of $P \cap Q$.

Proof: Let $B(p_i, q_j)$ be a bridge and refer to Fig. 2. $L(p_i, q_j)$ must be a line of support for both P and Q. Furthermore P and Q must both lie in $RH(p_i, q_j)$. Trace P in a clockwise manner starting at p_i until an edge of P intersects an edge of Q at I. Similarly trace Q in a counter-clockwise manner starting at q_j until an edge of Q intersects an edge of P. From convexity it follows that this intersection point is also I and thus I corresponds to $B(p_i, q_j)$. On the other hand assume that I is some intersection point between edge $\overline{p_k p_{k+1}} \in P$ and $\overline{q_l q_{l+1}} \in Q$. Since $P \in RH(p_k, p_{k+1})$ and $Q \in RH(q_l, q_{l+1})$ it follows that no edge of P or Q other than $\overline{p_k p_{k+1}}$ and $\overline{q_l q_{l+1}}$ may intersect the region $R = RH(p_{k+1}, p_k) \cap RH(q_{l+1}, q_l)$. Furthermore, since angle

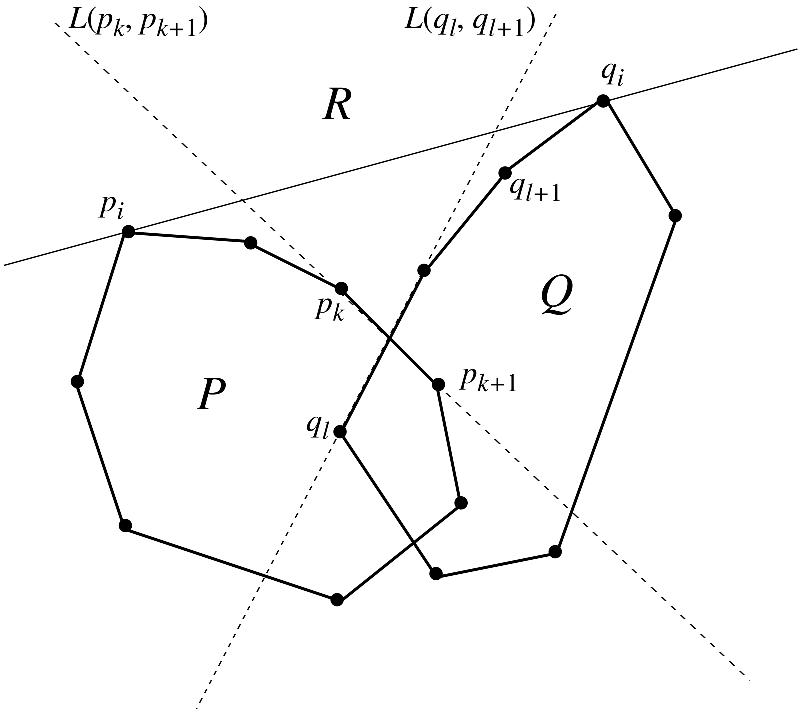

Fig. 2

$p_k\ I\ q_{l+1} < 180^{o}$ it follows that there must exist an edge $p_i q_j \in CH(P \cup Q)$ that intersects R and this is the bridge corresponding to I. Q.E.D.

We now define a restricted class of simple polygons and establish some results concerning their triangulation. While we are not explicitly interested in triangulating these polygons these results will be useful in understanding, and proving the correctness of the algorithm. A polygonal chain $C(p_i, p_{i+1}, \ldots, p_j)$ is a portion of consecutive vertices and edges of a simple polygon. If all turns are *right* (convex angles) we have a *convex chain*. If all turns are *left* (reflex angles) we have a *concave chain*.

Definition: A *sail* polygon P_s is one that contains an edge $\overline{p_i p_{i+1}}$ called the *mast* of P and a vertex p_j called the *sail tip* of P such that p_j is connected to p_i and p_{i+1} by *concave chains* (Fig. 3.) Note that P_s must be completely in $RH(p_i, p_{i+1})$.

Definition: A line segment, lying in P, that connects two non-adjacent vertices of P is a *diagonal* of P

Definition: Three consecutive vertices p_i, p_{i+1}, p_{i+2} are said to form an *ear* of P at p_{i+1} if the diagonal joining p_i and p_{i+2} lies in P.

Definition: Two ears are *non-overlapping* if their interior regions are disjoint.

Meisters [5] proves the following "two-ears" theorem

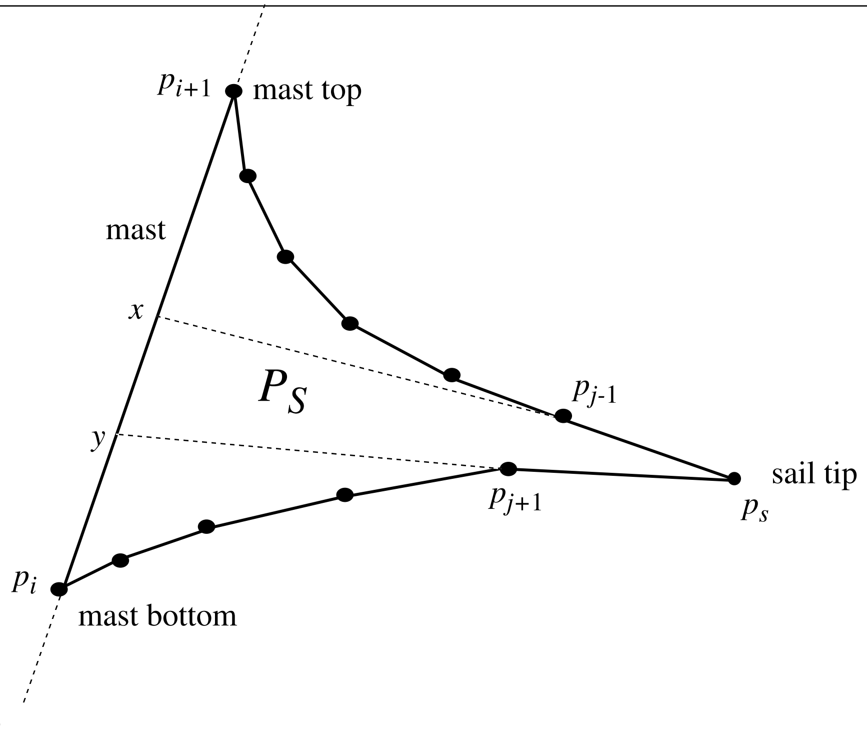

Fig. 3

Lemma 2.2: Every polygon of n sides (n>3) has at least two non-overlapping ears.

This theorem leads Meisters to propose an O(n^3) algorithm for triangulating simple polygons by finding ears and "cutting them off". *Sail* polygons on the other hand have enough structure that we can "cut off all the ears" in O(n) time. Note that, by definition, only *convex* vertices can be *ears*. Also, a sail polygon has the property that only p_i, p_{i+1} and p_j are *convex*, and thus candidates for *ears*. We thus have the following results.

Lemma 2.3: The *tip* of a *sail* polygon is an *ear*.

Proof: Extend $\overline{p_j p_{j-1}}$ and $\overline{p_j p_{j+1}}$ to intersect $L(p_i, p_{i+1})$ at x and y, respectively, (Fig. 3.) Point x must lie on $p_i p_{i+1}$ or else p_j could not be joined to p_i with a *concave* chain. The same argument holds for y. By construction $p_j\, p_{j-1}\, x\, y\, p_{j+1}\, p_j$ forms a triangle and by convexity it lies in P_s. Therefore the diagonal $\overline{p_{j-1} p_{j+1}}$ lies in P_s. Q.E.D.

Lemma 2.4: Either the *mast top* or the *mast bottom* of a *sail* polygon is an *ear*.

Proof: Only p_i, p_{i+1}, and p_j in P_s can be ears. By Lemma 2.3 p_j must be an ear. By Lemma 2.2 P_s must have at least two ears. Therefore either p_i or p_{i+1} must be an ear. Q.E.D.

Lemma 2.4 allows us to triangulate P_s in O(n) time by "wrapping the sail around the mast" until only the sail tip remains. In other words, starting at the mast we cut off the top ear or the bottom ear and proceed to the polygon remaining. The correctness of the algorithm follows from the induction hypothesis that, at each step, the polygon remaining is a *sail* polygon. The proof of this induction hypothesis is left as an easy exercise for the reader. The linearity follows from the fact that at each step, which takes constant time P_s contains one less vertex. Note that other linear time algorithms could be used for triangulating P_s. For example P_s is *edge-visible* from the mast and thus the algorithm of [13] can be used. Alternately, P_s is *monotonic* in the direction perpendicular to the *mast* and therefore the algorithm of Garey et al. [3] applies. The advantages of the algorithm presented here are that, first, unlike those of [13] and [3] it does not incorporate backtracking and is thus simpler, and second, the last diagonal to be added is $\overline{p_{j-1} p_{j+1}}$. This latter property is crucial for solving the polygon intersection problem, The "ear-cutting" algorithm is in essence a trimmed version of the algorithm of Garey et al. [3] that exploits the added structure that *sail* polygons have over *monotone* polygons.

3. The algorithm

Before describing the complete algorithm we present PROCEDURE STEPDOWN which receives as input a bridge $B_k(p_i, p_j)$ of $CH(P \cup Q)$ and exits with the corresponding pair of edges that determine the intersection point I_k. Without loss of generality assume p_1 and q_1 form the bridge, Q is given counter-clockwise order, and $\overline{p_s p_{s+1}} \cap \overline{q_t q_{t+1}}$ determines the intersection point I. (Fig.4.) A convenient data structure for P and Q here is a doubly-linked circular list so that we can scan in either direction and set up pointers between the vertices of P and those of Q. Procedure STEPDOWN finds the two vertices p_s and q_t that can then be used to compute I. The variables p_i

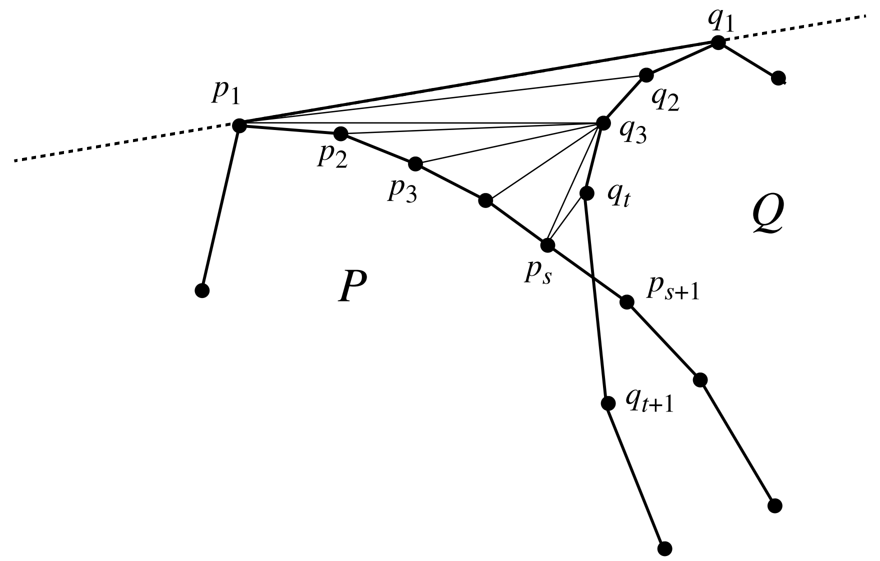

Fig. 4

and q_j are the "current" vertices under consideration and are a tentative solution. When the algorithm stops $p_i = p_s$ and $q_j = q_t$. The boolean variable "finished" indicates when p_s and q_t are reached by taking on the value "true" after an execution of the "repeat" loop.

PROCEDURE STEPDOWN

{initialization} i <-- 1; j <-- 1

repeat

finished <-- true

while ($p_i\, p_{i+1}\, q_{j+1}$) left **do**

begin

j <-- j + 1

finished <-- false

end

while ($q_j\, q_{j+1}\, p_{i+1}$) right **do**

begin

i <-- i + 1

finished <-- false

end

until finished

p_s <-- p_i; q_t <-- q_j

END STEPDOWN

Lemma 3.1: Procedure STEPDOWN correctly computes the intersection point corresponding to a bridge in O(n) time.

Proof: The proof follows essentially from the realization that STEPDOWN is an implementation of the "ear-cutting" triangulation algorithm for *sail* polygons given in the previous section. Note that (p_1, q_1, q_2,..., q_t, I, p_s, p_{s-1},..., p_2) would be a sail polygon if I were a vertex connected to p_s and q_t. Thus the "ear-cutting" algorithm must eventually arrive at $p_s q_t$. Now in a true *sail* polygon the algorithm automatically stops here because $p_{s+1} = q_{t+1}$. However, in this situation this is not the case since p_{s+1} and q_{t+1} belong to different polygonal chains. The tests for left and right turns in the inner WHILE loops of STEPDOWN not only prevent the algorithm from continuing past p_s and q_t, but also determine an ordering for "ear-cutting", by invoking Lemma 2.4. Q.E.D.

We now describe the algorithm for computing the intersection of two intersecting convex polygons P and Q. The portions of the boundaries of P and Q outside $P \cap Q$ will be referred to as *outer chains*, those portions inside $P \cup Q$ as *inner chains*.

ALGORITHM INTERCONPOL

Begin

Step 1. Find the convex hull of the union of P and Q, CH($P \cup Q$).

If $CH(P \cup Q) = P$ (or Q)

then Exit with Q (or P) as the intersection;

Else continue.

Step 2. For each *bridge* of CH($P \cup Q$) call procedure STEPDOWN to compute the

intersection points of $P \cap Q$.

Step 3. Merge the *inner chains* of P and Q determined by the intersection points found in step 2.

End

Theorem 3.1: Algorithm INTERCONPOL correctly computes the intersection polygon of two intersecting convex polygons P and Q in O(m+n) time.

Proof: The correctness of the algorithm follows from Lemmas 2.1 and 3.1. Therefore we turn to its complexity. Finding the convex hull of two intersecting convex polygons in step 1 can be done in O(m+n) time with several algorithms [7], [10], [11]. The simplest of all algorithms is the "rotating caliper" method [11] which, unlike those of [7] and [10], does not involve

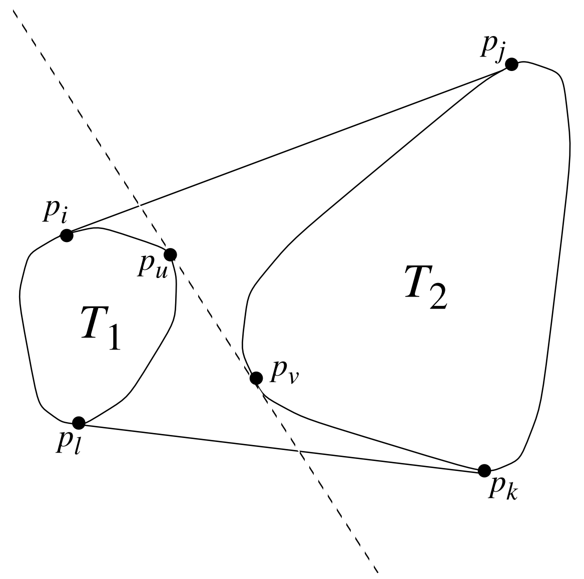

Fig. 5

backtracking and at the same time can answer the question of whether $CH(P \cup Q) = P$ or Q. If there are *k bridges* on $CH(P \cup Q)$ then STEPDOWN is called k times in step 2. Each call requires time linear in the number of vertices processed and the total number of these vertices is the sum total of the vertices on all the *outer chains* of P and Q. Thus step 2 runs in $O(m+n)$ time. Finally, if we leave pointers from the intersection points to the *inner* and *outer* chains in both directions, as we find them in step 2, then the merge step of the inner chains in step 3 can be done in linear time by a mere traversal of the two lists for P and Q.   Q.E.D

4. Concluding remarks

As a final remark we mention that the "ear-cutting" triangulation algorithm for *sail* polygons presented in section two can be applied to the problem of triangulating a set of n points on the plane in $O(n \log n)$ time via divide-and-conquer. Here, if the points have been presorted, at each step we must merge two triangulations T_1 and T_2 which are linearly separable triangulated convex polygons (Fig. 5.) The merge step consists of triangulating the *hourglass* polygon "in between" T_1 and T_2. This region lies outside T_1 and T_2 but inside $CH(T_1 \cup T_2)$. An *hourglass* polygon is a polygon consisting of two edges called the top (bridge p_i, p_j) and the bottom (bridge p_k, p_l) such that p_i and p_l (as well as p_k, p_j) are joined by *concave* chains and (p_i, p_j, p_k, p_l) forms a *convex* quadrilateral. Now consider a *critical line of support* between T_1 and T_2 at p_u and p_v. This line decomposes the *hourglass* polygon into two *sail* polygons P_{s_1} and P_{s_2}. Finding the bridges and the edge $p_u p_v$ can be done in linear time with the rotating calipers [11]. Triangulating the *sail* polygons will thus solve the merge of T_1 and T_2 in linear time which is sufficient to obtain the overall $O(n \log n)$ performance. Note that the triangulation algorithms of [13] and [3] *cannot* be used here since an *hourglass* polygon need be neither *edge-visible* nor *monotone*. Finally, we remark that this algorithm can be applied to the problem of computing distances between crossing convex polygons [12].

5. References

1. Chazelle B (1980) Computational geometry and convexity. Ph.D. thesis, Carnegie-Mellon University

2. Chazelle B, Dobkin D (1980) Detection is easier than computation. *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*. pp 146-153

3. Garey MR, Johnson DS, Preparata FP, Tarjan RE (1978) Triangulating a simple polygon. *Information Processing Lett* 7:175-179

4. Guibas L, Ramshaw L, Stolfi J (1983) A kinetic framework for computational geometry. Technical report. Xerox Park and Stanford University

5. Meisters GH (1975) Polygons have ears. *American Mathematical Monthly*. June/July 1975, 648-651

6. O'Rourke J (1982) A new linear algorithm for intersecting convex polygons. *Comput. Graph Image Processing* 19:384-391

7. Shamos MI (1978) Computational Geometry, Ph.D. thesis, Yale University

8. Shamos MI (1977) *Problems in Computational Geometry*. Carnegie-Mellon University

9. Shamos MI, Hoey D (1976) Geometric intersection problems *Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science*. October 1976, pp 208-215

10. Toussaint GT (1981) Computational geometric problems in pattern recognition. In: Kittler J (ed) *Pattern Recognition Theory and Applications*. p 73-91

11. Toussaint GT (1983) Solving geometric problems with the rotating calipers. *Proc. MELECON*, Athens. Greece

12. Toussaint GT (1984) An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons. *Computing* 32:357-364

13. Toussaint GT, Avis D (1982) On a convex hull algorithm for polygons and its application to triangulation problems. *Pattern Recognition*. 15:23-29

14. Kundu, S., (1987) A new O(n log n) algorithm for computing the intersection of convex polygons. *Pattern Recognition*. 20:419-424