An algorithm for recognizing palm polygons

Subir Kumar Ghosh* Anil Maheshwari† Sudebkumar Prasant Pal‡ C. E. Veni Madhavan‡

Abstract

In this paper, we propose an $O(E)$ time algorithm for recognizing a palm polygon P, where E is the size of the visibility graph of P. The algorithm recognizes the given polygon P as a palm polygon by computing the palm kernel of P. If the palm kernel is not empty, P is a palm polygon.

1. Introduction

In the past, several classes of simple polygons have been introduced in computation geometry. Famous examples of such classes are: star-shaped, monotone, spiral, edge visible and weakly externally visible polygons. Recently, ElGindy and Toussaint [ET] have introduced a new class of polygons called palm polygons. A polygon P is said to be ***palm polygon*** (Figure 1) if there exists a point $x \in P$ such that the Euclidean shortest path from x to any point $y \in P$ makes only left or right turn at each vertex in the path. The set of all such point x is called the ***palm kernel***. The problem of recognizing palm polygons is an open problem and is posed by ElGindy and Toussaint [ET]. In this paper, we propose an $O(E)$ time algorithm for recognizing a palm polygon P, where E is the size of the visibility graph of P. The algorithm recognizes the given polygon P as a palm polygon by computing the palm kernel of P. If the palm kernel is not empty, P is a palm polygon.

We assume that the simple polygon P is given as a counterclockwise sequence of ***vertices*** $v_1, v_2, \ldots, v_n$ with their respective x and y coordinates. We assume that no three vertices of P are collinear. The line segments $v_1v_2, \ldots, v_{n-1}v_n, v_nv_1$ are called ***edges*** of P. The symbol P is used to denote the region of the plane enclosed by P and $bd(P)$ denotes the boundary of P. If p and q are two points on $bd(P)$ then the counterclockwise $bd(P)$ from p to q is denoted as $bd(p,q)$. Two points are said to be ***visible*** if the line segment joining them lies totally inside P. If the line segment joining two points touches $bd(P)$, they are still considered to be visible. A point p is said to be ***weakly visible*** from an edge st, if there is a point z in the ***interior*** of st such that p and z are visible. The ***weak visibility polygon*** of P from an edge is the set of all points of P weakly visible from the edge. The ***visibility graph*** of P is the graph defined with the set of vertices of P as the vertex set and the set of visible pairs of vertices of P as the edge set. We denote the number of edges in the visibility graph by E. Let $SP(u,v)$ denote the Euclidean shortest path inside P from a point u to another point v. Given any three points $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$, and $p_k = (x_k, y_k)$. let $S = x_k(y_i - y_j) + y_k(x_j - x_i) + y_jx_i - y_ix_j$. If $S < 0$, then $p_ip_jp_k$ is a right turn. If $S > 0$. then $p_ip_jp_k$ is a left turn. If $S = 0$. then the three points are

*Computer Science Group, Tata Institute of Fundamental Research. Bombay-400005,INDIA

†Computer Systems and Communications Group, Tata Institute of Fundamental Research, Bombay-400005, INDIA

‡Department of Computer Science and Automation, Indian Institute of Science, Bangalore-560012, INDIA

collinear. An edge v_iv_j of $SP(v_k, v_m)$ is an *eave* if $SP(v_k, v_m)$ makes a right (or left) turn at v_i and makes a left (respectively, right) turn at v_j where v_i, v_j, v_k and v_m are distinct vertices.

2. The recognition algorithm

Let v_iv_k be an eave in P. Extend v_iv_k from v_i (respectively, v_k) till it does not intersect the exterior of P and let z_i (respectively, z_k) be the point of intersection. The segments v_iz_i and v_kz_k are called *lids* of v_i and v_k respectively. The segment v_iz_i is called the *right lid* of v_i if $z_i \in bd(v_i, v_k)$ (Figure 2) and the *left lid* of v_i (Figure 3), otherwise. If v_iz_i is a right (respectively, left) lid of v_i, then the region of P bounded by v_iz_i and $bd(v_i, z_i)$ (respectively, $bd(z_i, v_i)$) is called the *right* (respectively, *left*) *forbidden region* of v_i. The forbidden regions of v_i and v_k are also referred as the forbidden regions of v_iv_k. In the following lemma, we show that no point of the palm kernel lies in the forbidden region of any eave.

Lemma 1: A polygon P is a palm polygon if and only if there is a point $z \in P$ such that z is not in the forbidden region of any eave.

Proof: If there is a point $z \in P$ such that z is not in the forbidden region of any eave, we show that P is a palm polygon. Since z does not lie in the forbidden region of any eave, the shortest path from z to any point of P cannot have an eave. So z belongs to the palm kernel of P by definition. Therefore, P is a palm polygon.

Conversely, if every point $z \in P$ lies in the forbidden region of an eave, we show that P is not a palm polygon. Since z lies in the forbidden region of some eave v_iv_k, the shortest path from z to every point of the other forbidden region of v_iv_k contains v_iv_k. Therefore P is not a palm polygon. **Q.E.D.**

The above lemma suggests a simple procedure for computing the palm kernel as follows. Remove the forbidden regions of all eaves from P. If the resulting region of P is not empty, then the region is the palm kernel. Otherwise, P is not a palm polygon. Our algorithm first locates all eaves and their forbidden regions in P and then it removes the forbidden regions from P by a procedure similar to the algorithm of Lee and Preparata [LP] for computing the kernel of a simple polygon.

Now we state the procedure for locating all eaves and their forbidden regions by computing the weak visibility polygon from each edge of P. Compute $BVP(P, v_iv_{i+1})$ where $BVP(P, v_iv_{i+1})$ denotes the boundary of the weak visibility polygon of P from the edge v_iv_{i+1} (Figure 4). An edge of $BVP(P, v_iv_{i+1})$ is called a *constructed edge* if only the endpoints are on $bd(P)$. Note that one of the two endpoints of any constructed edge is a vertex of P. Let v_kz_k be a constructed edge. If v_k precedes z_k in clockwise order on $BVP(P, v_iv_{i+1})$, then we say v_kz_k is a *left* constructed edge and a *right* constructed edge, otherwise. Assume that v_kz_k is a left constructed edge. Let v_p be the previous vertex of v_k in $SP(v_{i+1}, v_k)$. If v_p is not v_{i+1} then v_pv_k is an eave and v_kz_k is the left lid of v_k. Assume that v_pv_k is an eave. Let z_p be the point of intersection of v_iv_{i+1} and the ray drawn from v_k through v_p. So, v_pz_p is the left lid of v_p. Analogously, from a right constructed edge we locate the corresponding eave and its forbidden regions. Since each eave introduces a constructed edge, all eaves and their forbidden regions can be located by considering all constructed edges in each visibility polygon. Our procedure computes the visibility polygon from all edges of P by the algorithm of Hershberger [H] for computing the visibility graph of a simple polygon.

Observe that P can have $O(n^2)$ eaves (Figure 5). To compute the palm kernel from eaves, we show that it is enough to consider at most two eaves for each vertex. Consider a vertex v_i. Assume that v_i is a vertex of several eaves. Let v_iv_k be the eave such that there is no eave connecting v_i to a vertex of $bd(v_i, v_k)$ (Figure 6). Observe that any left forbidden region of v_i is contained inside the left forbidden region of v_i due to the eave v_iv_k. So, by removing the left forbidden region of v_i due to the eave v_iv_k, we remove all left forbidden regions of v_i. Let v_iv_j be the eave such that there is no eave connecting v_i to a vertex of $bd(v_j, v_i)$ (Figure 6). Observe that any right forbidden region of v_i is contained inside the right forbidden region of v_i due to the eave v_iv_j. So, by removing the right forbidden region of v_i due to the eave v_iv_j, we remove all right forbidden regions of v_i. The eaves v_iv_j and v_iv_k can be located in time proportional to the number of edges incident on v_i in the visibility graph of P. Therefore the appropriate left and right lids for all vertices can be determined in $O(E)$ time. Since we consider only one left and right lids of a vertex v_i, from now on we assume that a vertex has at most one left and right lid.

Now we state the procedure for computing the palm kernel. The procedure traverses $bd(P)$ in counterclockwise order starting from v_1 and constructs a sequence of closed boundaries $R_0, R_1, ..., R_n$, where R_0 is $bd(P)$ and R_n is the boundary of the palm kernel of P. At each vertex v_i it computes R_i from R_{i-1} by computing the intersection of the left and right lids of v_i with R_{i-1}. Let $R_{i-1}=(c_1, c_2, ..., c_m)$ where the vertices are numbered in counterclockwise order. Let u_j and w_{j-1} be the endpoints of the lid containing c_jc_{j-1} where u_j, c_j, c_{j-1} and w_{j-1} are consecutive points on the lid (Figure 7). We refer the lid (u_j, w_{j-1}) as the corresponding lid of c_jc_{j-1}. Note that if $c_j \in bd(P)$, then $u_j = c_j$ and if $c_{j-1} \in bd(P)$, then $w_{j-1} = c_{j-1}$. For any vertex $v_s \in bd(u_j, u_{j+1})$, c_j is called the ***left apex*** of v_s. Analogously, for any vertex $v_s \in bd(w_{k-1}, w_k)$, c_k is called the ***right apex*** of v_s. While traversing $bd(P)$ in counterclockwise order, at each point u_j the left apex moves from c_{j-1} to c_j and at each point w_k the right apex moves from c_k to c_{k+1}. If a new edge is added to R_{i-1} while computing R_i from R_{i-1}, the endpoints of the corresponding lid of the new edge are inserted on $bd(P)$. Note that since one of the endpoints is a vertex of P, we still insert another copy of the vertex to indicate an endpoint of the lid. If an edge is removed from R_{i-1} while computing R_i from R_{i-1}, the endpoints of the corresponding lid of the edge are deleted from $bd(P)$. Before computing R_i from R_{i-1}, the doubly linked list representing $bd(P)$ consists of the vertices of P and the endpoints of the corresponding lids of the edges of R_{i-1}.

Assume that R_{i-1} has been computed so far and v_i is the current vertex under consideration. If v_i is not a vertex of any eave then $R_i = R_{i-1}$. Otherwise, the following two cases arise.

Case 1. The vertex v_i belongs to R_{i-1}.

Case 2. The vertex v_i does not belong to R_{i-1}.

Consider Case 1. If the left lid of v_i exists, then traverse R_{i-1} in clockwise order from v_i till the left lid of v_i intersects R_{i-1} at a point x (Figure 8). Remove the clockwise boundary of R_{i-1} from v_i to x and add v_ix to R_{i-1}. If the right lid of v_i exists, then traverse R_{i-1} in counterclockwise order from v_i till the right lid of v_i intersects R_{i-1} at a point y (Figure 8). Remove the counterclockwise boundary of R_{i-1} from v_i to y and add v_iy to R_{i-1}. Now R_{i-1} is R_i.

Consider Case 2. Let c_j and c_k be the left and right apexes of v_i. So, $v_i \in bd(u_j, w_k)$. The procedure is executed once for each lid of v_i. Here we describe the procedure for any lid of v_i. Let z_i be the other endpoint of the lid of v_i. The procedure performs one of the following steps.

Step 1. **if** $z_i \in bd(u_j. w_k)$ **then if** both u_j and w_k lies in the forbidden region of v_i **then** $R_i := \emptyset$ **else** $R_i := R_{i-1}$ (Figure 9).

Step 2. **if** v_iz_i intersects c_kw_k **then if** w_k lies in the forbidden region of v_i **then** $R_i := R_{i-1}$ **else** $R_i := \emptyset$ (Figure 10).

Step 3. **if** v_iz_i intersects c_ju_j **then if** u_j lies in the forbidden region of v_i **then** $R_i := R_{i-1}$ **else** $R_i := \emptyset$.

Step 4. **if** $z_i \in bd(w_k, u_j)$ and v_iz_i does not intersect both c_ju_j and c_kw_k **then**

if w_k lies in the forbidden region of v_i **then** (Figure 11)

begin traverse R_{i-1} from c_k in both clockwise and counterclockwise order to locate the intersection points x and y respectively of v_iz_i and R_{i-1}; remove the counterclockwise boundary of R_{i-1} from x to y; add the edge xy to R_{i-1}

end else

begin traverse R_{i-1} from c_j in both clockwise and counterclockwise order to locate the intersection points x and y respectively of v_iz_i and R_{i-1}; remove the counterclockwise boundary of R_{i-1} from x to y; add the edge xy to R_{i-1}

end

We now analyze the time complexity of the algorithm. All eaves and their forbidden regions can be computed in $O(E)$ time using the algorithm of Hershberger [H]. The algorithm of Hershberger [H] requires a triangulation of P. Due to the recent result of Chazelle [Ch] it is possible to triangulate P in $O(n)$ time. Hence the visibility polygon from all edges of P can be computed in $O(E)$ time. Since we consider at most two eaves for each vertex, the total number of endpoints of lids inserted on $bd(P)$ or deleted from $bd(P)$ is $O(n)$. Since each insertion or deletion takes $O(1)$ time, the total time for insertion and deletion of the endpoints of lids is $O(n)$. Since the left or right apex at each endpoints of a lid can be updated in $O(1)$ time, total time for updating the apexes is $O(n)$. For each lid, it takes $O(1)$ time to determine whether the lid intersects R_{i-1}. The cost of computing the intersection of R_{i-1} and the lid is proportional to the number of vertices deleted from R_{i-1}. It has been shown by Lee and Preparata [LP] that the total number of vertices deleted from $R_0, R_1, ..., R_{n-1}$ is $O(n)$. So, the total cost of computing $R_1, ..., R_n$ is $O(n)$. We summarize our result in the following theorem. (***Proof omitted in this version***)

Theorem 1: The palm kernel of a polygon P can be computed in $O(E)$ time where E is the size of the visibility graph of P.

References

[Ch] B. Chazelle, ***Triangulating a simple polygon in linear time***, Technical Report No. CS-TR-264-90, Department of Computer Science, Princeton University, may 1990.

[ET] H. ElGindy and G.T. Toussaint, ***On geodesic properties of polygons relevant to linear time triangulation***, The Visual Computer **5** (1989) 68-74.

[H] J. Hershberger, ***An optimal visibility graph algorithm for triangulated simple polygons***, Algorithmica 4 (1989) 141-155.

[LP] D.T.Lee and F.Preparata, ***An optimal algorithm for finding the kernel of a polygon***, Journal of ACM **26** (1979) 415-421.

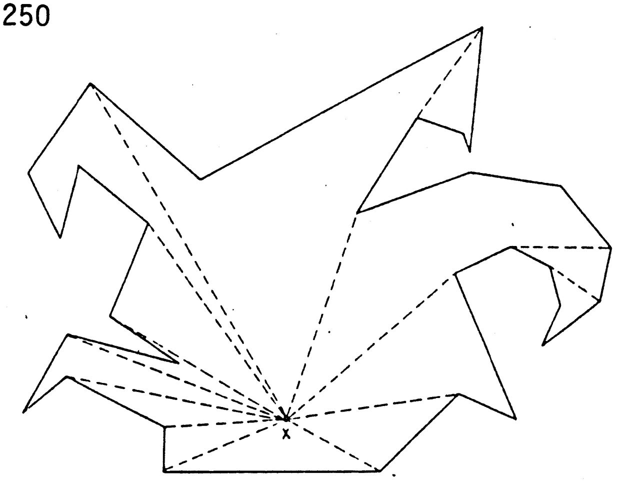

FIGURE 1.

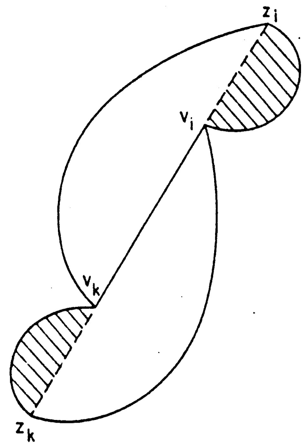

FIGURE 2.

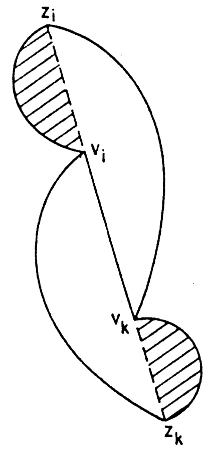

FIGURE 3.

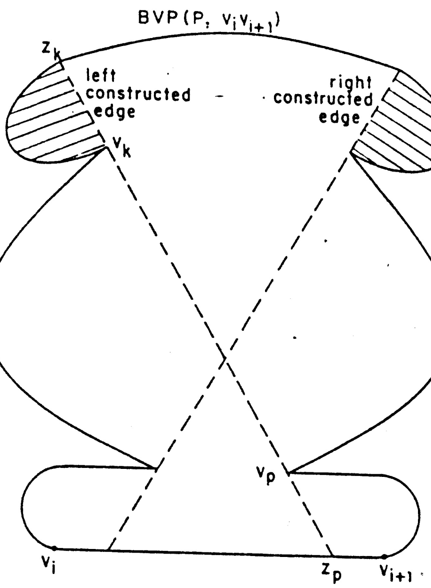

FIGURE 4

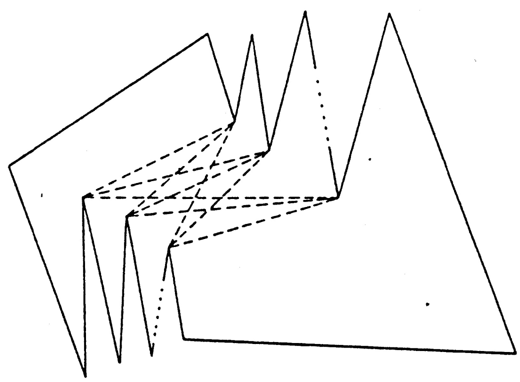

FIGURE 5.

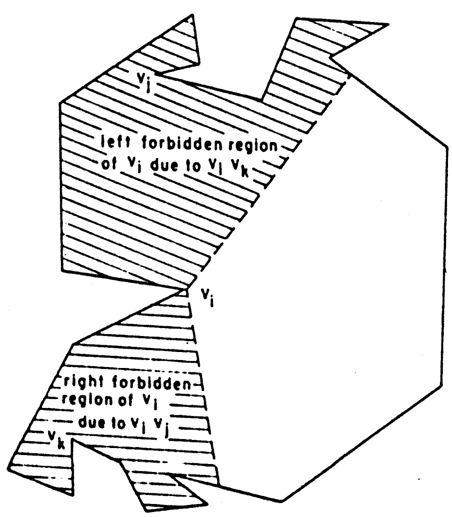

FIGURE 6

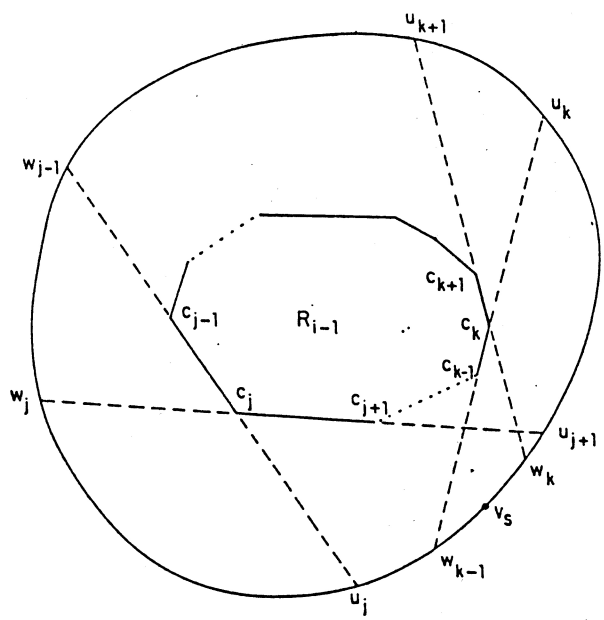

FIGURE 7.

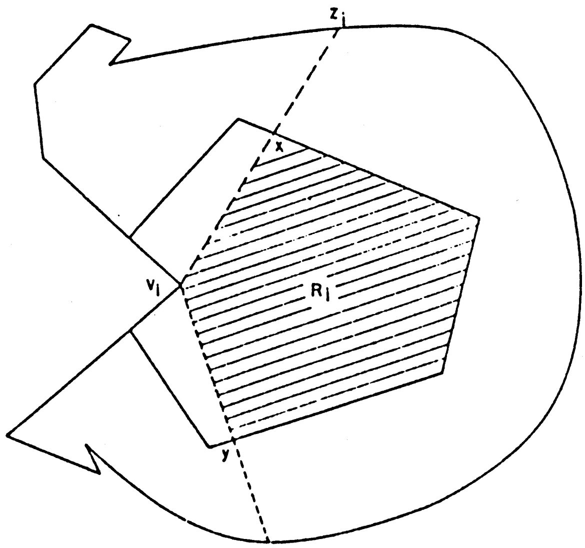

FIGURE 8.

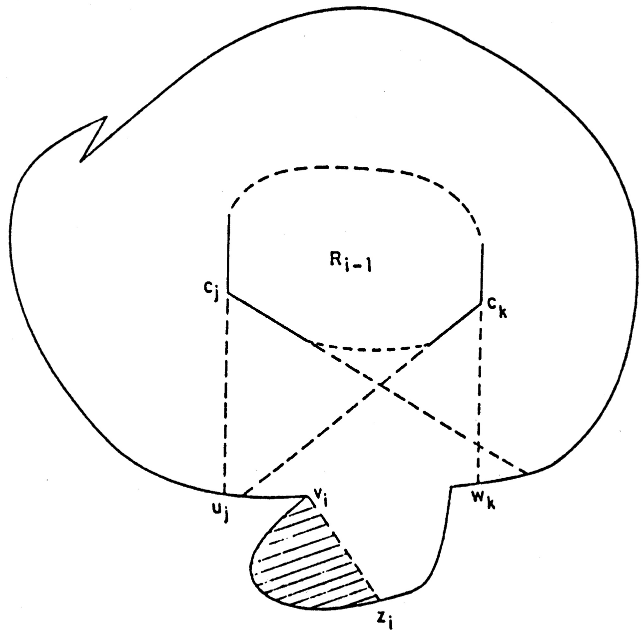

FIGURE 9.

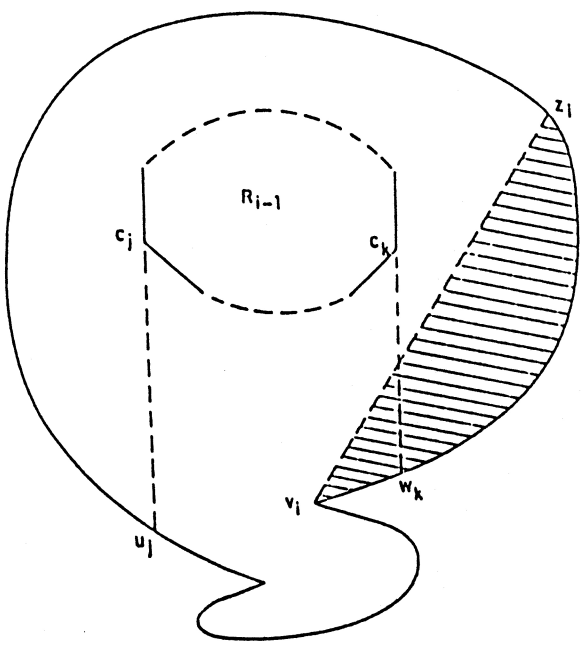

FIGURE 10.

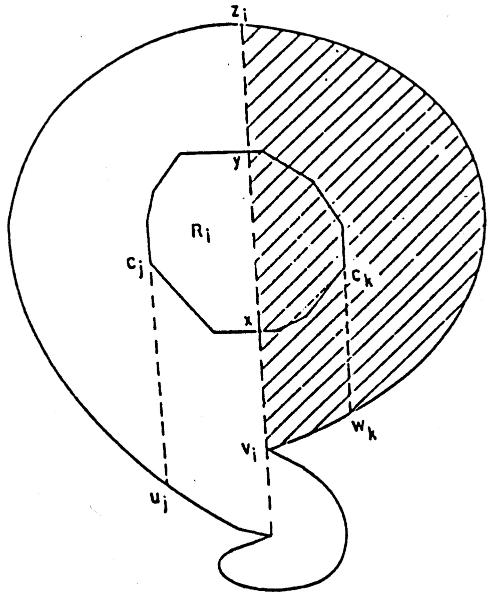

FIGURE 11.