AN O(N LOG N) ALGORITHM FOR COMPUTING THE LINK CENTER OF A SIMPLE POLYGON

Hristo N. Djidjev, Andrzej Lingas, and
J.-R. Sack

SCS-TR-148, JULY 1988
REVISED FULL VERSION SEPTEMBER 1990

Published in Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, Germany, February 1989, Lecture Notes in Computer Science 349, Springer Verlag, pp. 96-107.

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

AN O(N LOG N) ALGORITHM FOR COMPUTING THE LINK CENTER OF A SIMPLE POLYGON

by

Hristo N. Djidjev[1], Andrzej Lingas[2],
and
Jörg-Rüdiger Sack[3]

ABSTRACT

We present an algorithm that determines, in $O(n \log n)$ time, the link center of a simple n-vertex polygon P. As a consequence, we also obtain an $O(n \log n)$-time solution to the problem of determining the link radius of P. Both results are improvements over the $O(n^2)$ bounds previously established for these problems.

1. Introduction

We commence by introducing some notation which will be relevant for the discussion of the link center problem. Then we will briefly describe several results previously obtained for related problems.

A *link path* between a pair of points v and w in a simple polygon P is a polygonal line inside P connecting v and w. The *link distance* $d_L(v,w)$ between v and w is the minimum number of segments required to connect v to w, over all link paths. A path with minimum value $d_L(v,w)$ is called a *minimum link path* between v and w. For comparison, the *geodesic distance* is the minimum (Euclidean) length of a link path between the points and a *minimum geodesic path* is the shortest path connecting the points. Figure 1.1 illustrates a case where a minimum link path and the minimum geodesic path are different. The *link center* (respectively the *geodesic link center*) is the set of all points c in P for which the maximum link distance (respectively the maximum geodesic distance) between c and any point in P is minimized. See Figure 1.2 for an illustration. The *link radius* of P is the maximum link distance from a point in the link center to any vertex of P. The *link diameter* of P is the maximum link distance between any two vertices of P.

---

1 Center of Informatics and Computer Technology, Bulgarian Academy of Sciences, Acad. G. Bonchev str. bl. 25-A, Sofia 1113, Bulgaria

2 Department of Computer Science, Lund University, Box 118, S-22100 Lund, Sweden

3 School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

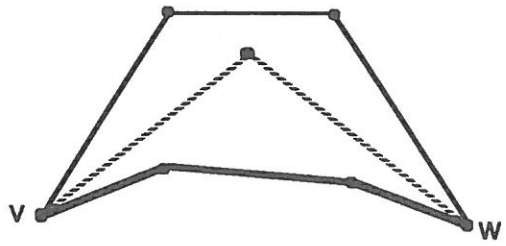

Figure 1.1: Illustrating a minimum link path (shaded) and the distinct minimum geodesic path (bold) connecting the vertices v and w.

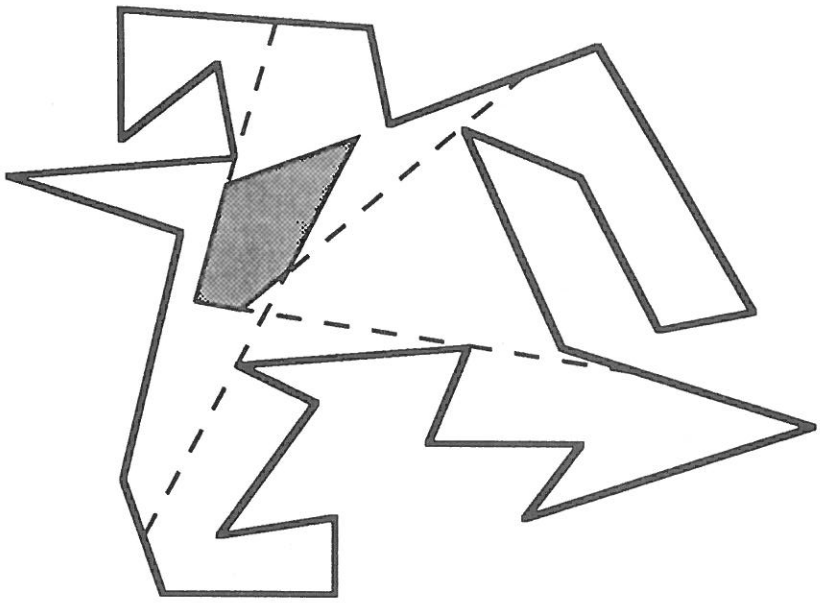

Figure 1.2: Illustrating a polygon P and its link center.

Solving minimum link distance problems seems to be more difficult than solving the corresponding problems using the geodesic distance measure. The difficulties stem from the fact that several minimum link paths may exist connecting a given pair of vertices, while the minimum geodesic path is always unique. As a result of intensive research on link distance problems and on the related visibility problems (see e.g. [GHLST], [SS], [Su87b]), several new upper bounds have recently been established. In particular, a linear-time algorithm for computing the link distance between any pair of points in a triangulated polygon [Su86] and an $O(n \log n)$-time algorithm for calculating the link diameter of a simple n-vertex polygon [Su87b] have been developed.

However, as yet, no efficient algorithm for the problem of finding the link center of a simple polygon has been designed. The currently known best algorithm for finding the link center has a

worst-case time complexity of $O(n^2)$ [LPSSSSTWY]. In their paper, Lenhart *et al.* pose the problem of finding the link center (or at least one point inside) in subquadratic time. The corresponding problem for finding the geodesic center has been solved in $O(n \log n)$ time in [PRS].

The link center problem has several applications, for instance in locating a transmitter so that the maximum number of retransmissions needed to reach any point in P is minimized, or in choosing the best location for a mobile unit minimizing the minimum number of turns needed to reach any point in a polygonal region P.

In this paper, we present an $O(n \log n)$ algorithm that finds the link center of any n-vertex simple polygon P. A preliminary version of this algorithm has been presented in [DLS]. An alternate method has been presented in [K]. As a consequence, we obtain also an $O(n \log n)$-time solution to the problem of determining the link radius of P (improvement over the $\Theta(n^2)$ worst-case time algorithm [LPSSSSTWY]). We derive some geometrical properties related to visibility inside a simple polygon in order to keep the amount of data that need to be repeatedly processed small.

The paper is organized as follows. In Section 2, we give some definitions and recall results related to visibility and to link distance as relevant to this paper. In Section 3, we present an $O(n \log n)$-time algorithm which determines a diagonal d of the polygon P in whose vicinity the link center is located. In Sections 4, we characterize regions whose intersection produces the link center and we describe the global structure of the algorithm. In Section 5, we show how to produce the regions efficiently based on the knowledge of a diagonal d. In Section 6, we describe the efficient computation of the link center as intersection of these regions. Section 7 gives some extensions and further applications of our techniques.

2. Preliminaries

If not otherwise mentioned, a polygon is given as a list of its vertices specified in counterclockwise order and is assumed to be simple with no three collinear vertices. Let P be an n-vertex polygon. The relation between the link radius and the link diameter of P is given by the following theorem.

Theorem 2.1: [LPSSSSTWY]. For any polygon P, the link radius R of P is either $\lceil D/2 \rceil$, or $\lceil D/2 \rceil+1$, where D is the link diameter of P.

The link diameter can be found in $O(n \log n)$ time using the technique derived in [Su87b]. Theorem 2.1 implies that any $O(n \log n)$ algorithm for constructing the link center which is based on knowledge of the *exact* value of R, can be used to find R in $O(n \log n)$ time [LPSSSSTWY]. In order to do this, apply the algorithm on P with $R = \lceil D/2 \rceil$. If the algorithm produces a non-empty link center, then R is equal to $\lceil D/2 \rceil$; otherwise, by Theorem 2.1, R equals $\lceil D/2 \rceil+1$.

We denote by segment(a, b) (or for short (a, b)), the segment inside P joining a and b. Let q be any point in P. The *visibility region* of q in P consists of the set of all points z in P which are *visible from* q, i.e. for which the segment (q,z) is contained in P. The boundary of the visibility region of q defines a polygon within P called the *visibility polygon* $Vis(q,P)$ of q. For a given segment s and a point q in P, q is said to be *visible* from s if there exists at least one point z of s such that q is visible from z. The *visibility region* of a segment s in P consists of all points q in P visible from s; the boundary of the visibility region forms a polygon called *visibility polygon* $Vis(s,P)$. As a generalization we define $VisPol(d, i, X)$ to be the polygon containing all points of a polygon X at link distance i from d. Unless otherwise stated, $VisPol(d, i)$ stands for $VisPol(d, i, P)$.

We use the following results obtained for problems related to visibility inside a triangulated simple n-vertex polygon P:

1) For any segment s in P, $Vis(s,P)$ can be computed in O(n) time [GHLST, T];

2) Given a fixed edge e of P, P can be preprocessed in O(n) time so that, given any point x inside P, the subsegment of e of the points visible from x can be constructed in O(log n) time [GHLST, S].

3) A triangulated polygon P can be preprocessed in O(n) time so that, given any point x inside P and any direction u, the first point on the boundary of P hit by the ray in direction u from x can be computed in O(log n) time [GHLST]. Such an operation is called a *shooting query* and the ray is called the *shot*.

3. Determining a region containing the link center

In this section, we show how to identify, in O(n log n) time, a "small" region within a given polygon P which contains the link center. Knowledge about this allows us to find the link center efficiently.

Let P be a simple triangulated polygon and let d be any edge of the triangulation. For the remainder of this section, we denote by $P_1(d)$ and $P_2(d)$ the polygons defined by the partitioning of P induced by d. Let s be any segment in P. The *covering radius* $CovRad(s,P)$ of s is defined by

$$CovRad(s, P) = \max_{p \in P} \min_{q \in s} d_L(q,p).$$

Let d be a diagonal minimizing the number $|CovRad(d,P_1(d))-CovRad(d,P_2(d))|$, called the *covering difference*, where $P_1(d)$ and $P_2(d)$ are the two subpolygons of P determined by d. Ties are broken arbitrarily.

The *covering radius set*,CR (d), of d is the set containing both covering radii of d. Since the covering radii to either side may be equal, CR (d) is, in general, a multiset. (When no ambiguity arises we use P_1 , P_2 and CR instead of P_1 (d), P_2 (d), and $CR(d)$, respectively.)

We first describe the location of the link center with respect to an arbitrary diagonal of P.

Lemma 3.1: Let d' be a diagonal of P whose covering radius set is equal to $\{c_1, c_2\}$, $c_1 \geq c_2$. Then $VisPol(d', R - c_2 + 1, P)$ contains the link center of P.

Proof: Let d' be the diagonal with covering radius set $\{c_1, c_2\}$, $c_1 \geq c_2$. Let P_1, P_2 be the subpolygons induced by d' such that $\text{CovRad}(d', P_i) = c_i$, $i = 1,2$. Assume that q is a point of the link center of P that does not belong to $VisPol(d', R - c_2 + 1, P)$. If $c_1 > c_2$ then P_2 will contain no point of the link center. Therefore w.l.o.g. assume that P_1 is that polygon among P_1 and P_2 which contains q. Since $CovRad(d', P_2) = c_2$, there exists a point v of P_2 such that $d_L(z,v) \geq c_2$ for each z on d'. Now construct a minimum link path $p=(p_1, \ldots, p_k)$ from $q=p_1$ to $v=p_k$. Since q is in P_1 the path p intersects d', say at point z'. Furthermore, q is not in $VisPol(d', R - c_2 + 1, P)$ and thus z' is on segment (p_i, p_{i+1}), for $i > R - c_2 + 1$. From $d_L(z', v) \geq c_2$ it follows that $k \geq R+1$. Hence q does not belong to the link center of P. □

We will show that a diagonal d of P can be found so that the link distance from d to any point of the link center is no more than two.

Lemma 3.2: The minimal covering difference over all edges of a given triangulation is at most one.

Proof. Assume by contradiction that the covering difference for all edges of the triangulation of the simple polygon P is at least 2. Let Δ be a triangle of the triangulation containing at least one point, p, of the link center, denoted by LC. Denote the covering radii of the edges of Δ as per Figure 3.1. Since p is in LC it follows that a_1, b_1 and c_1 are no larger than R. We refer to this observation as (+). Since R is the link radius at least one of these covering radii is at least R-1. Assume that $c_1 = \max\{a_1, b_1, c_1\}$. Two cases arise: (1) $c_1 = R$-1 and (2) $c_1 = R$.

(1) Assume that $c_1 = R$-1. By the initial assumption, $c_2 \geq R + 1$ and since p is in LC it follows that $c_2 = R + 1$. (The case where $c_2 \leq R - 3$ is handled analogously to case (2) below.) Consequently, a_1 or b_1 is at least R which contradicts that $c_1 = \max\{a_1, b_1, c_1\}$. Therefore, by the initial assumption, b_2 is no larger than R -2. (Since p is in LC then $b_2 \leq R + 1$). Thus $c_1 \leq R - 2$ which contradicts the assumption that $c_1 = R$-1.

(2) Assume now that $c_1 = R$. Then, by the initial assumption, $c_2 \leq R - 2$. By Lemma 13 in [LPSSSSTWY] there exists a point in LC which is in the subpolygon, called P_c, induced by c not containing Δ. Then apply the entire argument to triangles in P_c which will, inductively, lead to a contradiction Subsequent triangles examined will all be in P_c; in particular Δ will never be examined again. □

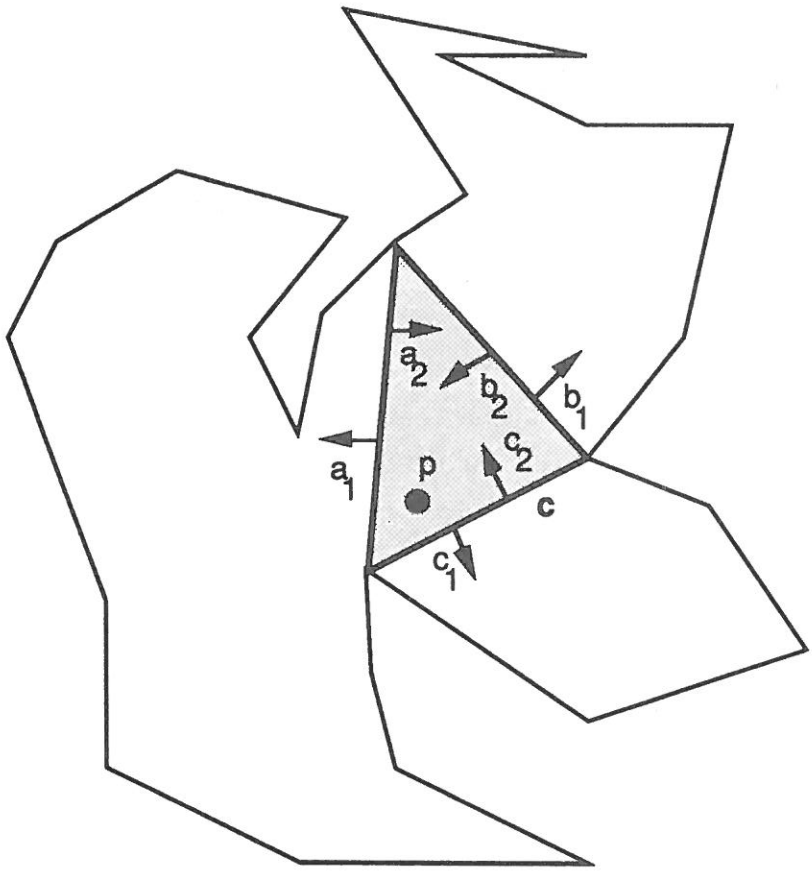

Figure 3.1: Illustrating the notation of Lemma 3.2.

Lemma 3.3. Let d be an edge of the triangulation with covering radius set $\{c_1,c_2\}$ minimizing the value $|c_1 - c_2|$. Then $\{c_1,c_2\} \in \{\{R-1, R-1\}, \{R, R-1\}, \{R, R\}, \{R+1, R\}\}$.

Proof. Let d be an edge of the given triangulation of P with covering radius set, CR, equal to $\{c_1,c_2\}$, and minimizing the value $|c_1 - c_2|$. Then by Lemma 3.2 $|c_1 - c_2| \leq 1$. Since R is the link radius $CR \neq \{R'_1, R'_2\}$ for any values for which either ($R'_1 > R$ and $R'_2 > R$), or ($R'_1 \leq R-2$ and $R'_2 \leq R-2$). Therefore it will be enough to show that $CR \neq \{R-1, R-2\}$. Although a triangulation edge d may exist with $CR = \{R-1, R-2\}$, we will show that d could not minimize the value $|c_1 - c_2|$. Assume the contrary, i.e. let the covering difference of any edge of the triangulation be at least one and let d be a triangulation edge with covering radius set $\{R-1, R-2\}$. For any triangulation edge e let $P(e)$ be the subpolygon of P determined by e with greater covering radius from e. Let d' be a triangulation edge with covering radius set $\{R-1, R-2\}$ such that no other edge d'' with covering radius set equal to $\{R-1, R-2\}$ and $P(d'') \subset P(d')$ exists.

Assume for ease of notation that d' is vertical. W.l.o.g. let the covering radii to the left be $R-2$ and to the right be $R-1$. Let the triangle adjacent to d' and to the right of d' has edges d', d_r and d_b. For one of d_r or d_b, say d_r, the covering radius set is $\{R_1, R_2\}$ with $R_i \in \{R-1, R-2\}$, $i=1,2$, and, since d' minimizes the value $|c_1 - c_2|$, then $R_1 \neq R_2$. Let R_1 be the covering radius of d_r with respect to the subpolygon induced by d_r that contains d' and R_2 be the other covering radius. In case R_1 equals $R-2$ the argument is repeated by replacing d' by d_r. Consider now the second

case, $R_1 = R\text{-}1$ and $R_2 = R\text{-}2$. Then d_b can not be an edge of the polygon, otherwise, any point in the triangle would reach all other points in at most R-1 links which contradicts the definition of R. Thus d_b is an edge of the triangulation and has covering radius set $\{R^*, R\text{-}1\}$, where R^* is either R-1 or R-2. In the former case we obtain a contradiction to the fact that the covering difference of any edge is at least one; the latter case contradicts to the choice of d' (we should have chosen d_b instead of d'). □

Lemma 3.4: Any triangulation of a simple polygon P contains an edge d so that VisPol(d, 2, P) contains the link center of P.

Proof. By Lemma 3.3 there exists an edge d of the triangulation of P whose covering radius set $\{c_1 , c_2 \}$ is such that the minimal covering radius is R-1. By Lemma 3.1 VisPol(d, R -(R -1)+1, P) = VisPol(d, 2, P) contains the link center. □

In Theorem 3.6, we will show how such a diagonal d of P can be determined in O(n log n) time.

Construct the dual graph T of the triangulation of P by defining a vertex of T for each triangle of the triangulation of P and joining a pair of vertices if, and only if, their corresponding triangles share an edge of the triangulation of P. T is a binary tree and a one-to-one correspondence exists between the edges of T and the internal edges of the triangulation of P. (See [Ch], [GHLST] and [LPSSSSTWY] for other algorithms based on this idea.)

We use the following well known fact stated e.g. in [LT,D].

Lemma 3.5: For any binary tree T with n vertices there exists an edge of T whose removal divides T into two subtrees of no more than $2n/3+1$ vertices each. Such an edge can be found in $O(n)$ time.

Theorem 3.6: For any n-vertex simple polygon P one can identify a diagonal d for which the visibility region *VisPol*(d, 2) contains the link center in O(n log n) time.

Proof. We show that Algorithm 3.1 stated below finds a diagonal d as in Lemma 3.4 in O(n log n) time. Let T be the tree corresponding to the triangulation of P. We perform Algorithm 3.1 initially letting T' equal T. Since the covering radii of an edge d^* of a triangulation can be determined in linear time [Su87b], each of the O(log n) iterations of the algorithm takes O(n) time. This yields an O(n log n)-time bound on the running time of Algorithm 3.1.

Every time the algorithm is called, i.e., $c_1 \geq c_2$ it follows from Lemma 3.3 that $c_1 \geq c_2 + 1$ otherwise, a required diagonal would have been found. If $c_1 = c_2+1$ then, as established in the proof of Lemma 3.3, the subpolygon (induced by the diagonal d^*) with the larger covering radius from d^* contains a diagonal which meets the requirements of the lemma. The algorithm recursively examines this polygon. Analogously, for the case that $c_1 > c_2 + 1$. Using Lemma 3.5 the correctness of the algorithm follows. □

Algorithm 3.1 Finding a diagonal d^* having the properties stated in Lemma 3.3

Input: A simple polygon and its triangulation T'

Output: A diagonal d^* having the properties stated in Lemma 3.3

If the triangulation T' contains a single edge

then output the diagonal corresponding to this edge and **stop**

else find an edge d^* of the triangulation of P whose corresponding edge in T' divides T' into two subtrees T_1 and T_2 each containing no more than $2|V(T')|/3$ vertices, where $V(T')$ denotes the vertex set of T' and $|V(T')|$ denotes its cardinality.

Compute the covering radii c_1, c_2 of d^*; $c_1 \geq c_2$.

Let T_1 be the subtree corresponding to that subpolygon induced by d^* whose covering radius is c_1 and let T_2 be the other subtree of T'.

If $(c_1, c_2) \in \{(R-1, R-1), (R-1, R), (R, R), (R+1, R)\}$ **then stop**

else apply the algorithm recursively for $T' = T_1$.

4. Global structure of algorithm

We introduce a set of segments inducing partitions P which can be identified in $O(n \log n)$ time. For each of these segments we choose one of the subpolygons of P determined by it so that the intersection of all subpolygons yields the link center. In this section we properly define these segments and outline an algorithm to determine the segments efficiently. The details of the algorithm are then discussed in Section 5.

Suppose that we have found a diagonal d of P as determined in Theorem 3.6. The polygons $VisPol(d, 1)$ and $VisPol(d, 2)$ can be determined in $O(n)$ time. On the boundary of $VisPol(d, 1)$, several segments, called *lids* of P, may exist that do not belong to P. Each lid divides P into two subpolygons of which the one not containing $VisPol(d, 1)$ is called a *pocket* of P.

In each of these pockets we recursively compute the polygon visible from the respective lid. Now associate with each of these visibility polygons a node of a tree W, where two nodes are connected by an arc if, and only if, their associated polygons share a lid. We define the node corresponding to $VisPol(d,1)$ to be the root of W. The tree W has been referred to by Suri [Su87b] as the *window tree* of d; it can be constructed in linear time given a triangulation of the polygon. To simplify the notation we identify the node of W representing a polygon by the polygon itself. The polygons associated with nodes of W will be called *regions* of P. We restate some notation and two theorems from [LPSSSSTWY]. The *k-neighborhood* about a point z in P is defined by

$$N_k(z) = \{z' \in P \mid d_L(z, z') \leq k\}.$$

Fact 4.1: [LPSSSSTWY] Let $k \geq 1$ and let v be a point of P. The $N_k(v)$ region is a subpolygon of P all of whose vertices lie on the boundary of P and each of whose edges is one of the following two types:

(1) a (portion of an) edge of P;

(2) a segment (r, z) connecting a reflex vertex r of P to a point z on the boundary of P.

Fact 4.2: [LPSSSSTWY] The link center of a polygon P is the intersection of the sets $N_R(v)$ over all convex vertices v of P, where R is the link radius of P.

It follows from Facts 4.1 and 4.2 that for any reflex vertex r there are at most two such segments(r, z) from the boundary of $N_i(v)$ referred to as *counterclockwise N_i-boundary segment* of r if the vertex r appears after z on the boundary of the corresponding N_i region counterclockwise, or a *clockwise N_i-boundary segment* of r, otherwise.

The following definitions are illustrated in Figure 4.1. Let s be a boundary segment from some $N_j(v)$ region, $j \geq 1$, partitioning P into two subpolygons; the subpolygon containing vertex v is referred to as $P(s, v)$, or simply $P(s)$ if no ambiguity arises.

Let v be a convex vertex of P at distance k from d (as determined in Theorem 3.6)An edge s on the boundary of $N_i(v)$ is called an *i-boundary segment*, for short an *i-BS*, (for v) if

(a) $i < k-1$ and d and v lie in different subpolygons of P induced by s
(the link center is far away from v in the subpolygon induced by s and d), or

(b) i=k-1 and s belongs to $VisPol(d,2)$
(since the link center is in $VisPol(d,2)$, s could be an edge of the link center), or

(c) $i > k$ -1 and s belongs to the boundary of $VisPol(s', i-k+1, P-P(s'))$, for some $(k-1)$-BS s', where $s \neq s'$ and s does not belong to the boundary of P.
(v and s are in different subpolygon induced by d; if s is already far away from the link center it could be ignored. This would however not improve the time overall complexity of the algorithm.)

If s is an i-BS for v then $P(s, v)$ is called the *i-BS polygon* for v induced by s. It is clear from the above definitions that $P(s, v) \supseteq N_i(v)$. We refer to the i-BS's, for the same i, as BS's of the same *generation.*

If for some v and i there is a single i-BS for v the i-BS will be denoted by $b_i(v)$ and the i-BS polygon is denoted by $P(b_i(v))$. To simplify the notations we define $b_0(v) = v$ and $P(v) = v$ for any vertex v.

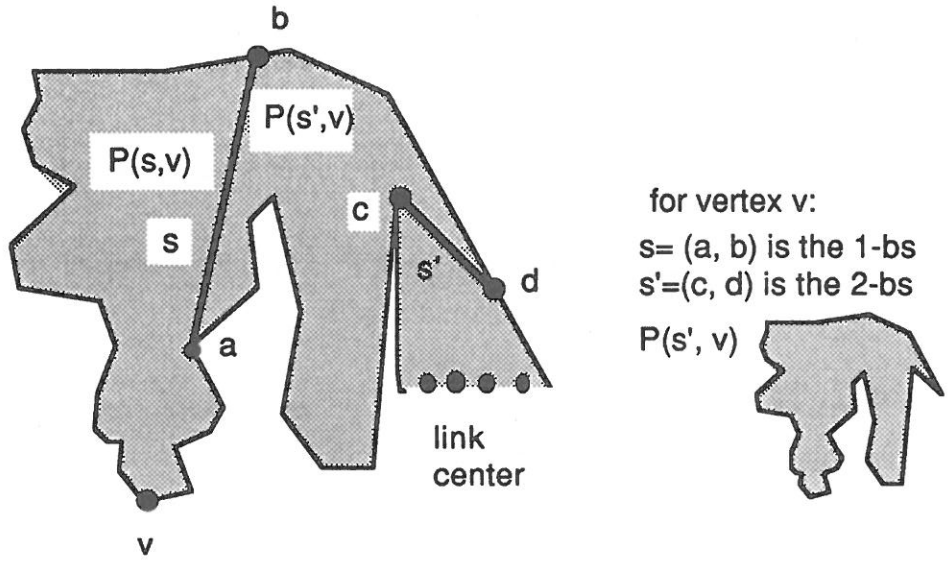

Figure 4.1: Illustrating the definitions of i-BS for i=1,2 and the the corresponding i-neighbourhood polygons.

From the definitions of link center and N_k-region it follows:

Corollary 4.3. Let k be an integer in [0, R]. A point z belongs to the link center of a polygon P if and only if the link distance between z and all polygons $N_k(v)$ is $\leq R - k$, for all convex vertices v.

The gain in efficiency of our algorithm compared to the $O(n^2)$ algorithm of [LPSSSTWY] results from the fact that instead of using the neighborhood regions $N_k(v)$ to compute the link center we use the k-BS polygons $P(b_k(v))$. Theorem 4.5 stated below will justify this approach.

First we argue that certain BS's are irrelevant for the computation of the link center. Assume that two k-BS's of the same generation are incident to the same reflex vertex r. Then one of the corresponding k-BS polygons will contain the other and it can be removed from further consideration (Fact 4.2 and the definition of i-BS). More generally, for any two segments of the same generation for which one of the corresponding polygons contains the other, the same argument holds, i.e. one can be removed. Assume now that two BS's are incident to a common reflex vertex, but they are from different generations. Similarly, the lower generation BS can be removed since the polygon corresponding to its BS successor will always contain that corresponding to the higher generation BS. Any BS that is not removed is called *relevant* and, as shown in Lemma 4.4, only relevant BS's need to be considered.

Lemma 4.4. Let k be an integer in [0, R]. A point z belongs to the link center of a polygon P if and only if the link distance between z and the polygons $P(s)$ is $\leq R - k$, for any relevant k-BS s.

Proof. "=>" Follows from the above Corollary 4.3 and the fact that $P(b_i(v)) \supseteq N_i(v)$.

"<=" The result is established by induction on k.

Let $k = 0$.

Since the link distance between a point z in the link center and any vertex v in P is at most R the results follows in this case by the definition of $P(b_0(v))$ to be v.

Assume for the induction that for some k, $0 \le k < R$, if the link distance between some point z^* and the polygon $P(s)$ is $\le R - k$ for any relevant k-BS s, then z^* belongs to the link center of P.

Assume now that the link distance between some point z and the polygon $P(s)$ is $\le R - k - 1$ for any relevant $(k+1)$-BS s. Consider an arbitrary relevant k-BS s'. By the definition of a BS and a relevant BS there exists a relevant $(k+1)$-BS s'' such that

$$\max_{z'' \in P(s'')} \min_{z' \in P(s')} d_L(z', z'') \le 1 .$$

Then by the last assumption the link distance between z and the polygon $P(s')$ is $\le R - k$. As s' was arbitrary, then the link distance between z and the polygon $P(s)$ is $\le R - k$ for any relevant k-BS s. Then from the inductive assumption z belongs to the link center. □

From the above it follows:

Theorem 4.5. The link center of a polygon P is the intersection of $VisPol(d,2)$ with the sets $P(b_R(v))$ over relevant $b_R(v)$ for all convex vertices v of P, where R is the link radius of P.

We have shown how to find a diagonal d whose $VisPol(d, 2)$ contains the link center. We outline the algorithm to compute the link center of P in $O(n \log n)$ time. The steps (as indicated) will be discussed in detail in the remaining section.

Algorithm 4.1. Outline of the algorithm for finding the link center

Input: A triangulated simple polygon P

Output: The link center of P

1. Find the link diameter D by the algorithm from [Su87b]. Choose $R = \lceil D/2 \rceil$ (the smaller of the two possible values given by Theorem 2.1).
2. Find a diagonal d as per Theorem 3.6.
3. Construct the window tree of P with respect to d in $O(n)$ time [Su87b].
4. Produce a post-order numbering [see e.g. AHU] of the vertices of the window tree.
   Let $R_1,\ldots,R_j$ be the corresponding ordering of the regions of P corresponding to the numbering of the vertices of the window tree. R_j is the visibility polygon from d. (Notice that according to this ordering all children of a region R_i will precede R_i in the region sequence. See Figure 4.1.)
5. For $i = 1,\ldots,j-1, j, j-1,\ldots,1$ find by Algorithm 5.1 below all relevant BS's incident with reflex vertices in R_i, whose boundary polygons do not intersect R_j.
6. Find the intersection of the boundary polygons determined by the relevant R-BS's using the algorithm from Algorithm 6.1.
7. If the intersection region computed in Step 6 is empty then repeat Steps 2-6 choosing $R = \lceil D/2 \rceil + 1$ (the other possible value of R)

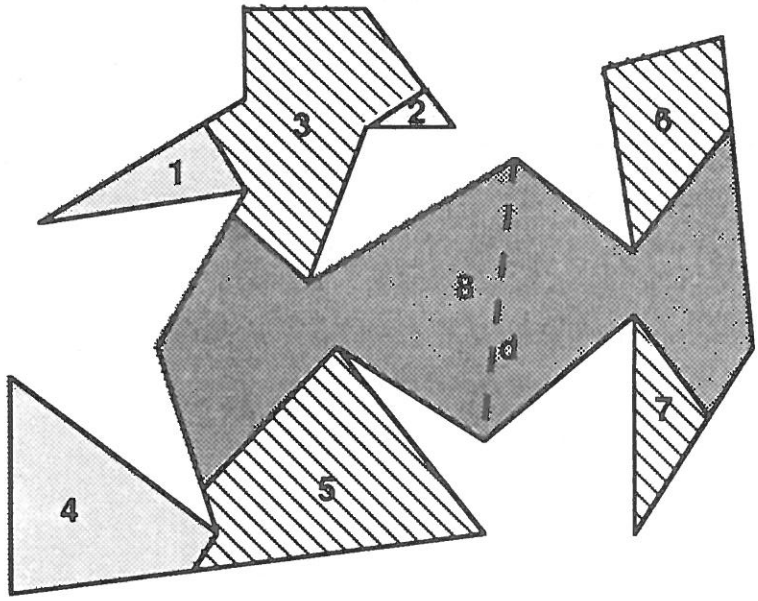

Figure 4.1: A possible ordering of the regions for Step 4.

5. Determining *i*-boundary segments efficiently

Consider a particular region U of the window tree with lid l and the children polygons of U; call the entire polygon so formed U^+. U^+ is that portion of the $VisPol(l, 2, P)$ that lies on the same side of l as does U. Assume that all children of the node corresponding to U have been processed (inductively). Let i-1 be the maximum index on BS's produced so far that intersects U^+ (see Algorithm 4.1). It is easy to notice that any BS in U^+ of generation i-4 or below can be omitted from consideration. We assume first that all relevant BS's in U^+ are relevant (i-1)-BS's leaving the generalization to (i-2), (i-3)-BS's to the end of this section.

We will describe an algorithm to compute all relevant i-BS's of type (a) (look at the definition of i-BS) incident to reflex vertices of U. The algorithm is shown to require $O(|U^+| \log |U^+|)$ time. Each reflex vertex in P is examined O(1) times. Thus the total time taken to find all i-boundary segments (counterclockwise and clockwise) for each reflex vertex in P is bounded by $O(n \log n)$. Without loss of generality we discuss only the determination of counterclockwise i-boundary segments (see Figure 4.1), simply referred to here as *i-boundary segments.*

For any reflex vertex r in U there might be as many as $|U^+|$ relevant (i-1)-BS's that are visible from r. We first show that at most one relevant (i-1)-BS needs to be considered to compute the relevant i-BS incident to r (if any) (Lemma 5.2). Given this unique (i-1)-BS, in Lemma 5.2, we show how to determine the relevant i-BS incident to r. A given (i-1)-BS can produce relevant i-

BS 's incident to several reflex vertices in U. The efficient computation of all such *i-BS* 's is described in Algorithms 5.1- 5.3.

5.1 Determining the unique $(i\text{-}1)BS$ that generates a relevant i-BS and determining its extremal segment

Let $s_l = (t_l,h_l)$ be a relevant i -BS. Then t_l, is called the tail of s_l and h_l is called its head. (By definition of i-BS the interior of the corresponding N_i (v) polygon is to the right of s.) Assume that the (i-1)-*BS*'s of U^+ are *lexicographically* sorted, i.e. for $s_l = (t_l, h_l)$ and $s_j = (t_j, h_j)$ s_l is lexicographically less than s_j if h_l occurs before h_j on a counterclockwise traversal of U^+ starting at some tail. Notice that as the BS's are relevant, it is not possible that $s_l \neq s_j$ and $h_l = h_j$. Produce a sorted cyclic list.

The lexicographically first (i-1)-*BS* for a given reflex vertex r, where r belongs to the clockwise polygonal chain from the tail to the head of the (i-1)-*BS*, is called the *first* (i-1)-*BS for r*. (See Figure 5.1 for an illustration.) We say that a relevant (i-1)-BS, $b_{i-1}(v)$, *contributes* to the i^{th} generation *at* reflex vertex r if a relevant i-BS, b_i (v), is incident to r.

Let $\mathcal{R}_k$ be the set of all reflex vertices for which a segment s_k is a first (i-1)-BS. Is is easy to see that the following Observation 5.1 holds.

Observation 5.1: All sets $\mathcal{R}_k$ of those reflex vertices in U for which s_k is the first (i-1)-*BS* for all relevant BS's s_k in U can be determined in O($|U^+|$) time. Furthermore, the sets $\mathcal{R}_k$ lie on non-overlaping chains on the boundary of U.

We mentioned above we assume first that all *BS*'s entering the particular region U of the window tree are *(i-1)-BS*'s for some i>0, i.e. they are from the same generation. For this situation, we next discuss how to compute the i-BS's.The next two lemmas show how the relevant i-BS's can be identified.

Note that, for i =1, (the basis of our iterative construction), the 0-*BS*'s for a vertex v equals v . We can see this as the case of a *BS* which is degenerated to a point.

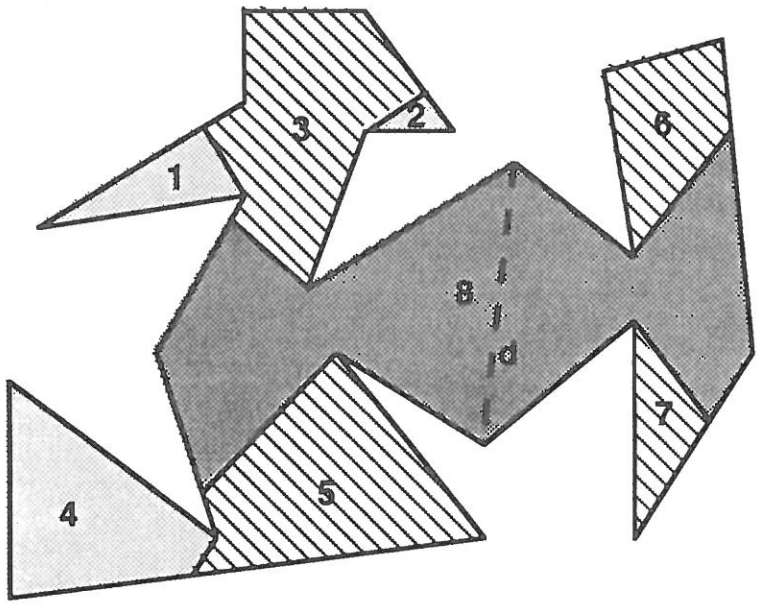

Figure 4.1: A possible ordering of the regions for Step 4.

5. Determining *i*-boundary segments efficiently

Consider a particular region U of the window tree with lid l and the children polygons of U; call the entire polygon so formed U^+. U^+ is that portion of the $VisPol(l, 2, P)$ that lies on the same side of l as does U. Assume that all children of the node corresponding to U have been processed (inductively). Let i-1 be the maximum index on *BS*'s produced so far that intersects U^+ (see Algorithm 4.1). It is easy to notice that any BS in U^+ of generation i-4 or below can be omitted from consideration. We assume first that all relevant *BS*'s in U^+ are relevant (i-1)-*BS*'s leaving the generalization to (i-2), (i-3)-*BS*'s to the end of this section.

We will describe an algorithm to compute all relevant i-BS's of type (a) (look at the definition of i-*BS*) incident to reflex vertices of U. The algorithm is shown to require $O(|U^+| \log |U^+|)$ time. Each reflex vertex in P is examined O(1) times. Thus the total time taken to find all i-boundary segments (counterclockwise and clockwise) for each reflex vertex in P is bounded by $O(n \log n)$. Without loss of generality we discuss only the determination of counterclockwise i-boundary segments (see Figure 4.1), simply referred to here as *i-boundary segments*.

For any reflex vertex r in U there might be as many as $|U^+|$ relevant (i-1)-BS's that are visible from r. We first show that at most one relevant (i-1)-*BS* needs to be considered to compute the relevant *i-BS* incident to r (if any) (Lemma 5.2). Given this unique (i-1)-*BS*, in Lemma 5.2, we show how to determine the relevant *i-BS* incident to r. A given (i-1)-*BS* can produce relevant *i*-

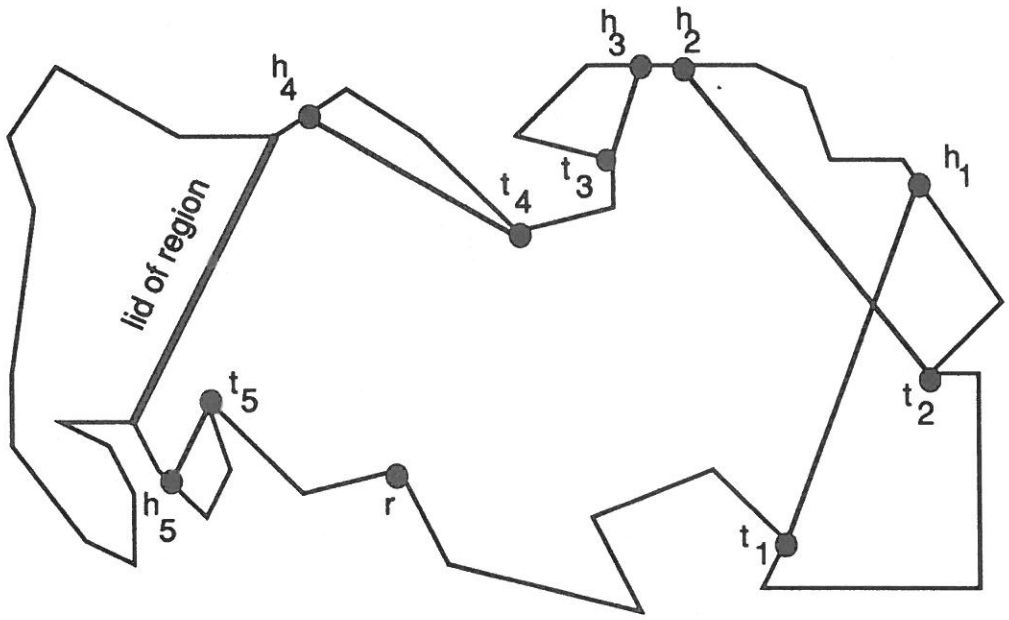

Figure 5.1: A reflex vertex r and segments $s_l = (t_l, h_l)$ lexicographically sorted from r. Segment $s_1 = (t_1, h_1)$ is the relevant i-BS for r.

All polygonal chains mentioned are counterclockwise (unless otherwise stated). We denote by $CH_Q(a, b)$ the polygonal chain from a boundary point a of P to a boundary point b of P. (The index Q is omitted when no ambiguity arises.) *Polygon*(a, b) stands for the chain $CH(a, b)$ closed by adding the segment(b, a).

Lemma 5.2. If a reflex vertex r is incident to a relevant i-BS then r sees its first $(i$-1)-BS.
Proof: Assume that r does not see its first *(i-1)-BS*, s, for vertex v and that the boundary segment which contributes to the i^{th} generation at r is $s'=b_{i-1}(w)$, $w \neq v$. Then r sees s'. Let the relevant i-BS incident to r be $b_i(w)=(r, z'')$. Let $s= (t,h)$ and $s' = (t', h')$. (See Figure 5.2 for an illustration.) Now let p be a point on s' that is visible from r and has minimum distance to h' among all such points. Assume w.l.o.g. that $p \neq h'$. Then there exist points z' and z on $CH(r,h')$ and $CH(h', z'')$ co-linear with r, z'' and visible from r, respectively. Since s is the first $(i$-1)-*BS* for r, then s lies inside the counterclockwise polygon(r, z'). Let $b_i(v)$ be any i-BS of v. We will show now that $P(b_i(w)) \supseteq P(b_i(v))$, which will contradict to the assumption that $b_i(w)$ is a relevant i-BS. Assume that there exists a point $q \in P(b_i(v)) - P(b_i(w))$. Then some segment (q, f) exists interior to P such that $f \in P(b_{i-1}(v))$. The segment (q,f) intersects $b_i(w)$, whence q would lie in *Polygon*(z'',r). Hence, f is in Polygon(z, z'') and therefore f cannot be in $P(b_{i-1}(v))$ which contradicts our assumption. □

Thus both endpoints of s' are on one of these chains:

(a) CH(r, z') which contradicts the assumption that s is the first (i-1)-BS for r or

(b) CH(z' , z) then s' is not visible from r, a contradiction, or

(c) CH(z, r) which contradicts the choice of angle.

The contradiction shows that (r,z'') is the boundary segment for r. □

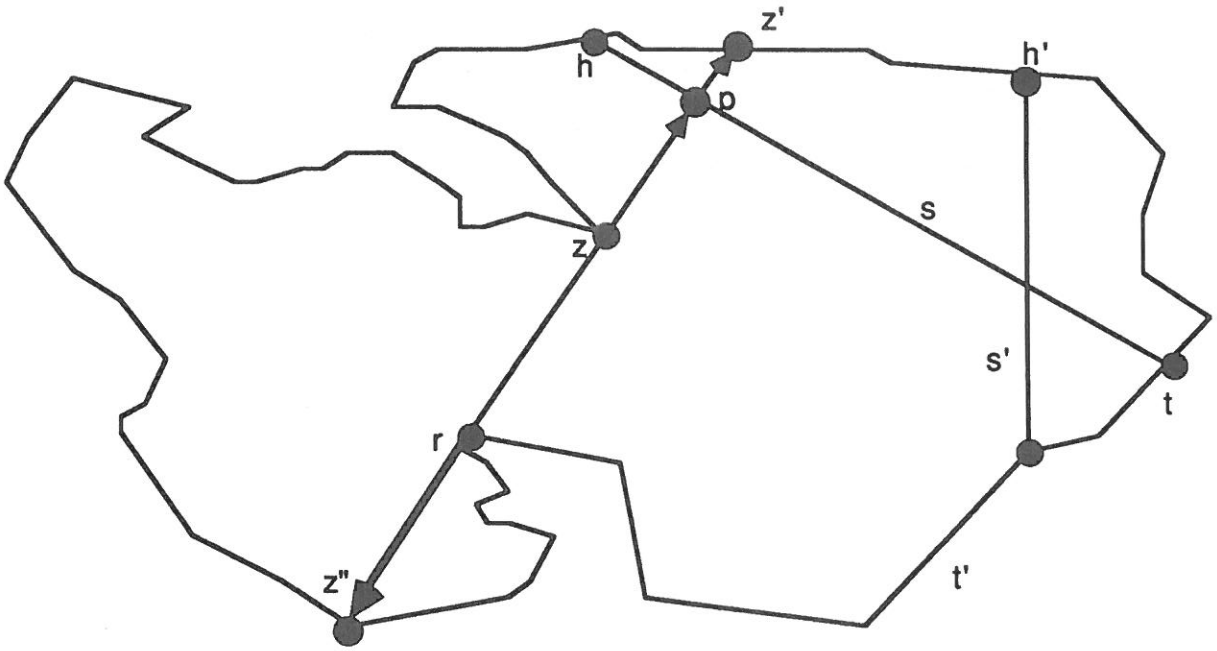

Figure 5.3: Illustrating Lemma 5.3. The back-extension (r, z'') of the extremal segment (r, p).

5.2: Computing relevant boundary segments

Let U be a region of the window tree. We still assume that all BS's necessary for the computation of i-BS for reflex vertices in U are (i -1)-BS's of the region U^+, where U^+ is the polygon formed by U and its children polygons in the window tree. Algorithm 5.1 below gives a sketch of the construction of all relevant i-BS's, which will be used for the determination of the link center as subsequently discussed in Section 6. For ease of description, we assume each region of the window tree is retriangulated separately.

Algorithm 5.1 Construction of the relevant i-BS's

Input: A region U of the window tree; all relevant (i-1)-BS's of U^+ of the window tree, where U^+ is the polygon formed by U and its children polygons

Output: All relevant i-BS's incident to reflex vertices in U generated from the (i-1)-BS's entering U^+

1. sort the (i-1)-BS's lexicographically starting at some tail of a segment;
2. let $\mathcal{S} = s_1, \ldots, s_{last}$ be the sorted list so produced;
3. **for** $s_k := s_1$ **to** s_{last} **do**

   Determine the set $\mathcal{R}_k$ of all reflex vertices in U for which s_k is a relevant (i-1)-BS;

   for each reflex vertex r in $\mathcal{R}_k$ **do**

   compute the relevant i-BS incident to r : {details given below in Algorithms 5.2 and 5.3}

By Observation 5.1 all sets $\mathcal{R}_k$ can be determined in $O(|U^+|)$ time. In this section we address the determination of the relevant i-BS's for all reflex vertices in $\mathcal{R}_k$. By Lemma 5.3 a boundary segment for a reflex vertex r in $\mathcal{R}_k$ is the back-extension of an extremal segment for r and s_k. To efficiently compute the extremal segments for all reflex vertices r in $\mathcal{R}_k$ we construct a polygon S_k in which we compute the extremal segments. The extremal segments computed in S_k are identical to those produced in P, but are more efficiently determined. The construction is as follows.

We will denote by $path(a'', b'')$ the minimum Euclidean length path inside a polygon from a point a'' to a point b''. Let (a, b) be a lid of a region U of the window tree ($(a, b) \neq d$) and let U^+ be U plus its children polygons. Let $r_f, \ldots, r_l$ be the reflex vertices in $\mathcal{R}_k$ listed in (counter clockwise) polygonal order starting at the head of the $(i\text{-}1)$-BS determining $\mathcal{R}_k$.

The polygons S_k are now constructed as hourglasses (slightly generalized) as in [GHLST]. More specifically, S_k is composed of the shortest paths $path(h_k, r_f)$ and $path(r_l, t_k)$, the chain $CH_U{}^+(r_f, r_l)$, and the segment s_k. The segment s_k serves as the base from which the hourglass is constructed. See Algorithm 5.2. for a description and refer to Figure 5.4 for an illustration of the procedure.

In case the shortest paths $path(h_k, r_f)$ and $path(r_l, t_k)$ share one or more edges, one notices (analogously to [GHLST]) that no reflex vertex on the chain $CH_U{}^+(r_f, r_l)$ can see its first segment s_k.

Algorithm 5.2 Construct S_k

Input: All relevant segments s_k together with their relevant reflex vertex sets $\mathcal{R}_k$

Output: All polygons S_k

for each pair $(s_k, \mathcal{R}_k)$ in polygonal order sorted by the tails of s_k **do**

let $r_f, \ldots, r_l$ denote the chain of reflex vertices in $\mathcal{R}_k$;

compute p_1 as $path(h_k, r_f)$

compute p_2 as $path(r_l, t_k)$,

if p_1 and p_2 share one or more edges remove S_k from consideration

polygon S_k is formed by connecting p_1, $CH_U(r_f, r_l)$, p_2, and the segment (t_k, h_k).

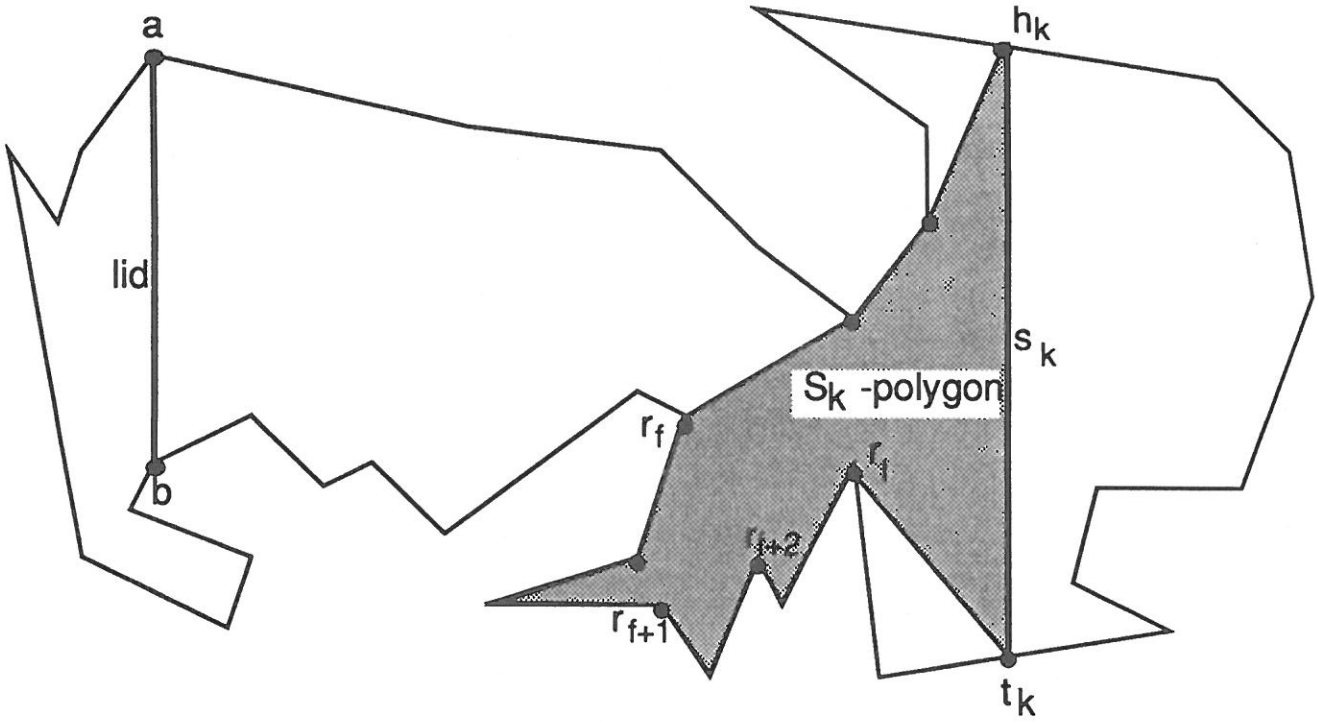

Figure 5.4: Constructing the S_k-polygons for efficient determination of back-extensions.

Lemma 5.4. All polygons S_k can be constructed in total $O(n \log n)$ time.

Proof. Let s_k be the segment corresponding to $\mathcal{R}_k$, i.e. the first segment for the reflex vertices r_f, ..., r_l. Note first that since r_f and r_l are in U and t_k and t_l are in U^+ the paths p_1 and p_2 lie in U^+. For different $\mathcal{R}_k$ the chains r_f, ..., r_l are disjoint; likewise are the paths $path(r_l, t_k)$. Now, any reflex vertex on p_2 may belong to at most one other polygon $S_{k'}$. For this, consider a reflex vertex r that lies on the two shortest paths, $path(h_k, r_f)$ and $path(h_{k'}, r_{f'})$.If r lies on the counterclockwise chain from h_k to r_f then r cannot lie on another such counterclockwise such chain for some $h_{k'}$, $r_{f'}$. The vertex r may, however, lie on some path $path(h_{k'}, r_{f'})$ if the path lies on the clockwise chain from $r_{f'}$ to $h_{k'}$. By the definition of first segment, r can then lie on at most one such path $path(h_{k'}, r_{f'})$.

By using the algorithm by [GHLST] which runs in time O(log n+ size of the path constructed) it follows that the total time complexity is $O(n \log n)$ for all $O(n)$ shortest paths so constructed.

q.e.d.

To compute extremal segments in P for each r in $\mathcal{R}_k$ it suffices to compute the extremal segments inside the polygon S_k. To see this we observe the following: each polygon S_k is a subpolygon of P; thus if two points are visible in S_k they are visible in P. Furthermore, all vertices of S_k are vertices of P except for h_k.

We have established the correctness of the following lemma.

Lemma 5.5. The extremal segment for each r in $\mathcal{R}_k$ (with respect to its relevant segment $s_k=(t_k,h_k)$) as computed in S_k is identical to that computed in P .

Therefore the boundary segments for all reflex vertices contributing to the link center can be determined by the following algorithm in $O(n \log n)$ time.

Algorithm 5.3 Compute Boundary Segments

Input: The polygons S_k and the pairs $(s_k, \mathcal{R}_k)$

Output: The boundary segments for each reflex vertex contributing to the link center

1. **for** each subpolygon S_k **do**
2. preprocess S_k for shooting towards s_k
3. **for** each r in $\mathcal{R}_k$ **do**
4. find extremal segment for (r, s_k) in S_k
5. determine the back-extension by shooting in P and output it as a boundary segment for r.

The correctness of the algorithm follows from the above.

5.3 Computing relevant boundary segment (general case)

We have shown how to construct i-BS's given that any boundary segment entering a region U is an $(i$-1)-BS. Now consider the case that for some convex vertex v one of its boundary segments enters U "earlier", i.e. some k-BS intersects U for $k<i$-1. The lemmas presented so far in this section are easily adapted to that situation.

Consider a particular region U of the window tree and assume that all children of its corresponding node have been processed (as in Section 4). Let m be the maximum index on BS's produced that intersects the region. Suppose first that all such boundary segment are m-BS's. Then we have the case considered above. Next assume that not all BS's are m-BS's.

Clearly all k-BS's for $k\leq m$-3 are not relevant BS's. This holds since the covering radius of any such segment s' in U is three and thus U is contained in $P(s')$.

For those boundary segments s' which are $(m$-2) -BS's we perform Algorithm 5.1 to obtain (m-1)-BS's (if any). Then we take these and all other (m-1)-BS's and apply again Algorithm 5.1 to obtain m-BS's. These are then treated together with the original m-BS's as discussed above.

Thus we get:

Theorem 5.6. The boundary segments for all reflex vertices contributing to the link center can be determined in O($n \log n$) time.

We have completed the main task for solving the problem of determining the link center, i.e. to identify the boundary segments contributing to the link center. For completeness we include an algorithm for computing the link center from the boundary segment determined in Theorem 5.6; it is stated in the next section.

6. Processing the boundary segments to obtain the boundary of the link center

The set of counterclockwise and the clockwise boundary segments of the reflex vertices computed in the previous sections contains all edges of the link center; (some boundary segments are not co-linear with any edges of the link center). To compute the link center from the boundary segments we apply the following algorithm.

Algorithm 6.1 Determining the boundary of the link center

Input: A simple polygon P , the polygon $V := VisPol(d, 2)$, a list of all reflex vertices in V, and their boundary segments.

Output: The link center of P

Let $R = r_1,\ldots,r_{last}$ be the list of all reflex vertices of V (given in clockwise order).

Denote by $s_{i,1}$ and $s_{i,2}$ the counterclockwise and the clockwise boundary segments of r_i, respectively.

LCB:=boundary of V {initialization of Link Center Boundary};

c:=clockwise;

for *j*:=1 **to** 2 **do**

{*j*=1 corresponds to *c*=clockwise and *j*=2 corresponds to *c*=counterclockwise}

begin

direct the edges of V and *LCB* according to *c*;

for *i*:=1 **to** *last*

begin

If r_i is not removed **then** $r := r_i$;

else {compute the first intersection point r of $s_{i,j}$ and *LCB*}

begin

z_1:=*delete*[r_i];

z_2:=the second endpoint of the most recently inserted segment in *LCB* ;

find the intersection point p of $s_{i,j}$ with the boundary of V;

if $p \in chain_V(z_{3-j}, z_j)$ then skip the current i-step {$s_{i,j}$ is not relevant for *LCB*}

r:=the intersection point of $s_{i,j}$ with $chain_{LCB}(z_{3-j}, z_j)$;

{since $chain_{LCB}(z_{3-j}, z_j)$ is a convex chain, r can be located in O(log n) time by a binary search}

end

e:=the edge of *LCB* incident with r in direction *c*;

remove *e* from *LCB*;

delete[first endpoint of *e*]:=r;

```
        repeat
            e:=next edge of LCB in direction c;
            remove e from LCB;
            delete[first endpoint of e] :=r;
            If s_{i,j} intersects e then
                compute the intersection point z of s_{i,j} and e;
                If shot_V(z,r)=segment(z,r) then
                    {e is the edge from LCB hit by s_{i,j} (see proof of Lemma 6.1. below)}
                    insert segment(r,z) and segment(z,v(e)) in LCB,
                     where v(e) is the endpoint of e that is in orientation c from z;
        until (a portion of) s_{i,j} has been inserted in LCB
    end
    R := r_last,...,r_1;
  c:=counterclockwise;
end
```

Lemma 6.1. Algorithm 6.1 computes the intersection of the regions determined by the boundary segments and its time complexity is $O(n \log n)$.

Proof: Denote by B the boundary of V. W.l.o.g. we consider the clockwise direction, i.e. j=1. In the repeat-loop of Algorithm 6.1 the edges of B are examined in a clockwise direction until an edge e is discovered which intersects with $s_{i,j}$ at a point z. Next the algorithm checks whether e is visible in P from r in direction $s_{i,j}$. For this, a shot towards r is made originating at point z; assume that this shot reaches r. In this case, we claim that e is the edge from LCB "hit" by $s_{i,j}$. To prove that we need to show that before the i-th iteration for j=1, r_i is visible from e in V in the direction opposite to $s_{i,1}$ if and only if r_i is visible from e in the polygon V' determined by LCB. There exist the following cases:

(i) The shot $s^*=shot_V(z,r_i)$ intersects an edge of both V and V'. Then r_i is neither visible in V nor in V'.

(ii) s^* intersects an edge f belonging to the set of edges of B-LCB (i.e. B without LCB). Then r_i is not visible in V. Moreover, there exists a segment t of LCB inside V that cuts off a portion of B containing f. Then s^* can not see r_i in V' because of (at least) t. Then r_i is not visible in V'.

(iii) s^* does not intersect an edge of B. Assume that s^* intersects an edge t of LCB. As e appears on LCB before t, the endpoints of t are between the second endpoint of e and r (clockwise). Therefore s^* hits another edge edge w of LCB-B (let t and w be the first two such edges). But the angle between any two consecutive edges from LCB-B is convex. Thus there exist edges of B-LCB on $chain_V(w,t)$. It follows that s^* intersects some chain of B . The contradiction shows that s* does not intersect LCB and thus r_i is visible both in V and in V'. The correctness of the iteration for j=2 is proved in the same way.

Notice that we can not directly shoot from r to LCB to locate the sought edge e, since LCB changes dynamically and such shooting can not be performed efficiently. Instead, in Algorithm 4.5 we do the shooting in V polygon, which requires only O(log n) time after linear time preprocessing of V.

We give an upper bound on the time complexity of Algorithm 6.1. Denote

n_1 = # of edges inserted in $LCB \leq 2*$(# of reflex vertices of V)=O(n);

n_2 = # of edges removed from $LCB \leq$ (# of vertices of V)+n_1=O(n);

n_3 = # of intersections computed $\leq n_1+n_2$ = O(n);

n_4 = # of shots produced $\leq$ # of reflex vertices of V =O(n).

Then the time required by Algorithm 4.5 does not exceed

n_1*O(1)+n_2*O(1)+n_3*O(1)+n_4*O(log n) = O(n log n). q.e.d.

The results from this paper can be summarized in the following theorem.

Theorem 6.2. For any simple n-vertex polygon P the link center and the link radius of P can be determined in O(n log n) time.

7. Extensions

A central link segment in an n-vertex simple polygon P is a segment s of P that minimizes the number $min_{q \in s}\, max_{p \in P}$ dist(q,p), where dist(q,p) is the link distance in P between two points p, q of P. Constructing the central link segment has applications to finding an optimal location of a robot in a polygonal region and to solving the problem of determining the minimum value k for which a given polygon is k-visible from some segment.The technique presented in this paper has been used in the design of an O(n log n)-time algorithm for finding a central link segment in P; see [ADS] (an alternate method is suggested in [K]).

One can generalize the problem of computing a link center of a simple polygon as follows. Let P be a simple polygon, let Z be a set of points and S be a set of segments both located on the boundary of P ; let the sizes of S *and* Z be linear in the number n of vertices of P. Then the first algorithm described here can be adapted to finding the set of points in P which minimize the maximum link distance to all points in S and in P.

The algorithm presented in Section 4 and 5 can also be used to solve the problem of determining the external link center of a polygon, i.e. that region(s) of the exterior of P for which each point inside the region minimizes the maximum (exterior) link distance to all points of P [ADS2].

References

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[ADS] L. Alexandrov, H. Djidjev, J.-R. Sack, Finding a central link segment of a simple polygon in O(n log n) time, technical report TR-89-7, Bulgarian Academy of Sciences, May 1989; also tech. rept. SCS-TR-163, School of Computer Science, Carleton University.

[ADS2] L. Alexandrov, H. Djidjev, J.-R. Sack, Finding the external link center of a simple polygon, unpublished manuscript, April 1990.

[Ch] B. Chazelle, A theorem on polygon cutting with applications, *Proc. 23rd IEEE Symp. on Foudations of Computer Science* (1982) pp. 339-349.

[D] H. Djidjev, Linear algorithms for graph separation problems, *Proc. SWAT'88*, Lecture Notes in Computer Science (1988) vol. 318, Springer Verlag, Berlin, Heidelberg, New York, Tokyo, pp. 216-221.

[DLS] H. N. Djidjev, A. Lingas, J.-R. Sack, An O(n log n) algorithm for computing a link center in a simple polygon, Proc. STACS'89, Paderborn FRG, February 1989, Lecture Notes in Computer Science, Vol. 349, Springer Verlag, Berlin, Heidelberg, New York, Tokyo, pp. 96-107.

[GHLST] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica* **2** (1987), pp. 209-233.

[K] Y. Ke, An efficient algorithm for link distance problems inside a simple polygon, ACM Symposium on Computational Geometry, Saarbrücken FRG, June 1989 pp. 69-78.

[LPSSSSTWY] W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides, and C. Yap, Computing the link center of a simple polygon, *Discrete and Computational Geometry* **3**, 3 (1988) pp. 281-293.

[LT] R.J. Lipton, and R.E. Tarjan, A separator theorem for planar graphs, *SIAM J. Appl. Math.* **36** (1979) 177-189.

[PRS] R. Pollack, G. Rote, and M. Sharir, Computing the geodesic center of a simple polygon, *Discrete and Computational Geometry* **4**:6 (1989) pp. 611-626.

[S] J.-R. Sack, Movability of polygons in the plane, *Proc. STACS 85*, Saarbrücken, FRG, January 1985, Lecture Notes in Computer Science, No. 182, Springer Verlag, Berlin, Heidelberg, New York, Tokyo, pp. 310-321; *Robotica* **5** (1987) pp. 55-63.

[SS] J.-R. Sack, and S. Suri, An optimal algorithm for detecting weak visibility of a polygon, to appear in *IEEE Transactions on Computers* (1990).

[Su86] S. Suri, A linear time algorithm for minimum link paths inside a simple polygon, *Computer Vision, Graphics, and Image Processing* **35** (1986) pp. 99-110.

[Su87a] S.Suri, The all-geodesic-furthest neighbors problem for simple polygons, *Proc. 3rd ACM Conference on Computational Geometry* (1987) pp. 64-75.

[Su87b] S. Suri, Minimum link paths in polygons and related problems, Ph. D. Thesis, Dept. of Comp. Science, Johns Hopkins University, August 1987.

[T] G.T. Toussaint, Shortest path solves edge-to-edge visibility in a polygon, Tech. Rept. SOCS-84.39, McGill University, Montréal, 1985.

School of Computer Science, Carleton University
List of Recent Technical Reports

SCS-TR-142 **An Algorithm for Distributed Mutual Exclusion on Arbitrary Networks**
H. Glenn Brauen and John E. Neilson, September 1988

SCS-TR-143 to 146 are unavailable.

SCS-TR-147 **On Transparently Modifying Users' Query Distributions**
B.J. Oommen and D.T.H. Ng, November 1988

SCS-TR-148 **An O(N Log N) Algorithm for Computing a Link Center in a Simple Polygon**
H.N. Djidjev, A. Lingas and J.-R. Sack, July 1988, revised full version Sept. 1990
Published in Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Lecture Notes in Computer Science, Springer-Verlag No. 349, pp 96-107.

SCS-TR-149 **Smallscript: A User Programmable Framework Based on Smalltalk and Postscript**
Kevin Haaland and Dave Thomas, November 1988

SCS-TR-150 **A General Design Methodology for Dictionary Machines**
Frank Dehne and Nicola Santoro, February 1989

SCS-TR-151 **On Doubly Linked List ReOrganizing Heuristics**
D.T.H. Ng and B. John Oommen, February 1989

SCS-TR-152 **Implementing Data Structures on a Hypercube Multiprocessor, and Applications in Parallel Computational Geometry**
Frank Dehne and Andrew Rau-Chaplin, March 1989

SCS-TR-153 **The Use of Chi-Squared Statistics in Determining Dependence Trees**
R.S. Valiveti and B.J. Oommen, March 1989

SCS-TR-154 **Ideal List Organization for Stationary Environments**
B. John Oommen and David T.H. Ng, March 1989

SCS-TR-155 **Hot-Spot Contention in Binary Hypercube Networks**
Sivarama P. Dandamudi and Derek L. Eager, April 89

SCS-TR-156 **Some Issues in Hierarchical Interconnection Network Design**
Sivarama P. Dandamudi and Derek L. Eager, April 1989

SCS-TR-157 **Discretized Pursuit Linear Reward-Inaction Automata**
B.J. Oommen and Joseph K. Lanctot, April 1989

SCS-TR-158 (revised) **Parallel Fractional Cascading on a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989 (Revised April 1990)

SCS-TR-159 **Epsilon-Optimal Stubborn Learning Mechanisms**
J.P.R. Christensen and B.J. Oommen, June 1989

SCS-TR-160 **Disassembling Two-Dimensional Composite Parts Via Translations**
Doron Nussbaum and Jörg-R. Sack, June 1989

SCS-TR-161 (revised) **Recognizing Sources of Random Strings**
R.S. Valiveti and B.J. Oommen, January 1990
Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989

SCS-TR-162 **An Adaptive Learning Solution to the Keyboard Optimization Problem**
B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989

SCS-TR-163 **Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time**
L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989

SCS-TR-164 **A Survey of Algorithms for Handling Permutation Groups**
M.D. Atkinson, January 1990

SCS-TR-165 **Key Exchange Using Chebychev Polynomials**
M.D. Atkinson and Vincenzo Acciaro, January 1990

SCS-TR-166 **Efficient Concurrency Control Protocols for B-tree Indexes**
Ekow J. Otoo, January 1990

SCS-TR-167 **A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem**
B.J. Oommen, January 1990

SCS-TR-168 **Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule**
R.S. Valiveti and B.J. Oommen, January 1990

SCS-TR-169 **Trade-Offs in Non-Reversing Diameter**
Hans L. Bodlaender, Gerard Tel and Nicola Santoro, February 1990

SCS-TR-170 **A Massively Parallel Knowledge-Base Server using a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-171 **Parallel Processing of Quad Trees on the Hypercube (and PRAM)**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-172 **A Note on the Load Balancing Problem for Coarse Grained Hypercube Dictionary Machines**
Frank Dehne and Michel Gastaldo, May 1990

SCS-TR-173 **Self-Organizing Doubly-Linked Lists**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-174 **A Presortedness Metric for Ensembles of Data Sequences**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-175 **Separation of Graphs of Bounded Genus**
Ljudmil G. Aleksandrov and Hristo N. Djidjev, May 1990

SCS-TR-176 **Edge Separators of Planar and Outerplanar Graphs with Applications**
Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora and Imrich Vrto, May 1990

SCS-TR-177 **Representing Partial Orders by Polygons and Circles in the Plane**
Jeffrey B. Sidney and Stuart J. Sidney, July 1990

SCS-TR-178 **Determining Stochastic Dependence for Normally Distributed Vectors Using the Chi-squared Metric**
R.S. Valivetei and B.J. Oommen, July 1990

SCS-TR-179 **Parallel Algorithms for Determining K-width- Connectivity in Binary Images**
Frank Dehne and Susanne E. Hambrusch, September 1990

SCS-TR-180 **A Workbench for Computational Geometry (WOCG)**
P. Epstein, A. Knight, J. May, T. Nguyen, and J.-R. Sack, September 1990

SCS-TR-181 **Adaptive Linear List Reorganization under a Generalized Query System**
R.S. Valiveti, B.J. Oommen and J.R. Zgierski, October 1990

SCS-TR-182 **Breaking Substitution Cyphers using Stochastic Automata**
B.J. Oommen and J.R. Zgierski, October 1990

SCS-TR-183 **A New Algorithm for Testing the Regularity of a Permutation Group**
V. Acciaro and M.D. Atkinson, November 1990

SCS-TR-184 **Generating Binary Trees at Random**
M.D. Atkinson and J.-R. Sack, December 1990

SCS-TR-185 **Uniform Generation of Combinatorial Objects in Parallel**
M.D. Atkinson and J.-R. Sack, January 1991