Nordic Journal of Computing 1(1994), 458–474.

AN $O(\log \log n)$ TIME ALGORITHM TO COMPUTE THE KERNEL OF A POLYGON*

SVEN SCHUIERER
Institut für Informatik, Universität Freiburg
Rheinstr. 10-12, D-79104 Freiburg, Fed. Rep. of Germany
schuierer@informatik.uni-freiburg.de

Abstract. The kernel of a polygon P is the set of all points that see the interior of P. It can be computed as the intersection of the halfplanes that are to the left of the edges of P. We present an $O(\log \log n)$ time CRCW-PRAM algorithm using $n/\log \log n$ processors to compute a representation of the kernel of P that allows to answer point containment and line intersection queries efficiently. Our approach is based on computing a subsequence of the edges that are sorted by slope and contain the "relevant" edges for the kernel computation.

CR Classification: F.2.2

Key words: computational geometry, visibility, polygons, kernel

1. Introduction

Visibility problems play an important role in many applications and are a well studied field in computational geometry [O'Rourke 1987], [Preparata and Shamos 1985]. Given a simple polygon P in the plane we say two points p and q in P are *visible* from each other or *see* each other if the line segment $\overline{pq}$ is contained in P. The *kernel of P kernel*(P) is then defined as the set of all points that see all the other points in P. It can be easily shown that *kernel*(P) is the intersection of all the halfplanes that lie to the left of the polygon's edges given a counterclockwise orientation of P [Lee and Preparata 1979].

Although $\Omega(n \log n)$ time is required to compute the intersection of n arbitrary halfplanes, the fact that the halfplanes correspond to the edges of a simple polygon can be exploited to obtain a linear time sequential algorithm [Lee and Preparata 1979] or an $O(\log n)$ time algorithm for a CREW-PRAM with $n/\log n$ processors [Cole and Goodrich 1992]. Other visibility related algorithms for parallel models of computation that have been previously studied are an optimal $O(\log n)$ time CREW-PRAM algorithm to find the visibility polygon of a point [Atallah, Chen, and Wagener 1991] and an optimal $O(\log n)$ time CREW-PRAM algorithm for detecting weak visibility of a simple polygon [Chen 1992].

* This work was supported by the Deutsche Forschungsgemeinschaft under Grant No. Ot 64/8-1.

Received May 1994. Accepted November 1994.

Our approach to compute the kernel of a polygon is very similar to the one used in [Cole and Goodrich 1992]. In a first step the halfplanes which are irrelevant for the kernel computation are filtered out and, then, a representation of the intersection of the remaining set of halfplanes is computed. This representation of the kernel allows to answer point containment and line intersection queries in time $O(\log n/\log p)$ if p processors are available.

As it turns out this idea can be implemented on a parallel random access machine where concurrent write and read accesses to one cell are allowed (*CRCW-PRAM*) in time $O(\log\log n)$ using $n/\log\log n$ processors. The particular model of CRCW-PRAM we make use of here is the *COMMON* CRCW-PRAM where concurrent write is only allowed if all processors write the same value to one cell.

The paper is organized as follows. After introducing some notation we describe in the third section which edges can be filtered out. The result are two sequences of halfplanes which are sorted by slope and whose intersection yields the kernel. It is shown how to compute these sequences of halfplanes efficiently in parallel. In Section 4 we describe how to use dualization to find a representation of the intersection of these halfplanes that allows efficient point-containment and line-intersection queries.

2. Definitions and Notation

As was pointed out above the kernel of a simple polygon P is the intersection of all the halfplanes that are to the left of its edges given a counterclockwise orientation of P. Since the approach we present works for closed and open curves we assume in the following that we are given a simple oriented polygonal chain $\mathcal{C} = (e_1, \ldots, e_n)$ where edge e_i starts at vertex v_i and ends at vertex v_{i+1}.

We denote the halfplane with e_i on its boundary and interior to the left of e_i by $h^+(e_i)$ and the halfplane with e_i on its boundary and interior to the right of e_i by $h^-(e_i)$. The *kernel* of $\mathcal{C}$ is defined as $\bigcap_{1\leq i\leq n} h^+(e_i)$. Hence, if $\mathcal{C}$ is a closed curve with a counterclockwise orientation, then the kernel of $\mathcal{C}$ is the set of all points that see the interior of $\mathcal{C}$.

One main step in our algorithm is to produce a sequence of edges that are sorted according to turning angle. By turning angle we mean the angle of the edge plus the number of times the chain has spiraled around itself. Here, the *angle* of edge e_i is defined as the angle between the directed line through e_i and the x-axis and is denoted by $\Theta(e_i)$.

The turning angle can be defined incrementally by considering the difference between the angles of two consecutive edges. If e_i and e_{i+1} are two consecutive edges of $\mathcal{C}$, then we define the *incremental angle* between e_i and e_{i+1} to be the signed angle between e_i and e_{i+1} if they are considered as vectors and denote it by $incr(e_i, e_{i+1})$. $incr(e_i, e_{i+1})$ is positive if e_{i+1} turns left from e_i and negative if e_{i+1} turns right.

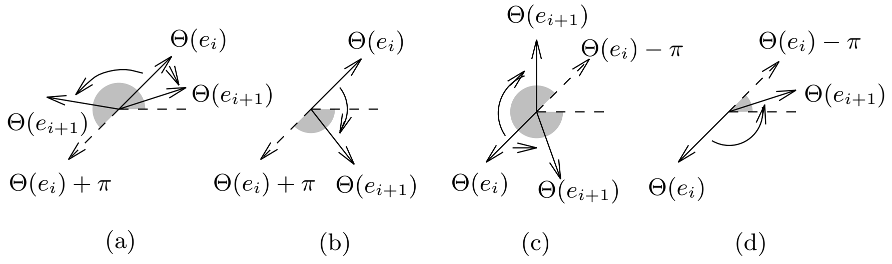

Fig. 1: The definition of $incr(e_i, e_{i+1})$.

More precisely, $incr(e_i, e_{i+1})$ is defined as follows (see Fig. 1):

$$incr(e_i, e_{i+1}) = \begin{cases} \text{(a) } \Theta(e_{i+1}) - \Theta(e_i) & \text{if } \Theta(e_{i+1}) < \Theta(e_i) + \pi \leq 2\pi; \\ \text{(b) } \Theta(e_{i+1}) - \Theta(e_i) - 2\pi & \text{if } \Theta(e_{i+1}) > \Theta(e_i) + \pi; \\ \text{(c) } \Theta(e_{i+1}) - \Theta(e_i) & \text{if } \Theta(e_{i+1}) > \Theta(e_i) - \pi \geq 0; \\ \text{(d) } \Theta(e_{i+1}) - \Theta(e_i) + 2\pi & \text{if } \Theta(e_{i+1}) < \Theta(e_i) - \pi. \end{cases}$$

We can now define the *turning angle* $\Theta_{\mathcal{C}}(e_i)$ of an edge e_i w.r.t. $\mathcal{C}$ inductively by

$$\begin{aligned} \Theta_{\mathcal{C}}(e_1) &= \Theta(e_1) \\ \Theta_{\mathcal{C}}(e_{i+1}) &= \Theta_{\mathcal{C}}(e_i) + incr(e_i, e_{i+1}) \\ &= \Theta(e_1) + \sum_{j=1}^{i} incr(e_j, e_{j+1}). \end{aligned}$$

The chain $\mathcal{C}$ *spirals* if there are two indices i and j such that $\Theta_{\mathcal{C}}(e_j) - \Theta_{\mathcal{C}}(e_j) > 3\pi$.

3. Highly Parallel Computation of the Kernel

In this section we present an $O(\log\log n)$ algorithm for a CRCW-PRAM with $O(n/\log\log n)$ processors to compute the kernel of a simple polygonal chain.

To this end we say edge e is *relevant* if the line through e intersects $kernel(\mathcal{C})$ in more than one point. We now construct two sets S and S' which contain all the relevant edges of $\mathcal{C}$.

Let $\theta_i = \max_{1\leq j\leq i} \Theta_{\mathcal{C}}(e_j)$ and let S be the set of edges e_i with $\Theta_{\mathcal{C}}(e_i) = \theta_i > \theta_{i-1}$, where $\theta_0 = -\infty$, i.e., S is the set of edges whose turning angle is larger than the turning angle of any previous edge. Furthermore, suppose we mirror $\mathcal{C}$ at the y-axis and revert the orientation of the edges and call this new chain $\mathcal{C}'$ consisting of the edges $e'_1, \ldots, e'_n$ where e'_i is the image of edge e_i. Note that the kernel of $\mathcal{C}'$ is the image of the kernel of $\mathcal{C}$ and a relevant edge of $\mathcal{C}'$ is the image of a relevant edge of $\mathcal{C}$. Hence, we can define

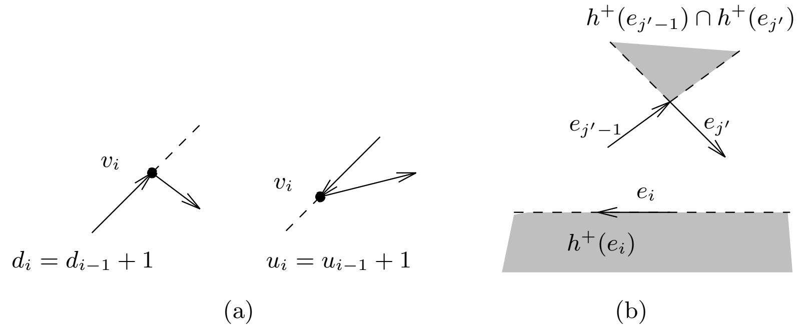

Fig. 2: (a) The definition of up- and downturns and (b) The kernel of $\mathcal{C}$ is empty if $\Theta_{\mathcal{C}}(e_i) - \Theta_{\mathcal{C}}(e_{j'}) \geq \pi$.

$\theta'_i = \max_{i\leq j\leq n} \Theta_{\mathcal{C}'}(e'_j)$. Let S' be the sequence of edges of $\mathcal{C}$ with $\theta'_i > \theta'_{i+1}$. In [Cole and Goodrich 1992] it is shown that if $\mathcal{C}$ does not spiral, then $S \cup S'$ contains the relevant edges of $\mathcal{C}$, i.e.

$$kernel(\mathcal{C}) \quad = \quad \bigcap_{e\in S} h^+(e) \cap \bigcap_{e \in S'} h^+(e).$$

With the above observations it seems that the following approach to computing the kernel is reasonable.

1. Compute S and S';
2. Merge S and S' into a single sequence S'';
3. Intersect the halfplanes in S'';

In the following we will address each of the three steps.

3.1 Computing S

The computation of S consists of two steps. First we have to compute $\Theta_{\mathcal{C}}(e_i)$, and then θ_i, for $1 \leq i \leq n$. Computing $\Theta_{\mathcal{C}}(e_i)$ essentially amounts to computing the sums $\sum_{j=1}^{i} incr(e_j, e_{j+1})$, for $1 \leq i \leq n$. Unfortunately, there is a lower bound of $\Omega(\log n / \log\log n)$ for the computation of the parity function of n bits on a CRCW-PRAM which, in particular, implies a lower bound for the computation of the sum of n numbers.

In order to beat this lower bound, we make use of a different representation of $\Theta_{\mathcal{C}}(e_i)$. We say vertex v_{i+1} with adjacent edges e_i and e_{i+1} is a *downturn* if $\Theta(e_{i+1}) > \Theta(e_i)+\pi$. Similarly, we say vertex v_{i+1} is an *upturn* if $\Theta(e_{i+1}) < \Theta(e_i) - \pi$ (see Fig. 2a). Let the number of upturns up to and including vertex v_i be u_i and the number of downturns d_i.

LEMMA 3.1. *If $\mathcal{C}$ is a polygonal chain, then*

$$\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) + (u_i - d_i)2\pi.$$

Proof. The proof is by induction on the number of edges. The claim obviously holds for $i = 1$. Now suppose the claim is true for some $i > 1$, i.e., $\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) + (u_i - d_i)2\pi$. We want to show that $\Theta_{\mathcal{C}}(e_{i+1}) = \Theta(e_{i+1}) + (u_{i+1} - d_{i+1})2\pi$ with $\Theta_{\mathcal{C}}(e_{i+1}) = \Theta_{\mathcal{C}}(e_i) + \mathit{incr}(e_i, e_{i+1})$. In order to do so we distinguish four cases depending on $\Theta(e_i)$.

(1) $\Theta(e_{i+1}) < \Theta(e_i) + \pi \leq 2\pi$.
By definition we have $\mathit{incr}(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i)$, $d_{i+1} = d_i$ and $u_{i+1} = u_i$. Hence,

$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i+1}) &= \Theta_{\mathcal{C}}(e_i) + \mathit{incr}(e_i, e_{i+1}) \\ &= \Theta(e_i) + (u_i - d_i)2\pi + \Theta(e_{i+1}) - \Theta(e_i) \\ &= \Theta(e_{i+1}) + (u_{i+1} - d_{i+1})2\pi.\end{aligned}$$

(2) $\Theta(e_{i+1}) > \Theta(e_i) + \pi$.
We have $\mathit{incr}(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i) - 2\pi$, $d_{i+1} = d_i + 1$, and $u_{i+1} = u_i$. Hence,

$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i+1}) &= \Theta_{\mathcal{C}}(e_i) + \mathit{incr}(e_i, e_{i+1}) \\ &= \Theta(e_i) + (u_i - d_i)2\pi + \Theta(e_{i+1}) - \Theta(e_i) - 2\pi \\ &= \Theta(e_{i+1}) + (u_{i+1} - (d_i + 1))2\pi \\ &= \Theta(e_{i+1}) + (u_{i+1} - d_{i+1})2\pi.\end{aligned}$$

(3) The cases $\Theta(e_{i+1}) > \Theta(e_i) - \pi \geq 0$ and $\Theta(e_{i+1}) < \Theta(e_i) - \pi$ can be handled analogously.

Hence, the computation of $\Theta_{\mathcal{C}}(e_i)$ can be reduced to computing $t_i = u_i - d_i$. If we consider the sequence (τ_i), where

$$\tau_i = \begin{cases} +1 & \text{if } v_i \text{ is an upturn,} \\ -1 & \text{if } v_i \text{ is a downturn, and} \\ 0 & \textit{otherwise,} \end{cases}$$

then (t_i) is the sequence of the prefix sums of τ_i. As we pointed out before there is a lower bound of $\Omega(\log n / \log\log n)$ to compute the parity of bit sequences and, thus, prefix sums [Beam and Hastad 1987]. But if the sequence (t_i) originates from a simple starshaped polygonal curve, then it spans only a very small range. In order to proof this we need the following observation.

Lemma 3.2. *Let e_i and $e_{i'}$ be two consecutive edges in S. If there is an edge e_j with $i < j < i'$ such that $\Theta_{\mathcal{C}}(e_i) - \Theta_{\mathcal{C}}(e_j) \geq \pi$, then kernel$(\mathcal{C})$ is empty.*

Proof. Let j' be the smallest index $j' > i$ with $\Theta_{\mathcal{C}}(e_i) - \Theta_{\mathcal{C}}(e_{j'}) \geq \pi$. Hence, the part of $\mathcal{C}$ from e_i to $e_{j'-1}$ is monotone w.r.t. $\Theta(e_i) + \pi$ and $h^+(e_i) \cap h^+(e_{j'-1}) \cap h^+(e_{j'}) = \emptyset$ (see Fig. 2b). □

In the following we call a vertex v_i that is an upturn or a downturn a *turn* and we say two turns v_i and v_j are *consecutive* if there is no turn between v_i and v_j.

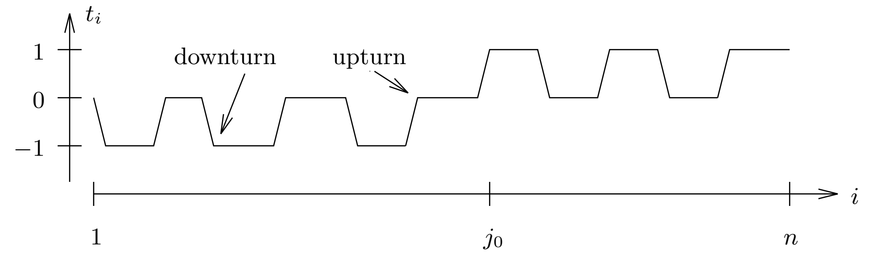

Fig. 3: The graph of the t_i-values of simple polygon with non-empty kernel.

LEMMA 3.3. *If $\mathcal{C}$ is a polygonal chain with non-empty kernel, then t_i is contained in the interval $[-1, 1]$, for all $1 \le i \le n$. Furthermore, there are no two consecutive downturns.*

PROOF. The proof is by contradiction. If there is an i with $t_i > 1$, then we have $\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) + t_i 2\pi > 4\pi$ which implies that $kernel(\mathcal{C})$ is empty [Cole and Goodrich 1992], [Lee and Preparata 1979] in contradiction to the assumption of the lemma.

If there is an i with $t_i < -1$, then $\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) - t_i 2\pi < -2\pi$. Let e_{i_l} be the edge in S with maximum $i_l < i$. Since $\Theta_{\mathcal{C}}(e_{i_l}) > \Theta_{\mathcal{C}}(e_f(\mathcal{C}))$, we have $\Theta_{\mathcal{C}}(e_{i_l}) - \Theta_{\mathcal{C}}(e_i) > \Theta_{\mathcal{C}}(e_f(\mathcal{C})) - \Theta_{\mathcal{C}}(e_i) > 2\pi$ which implies by Lemma 3.2 that $kernel(\mathcal{C})$ is also empty in this case.

To see the second claim assume there are two downturns v_i and v_j such that there is no upturn between v_i and v_j. Consider the edge e_{i-1} and some edge e_k with $i \le k < j$. By Lemma 3.1 we have

$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i-1}) - \Theta_{\mathcal{C}}(e_k) &= \Theta(e_{i-1}) - \Theta(e_k) + (u_{i-1} - u_k - (d_{i-1} - d_k))2\pi \\ &\ge -2\pi + 2\pi \ge 0.\end{aligned}$$

Thus, if e_{i_l} is the edge in S with maximum $i_l < j$, then $i_l \le i - 1$ and

$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i_l}) - \Theta_{\mathcal{C}}(e_j) &\ge \Theta_{\mathcal{C}}(e_{i-1}) - \Theta_{\mathcal{C}}(e_j) \\ &= \Theta(e_{i-1}) - \Theta(e_j) + (u_{i-1} - u_j - (d_{i-1} - d_j))2\pi \\ &\ge -2\pi + 2 \cdot 2\pi \ge 2\pi.\end{aligned}$$

and, hence, the $kernel(\mathcal{C})$ is empty by Lemma 3.2. □

Note that there may be two consecutive upturns. But since no two consecutive downturns may occur, Lemma 3.3 implies there is at most one pair of consecutive upturns. Fig. 3 displays a possible graph of the t_i-values for a simple polygon with non-empty kernel if a pair of consecutive upturns occurs.

In order to compute the values of t_i, for $1 \le i \le n$, we make use of an algorithm that solves the *all nearest smaller values* (*ANSV*) problem which is defined as follows [Berkman, Schieber, and Vishkin 1988]. Given a sequence of n elements $(a_1, \dots, a_n)$ from a totally ordered domain, find, for each element a_i, $1 \leq i \leq n$, the left and right closest smaller element, i.e., find the maximum $j < i$ with $a_j < a_i$ and find the minimum $k > i$ with $a_k < a_i$. The element a_j is called the *left match* of a_i and a_k is called the *right match*.

We now apply the ANSV-algorithm to the vertices of $\mathcal{C}$. For each vertex v_i we compute the closest turn v_j with $j > i$. In order to do so, processor i writes $n - i$ into an array cell a_i if vertex v_i is a turn and $n + 1$ otherwise. If we apply the ANSV-algorithm to array a, then we obtain a right match of each vertex. In a similar fashion we compute the left match. Given the two matches for each vertex, we can test in constant time with n processors if there are two or more consecutive downturns or three or more consecutive upturns. If this is the case, then $kernel(\mathcal{C})$ is empty by Lemma 3.3 and we stop.

We also have to check for pairs of consecutive upturns. In order to do so, we apply the ANSV-algorithm once again to compute the closest pair of consecutive upturns to the left and right of each vertex. If there is more than one such pair or there is one such pair and the first turn of $\mathcal{C}$ is an upturn, then $kernel(\mathcal{C})$ is empty by Lemma 3.3 and we stop.

Now let i_l be the index of the left match of vertex v_i. If a the pair of consecutive upturns exists, then let j_0 be the higher index of the pair; otherwise we set $j_0 = n + 1$ if the first turn of $\mathcal{C}$ is a downturn and $j_0 = 0$ if the first turn of $\mathcal{C}$ is an upturn. Note that i_l and j_0 have been precomputed in the previous steps. For each vertex v_i we set

$$t_i = \begin{cases} -1/2 + 1/2\tau_{i_l} + \tau_i & \text{if } i < j_0 \\ 1/2 + 1/2\tau_{i_l} + \tau_i & \text{if } i \geq j_0 \end{cases}$$

which by the above considerations assigns the correct t_i-value to each vertex. All of the above operations can be carried out in time $O(\log\log n)$ with $n/\log\log n$ processors.

Given the t_i values for each vertex, $\Theta_{\mathcal{C}}(e_i)$ can be computed in time $O(\log\log n)$, for all $1 \leq i \leq n$, by Lemma 3.1. In order to obtain S it is necessary to compute $\theta_i = \max_{1 \leq j \leq i} \Theta_{\mathcal{C}}(e_j)$ which is a prefix maxima operation on $\Theta_{\mathcal{C}}(e_i)$ and can be carried out in time $O(\log\log n)$ with $n/\log\log n$ processors [Berkman, Schieber, and Vishkin 1988].

The sequence S' can be computed analogously.

Finally, we have to check if $\mathcal{C}$ spirals. Cole and Goodrich [1992] show that by computing the minimum of $\Theta_{\mathcal{C}_i}(e_i)$, θ_i and the prefix maximum of $(\theta_i + \theta'_i)$ it can be decided if $\mathcal{C}$ spirals or not. Again each of these steps can be carried out in $O(\log\log n)$ steps with $n/\log\log n$ processors [Berkman, Schieber, and Vishkin 1988].

3.2 Merging S and S'

The important property of S and S' is that the edges in both sequences are sorted by turning angles. Hence, an application of an optimal $O(\log\log n)$ merging algorithm to obtain a merged sequence S'' seems possible. Unfortunately, the edges of S and S' are scattered among the edges of $\mathcal{C}$ which we assume to be stored in an array A. In order to compact the edges of S and S' into a contiguous part of A we need at least $\Omega(\log|S|/\log\log n)$ time [Ragde 1990] which we cannot afford since S (or S') can be as large as $\Omega(n)$. The solution is to apply the ANSV-algorithm in order to find, for each edge e not in S, its left closest match $lcm_S(e)$ in S and the same for edges not in S'. We create two new arrays A and A' with $A[j] = e_j$ if e_j is in S and $A[j] = lcm_S(e_j)$ otherwise and similar for S' and A'. Note that $\bigcap_{e\in A} h^+(e) = \bigcap_{e\in S} h^+(e)$ and $\bigcap_{e\in A'} h^+(e) = \bigcap_{e\in S'} h^+(e)$. Hence, we can merge the two arrays A and A' into one array A'' of size $2n$ in time $O(\log\log n)$.

3.3 Intersecting the Halfplanes of S and S'

Since the *turning angles* of the edges in A'' vary at most between -2π and 4π, we can split A'' into $O(1)$ contiguous parts such that the *angles* of the edges in these parts are in the intervals $[-\pi/2, \pi/2)$ or $[\pi/2, 3\pi/2)$ and merge these parts again into two sequences E_+ and E_- such that the angles of all edges in E_+ are sorted and in the range $[-\pi/2, \pi/2)$ and the angles of the edges in E_- are also sorted and in the range $[\pi/2, 3\pi/2)$. In order to compute the intersection of the halfplanes associated with the edges in $E_+ \cup E_-$ we apply a dualization method as described below.

4. Halfplane Intersection and the Convex Hull

In this section we show how to reduce the intersection of halfplanes that are sorted by angle to the computation of the convex hull of certain point sets that are sorted on one coordinate. To this end we apply the concept of *geometric duality* which maps points onto lines and vice versa while keeping incidence relations. More precisely we apply the following dual map D; see Edelsbrunner [1987].

$$D(p) = \{(x,y) \in I\!R^2 \mid y = p_1x - p_2\} \text{ with } p = (p_1, p_2) \text{ and}$$
$$D(\ell) = (a_1, -a_2) \text{ with } \ell = \{(x,y) \in I\!R^2 \mid y = a_1x + a_2\}.$$

D has the special property that it preserves order which means in this case that if p is above the line ℓ, then $D(\ell)$ is above $D(p)$. In order to compute the intersection of the halfplanes $\mathcal{H} = \{h^+(e) \mid e \in A\}$ we now proceed as follows. $\mathcal{H}$ is split into the two sets $\mathcal{H}_+ = \{h^+(e) \mid e \in E_+\}$ and $\mathcal{H}_- = \{h^+(e) \mid e \in E_-\}$. Hence, $\bigcap \mathcal{H} = \bigcap \mathcal{H}_+ \cap \bigcap \mathcal{H}_-$. Note that all the halfplanes in $\mathcal{H}_+$ have their interior above their boundary and all the halfplanes in $\mathcal{H}_-$ have their interior below their boundary. Further, let $\mathcal{L}_+$

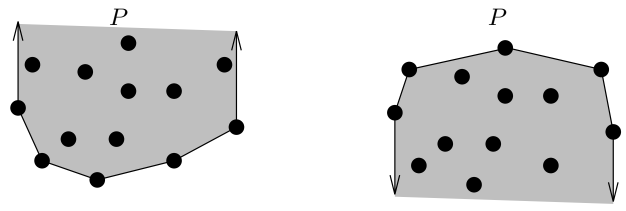

Fig. 4: The upper and lower convex hull of a point set P.

be the set of directed boundary lines of the halfplanes in $\mathcal{H}_+$ and $\mathcal{L}_-$ be defined analogously for $\mathcal{H}_-$. Now $p \in \bigcap \mathcal{H}_+$ if and only if p is above all lines ℓ in $\mathcal{L}_+$; in the dual plane this corresponds to the fact that for all $\ell \in \mathcal{L}_+$, the point $D(\ell)$ is above the line $D(p)$, i.e., that the upper convex hull of $D(\mathcal{L}_+)$ is above $D(p)$. The *upper convex hull* of $D(\mathcal{L}_+)$ is defined as the set of points that are above or in the convex hull of $D(\mathcal{L}_+)$. Similarly, the *lower convex hull* of $D(\mathcal{L}_-)$ is defined as the set of points that are below or in the convex hull of $D(\mathcal{L}_-)$ (see Fig. 4). We denote the boundary of the upper convex hull of $D(\mathcal{L}_+)$ by $\mathcal{B}_+(\mathcal{L}_+)$ and the boundary of the lower convex hull of $D(\mathcal{L}_-)$ by $\mathcal{B}_-(\mathcal{L}_-)$.

In the same way we get that $p \in \bigcap \mathcal{H}_-$ if and only if, for all $\ell \in \mathcal{L}_-$, the point $D(\ell)$ is below the line $D(p)$, i.e., if the lower convex hull of $D(\mathcal{L}_-)$ is below $D(p)$. The angles of the lines in $\mathcal{L}_+$ are sorted and contained in the interval $[-\pi/2, \pi/2)$ and the angles of the lines in $\mathcal{L}_-$ are sorted and contained in $[\pi/2, 3\pi/2)$. If the equation of the line bounding a halfplane with angle θ is $y = a_1x + a_2$, we have the relation $a_1 = \tan\theta$. Hence, $D(\mathcal{L}_+)$ and $D(\mathcal{L}_-)$ are point sets that are sorted by x-coordinate, and the optimal $O(\log\log n)$ time algorithm of Wagener [1992] can be used to compute the convex hull of $D(\mathcal{L}_+)$ and $D(\mathcal{L}_-)$.

Unfortunately, there is a lower bound of $\Omega(\log n/\log\log n)$ for the computation of an array representation of the convex hull of a sorted point set provided the number of processors remains polynomially bounded. This again follows from a reduction to compute the parity of the sum of n bits on a CRCW PRAM [Wagener 1992], [Beam and Hastad 1987]. In the appendix we show that the parity problem can also be reduced to computing the number of edges of the kernel of a polygon. If the edges of $kernel(\mathcal{C})$ are stored in a contiguous part of an array, then their number can be easily determined. Hence, $\Omega(\log n/\log\log n)$ time is required to compute an array-representation of $kernel(\mathcal{C})$ as well.

Therefore, a special data structure—called the *bridge tree*—is used in [Wagener 1992]. The bridge tree is defined on point sets and is not designed to be redualized. Nevertheless, we can use the bridge tree representation of the convex hull of $D(\mathcal{L}_+)$ and $D(\mathcal{L}_-)$ to answer several queries about $\bigcap \mathcal{H}_+$ and $\bigcap \mathcal{H}_-$.

If P is a point set with n points, a bridge tree supports the following queries in time $O(\log n/(\log p + 1) + 1)$ if p processors are assigned to the query:

(i) Given a point q, test whether q is contained in the upper hull of P.

(ii) Given a point q, report the tangents to the upper hull of P passing through q.

(iii) Given a line ℓ, report the intersection of ℓ with the upper hull of P.

(iv) Given a direction d, report the extremal points of the upper hull of P in direction d.

Of course, the same type of queries can be answered for the lower hull of P.

In the following it is our aim to show that queries (i)–(iv) can also be handled for $\mathcal{K} = \bigcap \mathcal{H}_+ \cap \bigcap \mathcal{H}_-$ within the same time bounds. A query of type (i) can be answered if we make use of the observation that the points contained in $\bigcap \mathcal{H}_+$ are mapped to lines that do not intersect the upper hull of $D(\mathcal{L}_+)$. Hence, to test if a point p is contained in $\mathcal{K}$ can be mapped to the query whether the dual line $D(p)$ does neither intersect the upper hull of $D(\mathcal{L}_+)$ nor the lower hull of $D(\mathcal{L}_-)$.

In order to answer a query of type (iii) we observe the following.

Lemma 4.1. *The line ℓ intersects $\mathcal{K}$ if and only if two of the four tangents of $D(\ell)$ to $\mathcal{B}_+(\mathcal{L}_+)$ and $\mathcal{B}_-(\mathcal{L}_-)$ do neither intersect the interior of the upper hull of $D(\mathcal{L}_+)$ nor the lower hull of $D(\mathcal{L}_-)$.*

Proof. The line ℓ intersects $\mathcal{K}$ if and only if there are two points s_1 and s_2 on ℓ which are also on the boundary of $\bigcap \mathcal{H}_+$ and/or $\bigcap \mathcal{H}_-$ and which belong to $\mathcal{K}$. Hence, the point $D(\ell)$ is the intersection point of the lines $D(s_1)$ and $D(s_2)$ and $D(s_1)$ and $D(s_2)$ are tangents to $\mathcal{B}_+(\mathcal{L}_+)$ and/or $\mathcal{B}_-(\mathcal{L}_-)$. Furthermore, $D(s_1)$ and $D(s_2)$ belong to $D(\mathcal{K})$ and, hence, do neither intersect the upper hull of $D(\mathcal{L}_+)$ nor the lower hull of $D(\mathcal{L}_-)$. □

Since there are only four tangents through $D(\ell)$ to the upper hull of $D(\mathcal{L}_+)$ and the lower hull of $D(\mathcal{L}_-)$ and these tangents can be found and tested for intersection with the interiors of the hulls in time $O(\log n/(\log p + 1) + 1)$ with the help of a bridge tree, we can report the intersection points of a line ℓ with $\mathcal{K}$, if they exist, within the same time bound (see Fig. 5).

4.1 Computing the Intersection Points of $\mathcal{B}_+(\mathcal{L}_+)$ and $\mathcal{B}_-(\mathcal{L}_-)$

In order to answer queries of type (ii) and (iv) for $\mathcal{K}$, we first have to find the intersection points p_1 and p_2 of the boundary of $\bigcap \mathcal{H}_+$ with the boundary of $\bigcap \mathcal{H}_-$ since, for example, p_1 is the extremal point of $\mathcal{K}$ in direction π. The lines $t_1 = D(p_1)$ and $t_2 = D(p_2)$ are simultaneously tangents to the upper hull of $D(\mathcal{L}_+)$ and the lower hull of $D(\mathcal{L}_-)$ (see Fig. 6).

Note that the upper hull of $D(\mathcal{L}_+)$ is on a different side of t_1 (t_2) as the lower hull of $D(\mathcal{L}_-)$. In order to compute t_1 and t_2 we make use of the

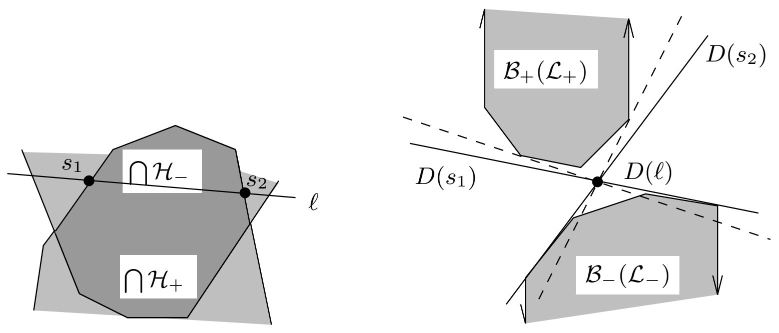

Fig. 5: The tangents to the upper hull of $D(\mathcal{L}_+)$ and the lower hull of $D(\mathcal{L}_-)$ correspond to the intersection points of ℓ.

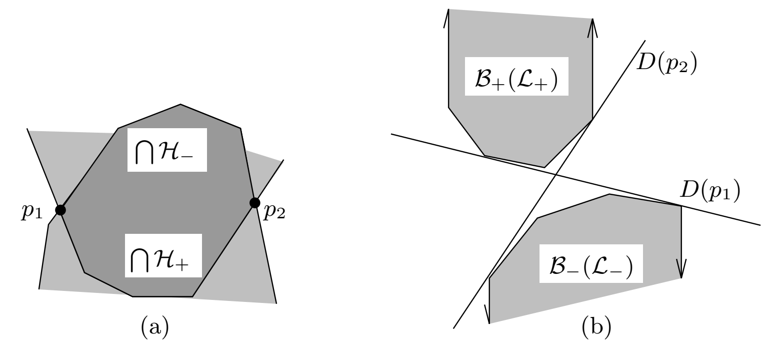

Fig. 6: The duality between $\bigcap \mathcal{H}$ and the upper hull of $D(\mathcal{L}_+)$ and the lower hull of $D(\mathcal{L}_-)$.

internal structure of the bridge tree for the the lower hull of $D(\mathcal{L}_-)$. It can be described as follows. On the first level of the bridge tree the points of $D(\mathcal{L}_-)$ are split into $n^{1/6}$ contiguous sequences $\mathcal{P}_i$ of points, each consisting of at most $O(n^{5/6})$ points. For each $\mathcal{P}_i$, at most two *bridges* are stored where a bridge is an edge of the lower hull of $D(\mathcal{L}_-)$ such that the x-range of the bridge intersects the x-range of $\mathcal{P}_i$. A bridge b is called a *proper* bridge of $\mathcal{P}_i$ if one of the end points of b belongs to $\mathcal{P}_i$; otherwise it is called a *passing* bridge. In the latter case no point of $\mathcal{P}_i$ belongs to the boundary of the lower hull of $D(\mathcal{L}_-)$. This structure is recursively applied to all $\mathcal{P}_i$.

Our algorithm proceeds according to the recursive structure of the bridge tree. In the first step it is our aim to sort out the two sets of consecutive points $\mathcal{P}_{i_0}$ and $\mathcal{P}_{i_1}$ that contain the vertices that are incident to t_1 and t_2. We then apply the procedure recursively to $\mathcal{P}_{i_0}$ and $\mathcal{P}_{i_1}$.

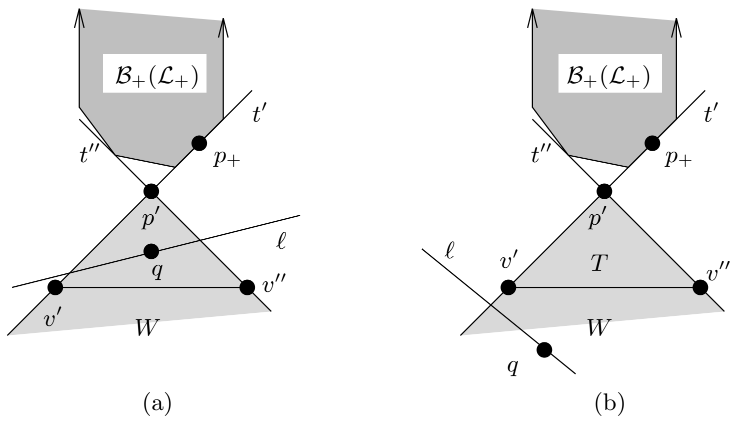

Fig. 7: q is not in the interior of W.

Let $v_1, \ldots, v_k$, $k \le 2n^{1/6}$, be the vertices that belong to the bridges on the first level of the bridge tree. With $n^{1/6} \cdot n^{2/3} = n^{5/6}$ processors we can find, for each of the k vertices, the tangents to the upper hull of $D(\mathcal{L}_+)$ in time $O(\log n/(\log n^{2/3}+1)+1) = O(1)$ with the help of the bridge tree for the upper hull of $D(\mathcal{L}_+)$. With $n^{1/3}$ processors we can check in constant time which tangents have all the vertices $v_1, \ldots, v_k$ to one side. Let t' and t'' be these two tangents and v' and v'' be the vertices incident to t' and t'', respectively, with v' to the left of v''. The lines t' and t'' form four wedges. Let p' be the intersection point of t' and t'' and let W be the wedge that contains v' and v'' and W' the wedge opposite to W.

Lemma 4.2. *The support points of the tangents t_1 and t_2 at the lower hull of $D(\mathcal{L}_-)$ are not contained in the interior of W.*

Proof. To see this let q be first a point in the interior of the triangle T spanned by p', v' and v''. Since the line segment from v' to v'' is contained in the lower hull of $D(\mathcal{L}_-)$, any line ℓ that is incident to q and that is the dual of a point in $\mathcal{K}$ does not intersect $\overline{v'v''}$ and, hence, intersects the line segments from v' to p' and from v'' to p'. This implies that W' is entirely above ℓ and, hence, ℓ is not a tangent to $D(\mathcal{L}_+)$ (see Fig. 7a).

Now consider a point q in W below the line segment from v' to v''. We claim that neither t_1 nor t_2 intersects q. For consider a line ℓ through q. Either ℓ intersects t' below v' or t'' below v''. In the first case v' and the intersection point p_+ of t' with $D(\mathcal{L}_+)$ are on the same side of ℓ and, hence, ℓ cannot be t_1 or t_2. An analogous statement holds in the second case (see Fig. 7b). Therefore, the intersection points of t_1 and t_2 and $D(\mathcal{L}_-)$ are not in the interior of W. □

A consequence of Lemma 4.2 is that we only have to consider points that are outside W. Let b' be the bridge incident to v'. Suppose that b' starts at vertex u' and ends at v'. Let w' be the next vertex after v' in the top level of the bridge tree that belongs to a bridge. We denote the x-coordinate of a point p by p_x.

LEMMA 4.3. *If p is a point above t', then p_x is between v'_x and w'_x.*

PROOF. Let x' be a vertex of $\mathcal{B}_-(\mathcal{L}_-)$ after w'. If x' is above t', then w' is entirely below the line segment from v' to x' and, hence, does not belong to the boundary of the lower hull of $D(\mathcal{L}_-)$ in contradiction to our choice of w'. A similar argument holds if x' is before u'. And if there exists a vertex x' between u' and v' that is above t', then b' is below x' and, hence, does not belong to the boundary of the lower hull of $D(\mathcal{L}_-)$. □

Of course, we can argue in a similar way if b' starts in v'. Furthermore, b'', v'', and t'' can be dealt with analogously.

If v' belongs to $\mathcal{P}_{i'}$, then our observations imply that we can recur on the $O(n^{5/6})$ vertices of $\mathcal{P}_{i'}$ in order to find t_1 and similarly for t_2. Since we reduce the exponent of the number of vertices we have to look at with each recursion step by a factor of 5/6, the total time needed to find t_1 and t_2 is $O(\log\log n)$. If we stop the recursion after, say, ten levels, then the number of vertices to be considered has been reduced to $O(n^{1/6})$ and the problem can be solved directly in constant time as in the first step. Hence, in fact, we need only constant time to compute t_1 and t_2. If we do not find the two tangents t_1 and t_2, then the upper hull of $D(\mathcal{L}_+)$ intersects the lower hull of $D(\mathcal{L}_-)$ and the kernel of $\mathcal{C}$ is empty. In particular, this implies that we can check in time $O(\log\log n)$ if $\mathcal{C}$ is starshaped or not.

With the help of t_1 and t_2 we also obtain a more accurate representation of $\mathcal{K}$ in the dual plane. To this end let $q_{i,+}$ be the intersection point of $\mathcal{B}_+(\mathcal{L}_+)$ with t_i and $q_{i,-}$ be the intersection point of $\mathcal{B}_-(\mathcal{L}_-)$ with t_i, for $i = 1, 2$. We denote the part of the boundary of $\mathcal{B}_+(\mathcal{L}_+)$ between $q_{1,+}$ and $q_{2,+}$ by $\mathcal{B}^*_+(\mathcal{L}_+)$ and the part of the boundary of $\mathcal{B}_-(\mathcal{L}_-)$ between between $q_{1,-}$ and $q_{2,-}$ by $\mathcal{B}^*_-(\mathcal{L}_-)$. Let $r_{i,+}$ be the ray that is contained in t_i, starts at $q_{i,+}$, and that does not contain $q_{i,-}$, for $i = 1, 2$; similarly, let $r_{i,-}$ be the ray that is contained in t_i, starts at $q_{i,-}$, and that does not contain $q_{i,+}$, for $i = 1, 2$ (see Fig. 8).

LEMMA 4.4. *The points in $\mathcal{K}$ are duals of lines that do not intersect $\mathcal{B}^*_+(\mathcal{L}_+)$, $\mathcal{B}^*_-(\mathcal{L}_-)$, $r_{1,+}$, $r_{2,+}$, $r_{1,-}$, or $r_{2,-}$.*

PROOF. Let ℓ be a line that does not intersect $r_{1,+} \cup \mathcal{B}^*_+(\mathcal{L}_+) \cup r_{2,+}$. ℓ is below $r_{1,+} \cup \mathcal{B}^*_+(\mathcal{L}_+) \cup r_{2,+}$ and, hence, below $\mathcal{B}_+(\mathcal{L}_+)$. Similaly, if ℓ does not intersect $r_{1,-} \cup \mathcal{B}^*_-(\mathcal{L}_-) \cup r_{2,-}$, then ℓ is above $r_{1,-} \cup \mathcal{B}^*_-(\mathcal{L}_-) \cup r_{2,-}$ and, hence, above $\mathcal{B}_-(\mathcal{L}_-)$. This implies that ℓ is the dual of a point $\mathcal{K}$.

If ℓ intersects $r_{1,+}$ and is above $q_{1,+}$, then ℓ intersects $\mathcal{B}_+(\mathcal{L}_+)$. If ℓ intersects $r_{1,+}$ and is below $q_{1,+}$, then ℓ intersects $\mathcal{B}_-(\mathcal{L}_-)$ since any line that is below $q_{1,+}$ and has a steeper slope than t_1 intersects $\mathcal{B}_-(\mathcal{L}_-)$. Similar arguments hold if ℓ intersects $r_{2,+}$, $r_{1,-}$, or $r_{2,-}$. □

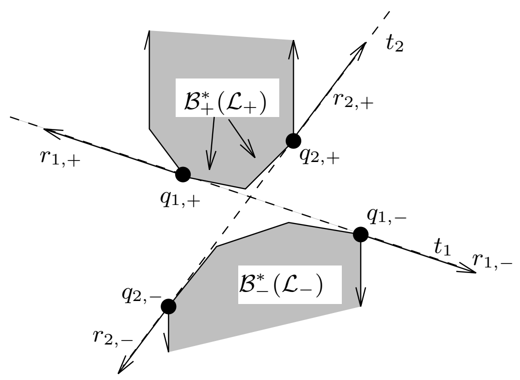

Fig. 8: Defining $q_{i,+}$, $r_{i,+}$, $q_{i,-}$, $r_{i,-}$, $\mathcal{B}^*_+(\mathcal{L}_+)$, and $\mathcal{B}^*_-(\mathcal{L}_-)$.

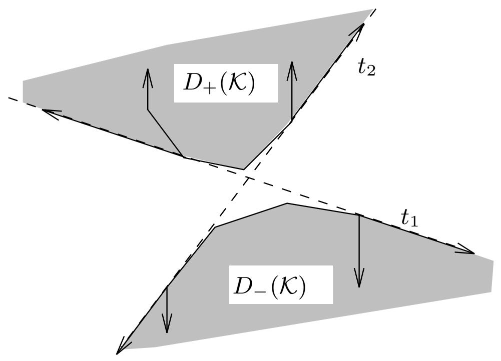

Fig. 9: The dual of $\mathcal{K}$.

We denote the region above $r_{1,+} \cup \mathcal{B}^*_+(\mathcal{L}_+) \cup r_{2,+}$ by $D_+(\mathcal{K})$ and the region below $r_{1,-} \cup \mathcal{B}^*_-(\mathcal{L}_-) \cup r_{2,-}$ by $D_-(\mathcal{K})$. $\mathcal{K}$ is the set of points whose dual line do not intersect the interior of $D_+(\mathcal{K}) \cup D_-(\mathcal{K})$ (see Fig. 9).

In order to answer a query of type (ii), that is in order to compute the tangent to $\mathcal{K}$ through a given point p, we just note that the intersection points q_1 and q_2 of $D(p)$ with the boundary of $D_+(\mathcal{K}) \cup D_-(\mathcal{K})$ are the duals of the tangents to $\mathcal{K}$. To see this just note that $D(q_1)$ is a tangent to $\mathcal{K}$ through p since q_1 is on the boundary of $D(\mathcal{K})$ and $D(p)$ is incident to q_1.

Finally, in order to answer a query of type (iv) we note that the set of parallel lines in the primal plane with slope d can be represented by the vertical line ℓ_d through the point $(d, 0)$ in the dual plane. Hence, we have to intersect ℓ_d with the boundaries of $D_+(\mathcal{K})$ and $D_-(\mathcal{K})$.

Clearly, all of the above queries require only a constant number of queries to the bridge tree of the upper hull of $D(\mathcal{L}_+)$ and the lower hull of $D(\mathcal{L}_-)$

and, hence, the query time is still $O(\log n/(\log p+1)+1)$.

5. Conclusions

We have presented an $O(\log\log n)$ time algorithm to compute the kernel of a polygonal chain with $O(n/\log\log n)$ processors on a COMMON CRCW-PRAM. Our approach is based on the algorithm by Cole and Goodrich [1992]. We show how to avoid parallel prefix sum computation which is needed in their algorithm. Since our intersection of halfplanes algorithm makes use of dualization and the new optimal $O(\log\log n)$ convex hull algorithm of Wagener [1992], we also address the question of how to answer the following four queries for the kernel of a polygon P in the primal plane given a query structure in the dual plane.

(i) Given a point q, test whether q is in the $kernel(P)$.

(ii) Given a point q, report the tangents to the $kernel(P)$ passing through q.

(iii) Given a line ℓ, report the intersection of ℓ with $kernel(P)$.

(iv) Given a direction d, report the extremal points of $kernel(P)$ in direction d.

All of these queries can be answered in time $O(\log n/(\log p+1)+1)$ if p processors are available.

6. Acknowledgements

I would like to thank the unknown referee whose valuable comments and suggestions have considerably improved the presentation of the paper as well as led to the inclusion of some new results.

References

M. J. Atallah, D. Z. Chen, and H. Wagener. Optimal parallel algorithm for visibility of a simple polygon from a point. *Journal of the ACM*, **38**:516–553, 1991.

P. Beame and J. Hastad. Optimal bounds for decision problems no the CRCW PRAM. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 83–93, 1987.

O. Berkman, B. Schieber, and U. Vishkin. Some doubly logarithmic optimal parallel algorithms based on finding all nearest smaller values. Technical Report UMIACS-TR-88-79, Institute for Advance Computer Studies, University of Maryland, College Park, MD, 1988.

D. Z. Chen. An optimal parallel algorithm for detecting weak visibility of a simple polygon. In *Proc. 8th ACM Symposium on Computational Geometry*, pages 63–72, 1992.

R. Cole and M. Goodrich. Optimal parallel algorithms for polygon and point-set problems. *Algorithmica*, **7**:3-23, 1992.

H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, 1987.

D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *Journal of the ACM*, **26**(3):415–421, 1979.

J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

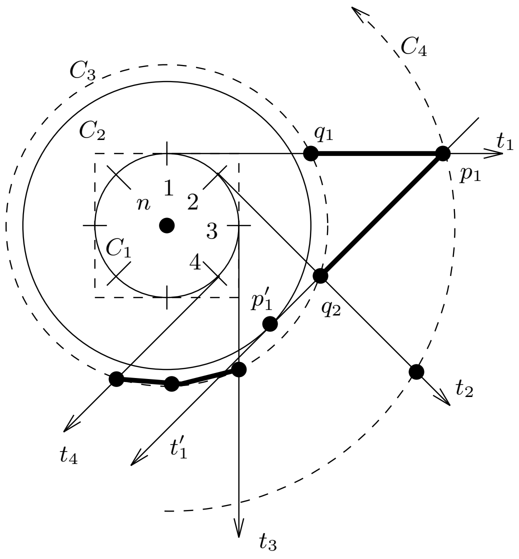

Fig. 10: Illustrating the construction of polygon P.

F.P. PREPARATA AND M.I. SHAMOS. *Computational Geometry — an Introduction*. Springer Verlag, 1985.

P. RAGDE. The parallel simplicity of compaction and chaining. In *Proc. 17th International Colloquium on Automata, Languages, and Programming*, pages 744–751, 1990.

H. WAGENER. Optimal parallel hull construction for simple polygons in $O(\log\log n)$ time. In *Proc. 33rd IEEE Syposium on Foundations of Computer Science*, pages 593–599, 1992.

Appendix A. A Lower Bound to Compute the Size of the Kernel

As was indicated before the lower bound is obtained by reducing the parity problem to the problem of computing the number of edges of the kernel of a simple polygon. So let $b_1, \ldots, b_n$ be a sequence of n bits. W.l.o.g. we assume that $n = 4k$, for some $k > 1$. We set the bits b_1, b_{k+1}, b_{2k+1}, and b_{3k+1} to 1. If the parity of the bit sequence $b_1 b_{k+1} b_{2k+1} b_{3k+1}$ was odd before, then we invert bit b_2.

We now construct a polygon P consisting of $2n$ edges $e_1, \ldots, e_{2n}$ such that edge e_i is relevant if and only if $i = 2j - 1$ and $b_j = 1$, for some $1 \leq j \leq n$. The construction is illustrated in Fig. 10. Let C_1 be a unit circle that is divided by n points into equal parts. At each point i we construct the tangent t_i to C_1. Let C_4 be a circle that is concentric to C_1 and that has radius $r_4 > 2$ and let p_i be the intersection point of C_4 with t_i. Furthermore, let C_2 be the circle of radius 2 which is concentric to C_1. Consider the tangent t_i' to C_2 that goes through p_i. The intersection points q_i of t_i' with t_{i+1} all lie on a circle C_3 that is also concentric to C_1. The points q_i are sorted counterclockwise on C_3 if the points on C_1 are sorted counterclockwise.

If we choose $r_4 > 2\sqrt{1 + 2\sqrt{3}}$, then a simple calculation shows that t_{i+k} intersects t'_i in the line segment between p_i and the intersection point p'_i of t'_i with C_2; this implies in particular that t_{i+1} intersects t'_i in the line segment between p_i and p'_i. Hence, if r_3 is the radius of C_3 and $r_4 > 2\sqrt{1 + 2\sqrt{3}}$, then $r_2 < r_3 < r_4$ as drawn in Fig. 10. Note that p_i and q_i can be computed independently, for each $1 \leq i \leq n$.

We now construct P. The points q_i are a subset of the vertices of P. If bit b_i is 1, then edge e_{2i-1} consists of the line segment from q_i to p_i and edge e_{2i} consists of the line segment from p_i to q_{i+1}. This is shown for q_1, p_1, and q_2 in Fig. 10. If bit b_i is 0, then edge e_{2i-1} consists of the line segment from q_i to some point q'_i on C_3 between q_i and q_{i+1} and edge e_{2i} consists of the line segment from q'_i to q_{i+1}. This is shown for q_3 and q_4 in Fig. 10. Note that only the halfplanes corresponding to the edges e_{2j-1} with $b_j = 1$ intersect the smallest axis-parallel square that contains C_1. Since $b_{jk+1} = 1$, for $0 \leq j \leq 3$, this square contains $kernel(P)$. Furthermore, $h^+(e_{2j-1})$ contains a neighbourhood of all the points $1, \ldots, n$ except for point j if $b_j = 1$. This implies that edge e_{2j-1} is relevant if and only if $b_j = 1$. Furthermore, there are no other relevant edges. Hence, the the number of edges of the kernel of P is the number of 1's in $b_1, \ldots, b_n$ and the parities are the same.