An $O(\log\log n)$ Time Algorithm to Compute the Kernel of a Polygon*

S. Schuierer†

Abstract

The kernel of a polygon $\mathbf{P}$ is the set of all points that see the interior of $\mathbf{P}$. It can be computed as the intersection of the halfplanes that are to the left of the edges of $\mathbf{P}$. 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 $\mathbf{P}$ 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.

1 Introduction

Visibility problems play an important role in computational geometry. Given a simple polygon $\mathbf{P}$ in the plane we say two points p and q in $\mathbf{P}$ are *visible* from each other or *see* each other if the line segment $\overline{pq}$ is contained in $\mathbf{P}$. The *kernel of* $\mathbf{P}$ $kernel(\mathbf{P})$ is then defined as the set of all points that see all the other points in $\mathbf{P}$. It can be easily shown that $kernel(\mathbf{P})$ is the intersection of all the halfplanes that lie to the left of the polygon's edges given a counterclockwise orientation of $\mathbf{P}$ [6].

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 [6] or an $O(\log n)$ time algorithm for a CREW-PRAM with $n/\log n$ processors [4]. 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 [1] and an optimal $O(\log n)$ time CREW-PRAM algorithm for detecting weak visibility of a simple polygon [3].

Our approach is very similar to the one used in [4]. 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.

*This work was supported under a Deutsche Forschungsgemeinschaft Grant, Project "Datenstrukturen", Ot 64/5-4.

†Institut für Informatik, Universität Freiburg, Rheinstr. 10, D-79110 Freiburg, FRG, email: schuierer@informatik.uni-freiburg.de

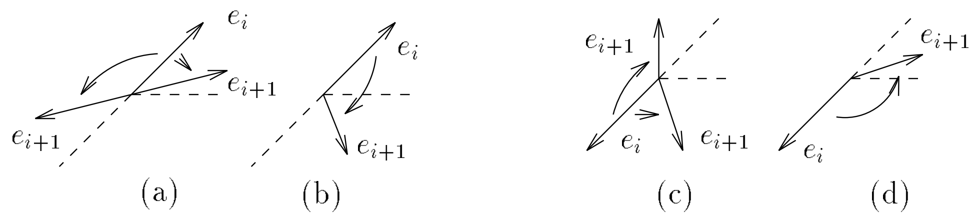

Figure 1: *The definition of* $incr(e_i, e_{i+1})$.

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 second 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. In Section 4 we consider the problem of computing these sequences of halfplanes in parallel. In Section 5 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 $\mathbf{P}$ is the intersection of all the halfplanes that are to the left of its edges given a counterclockwise orientation of $\mathbf{P}$. 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 $\bigcap_{1\le i\le n} h^+(e_i)$. Hence, if $\mathcal{C}$ is 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 to the left from e_i and negative if e_{i+1} turns to the right.

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

(a) If $\Theta(e_i) \le \pi$ and $\Theta(e_{i+1}) < \Theta(e_i) + \pi$, then $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i)$.

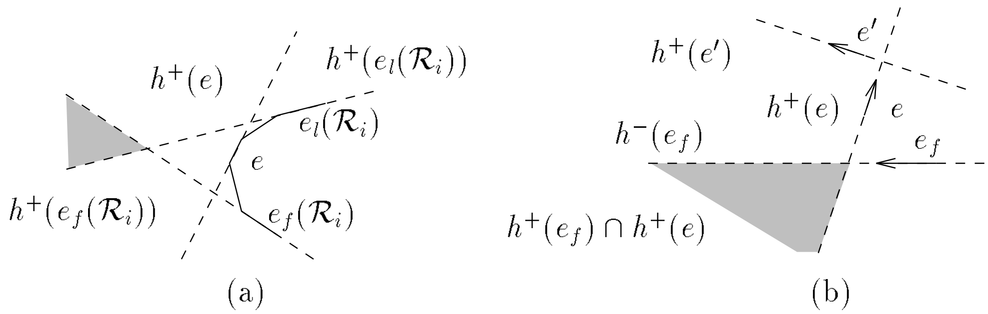

Figure 2: *An edge of a reflex or convex chain is irrelevant.*

(b) If $\Theta(e_i) \leq \pi$ and $\Theta(e_{i+1}) > \Theta(e_i) + \pi$, then $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i) - 2\pi$.

(c) If $\Theta(e_i) > \pi$ and $\Theta(e_{i+1}) > \Theta(e_i) - \pi$, then $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i)$.

(d) If $\Theta(e_i) > \pi$ and $\Theta(e_{i+1}) < \Theta(e_i) - \pi$, then $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i) + 2\pi$.

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

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

3 Computing the Kernel of a Polygonal Chain

In this section we show that there exist sorted sequences of edges S_1 and S_2 that contain all of the edges that are "relevant" for the computation of $kernel(\mathcal{C})$. We say edge e is *relevant* if the line through e intersects $kernel(\mathcal{C})$ in more than one point. Our approach is analogous to the one presented in [4] though the proof we present is different.

Consider two consecutive edges e_i and e_{i+1} of $\mathcal{C}$ that meet in vertex v_{i+1}. Vertex v_{i+1} is said to be *convex* if e_{i+1} is in $h^+(e_i)$, otherwise it is called *reflex*. A subchain $\mathcal{C}'$ of $\mathcal{C}$ is called *convex* (*reflex*) if all the vertices of $\mathcal{C}'$ except the first and last are convex (reflex).

We assume that $\mathcal{C}$ consists of the maximal reflex chains $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$ and the maximal convex chains $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ starting with $\mathcal{R}_1$ which is followed by $\mathcal{C}_1$ which in turn is followed by $\mathcal{R}_2$ and so on till we reach the last chain $\mathcal{C}_m$. Clearly, it is possible to decompose any polygonal chain into such a sequence of convex and reflex chains where the first and/or last chain $\mathcal{R}_1$ resp. $\mathcal{C}_m$ may be empty. For a chain $\mathcal{C}$, we denote the first edge by $e_f(\mathcal{C})$ and the last edge by $e_l(\mathcal{C})$. Note that we always have $e_l(\mathcal{R}_i) = e_f(\mathcal{C}_i)$ and $e_l(\mathcal{C}_i) = e_f(\mathcal{R}_{i+1})$.

We start with the following observation about edges of reflex chains.

Observation 3.1 *If e is an edge of $\mathcal{R}_i$ between $e_f(\mathcal{R}_i)$ and $e_l(\mathcal{R}_i)$ and $\Theta_{\mathcal{C}}(e_f(\mathcal{R}_i)) - \Theta_{\mathcal{C}}(e_l(\mathcal{R}_i)) < \pi$, then e is not relevant (see Figure 2a).*

Next we consider the relevant edges among the convex chains.

Lemma 3.2 *If $\mathcal{C}$ is a polygonal chain consisting of reflex chains $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$ and convex chains $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ with $e_f = e_f(\mathcal{C})$ such that*

(i) there is no edge $e \in \mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_m$ with $\Theta_{\mathcal{C}}(e) > \Theta_{\mathcal{C}}(e_f)$ and

(ii) there is no edge $e \in \mathcal{C}$ with $\Theta_{\mathcal{C}}(e_f) - \Theta_{\mathcal{C}}(e) \geq \pi$,

then $kernel(\mathcal{C})$ equals the intersection of $h^+(e_f)$ with the halfplanes corresponding to the last edges of the reflex chains $\mathcal{R}_i$, $1 \leq i \leq m$.

Proof: W.l.o.g. we can assume that $\Theta(e_f) = \pi$. Our first claim is that $\mathcal{C}$ is y-monotone. In order to show this we prove that $\Theta_{\mathcal{C}}(e)$ is in $(0, \pi)$, for all $e \in \mathcal{C}$. Since the angles of the edges along a convex chain are increasing, the maximum turning angle is achieved at some edge e_{max} with $e_{max} = e_l(\mathcal{C}_{i_1})$, $1 \leq i_1 \leq m$; the angles along a reflex chain are decreasing, so the minimum is achieved at some edge e_{min} with $e_{min} = e_l(\mathcal{R}_{i_2})$, $1 \leq i_2 \leq m$. By Assumption (i) $\Theta_{\mathcal{C}}(e_{max})$ is less than $\Theta_{\mathcal{C}}(e_f) = \pi$ and, furthermore, by Assumption (ii) $\Theta_{\mathcal{C}}(e_{min}) > \Theta_{\mathcal{C}}(e_f) - \pi = 0$. This immediately implies that $\mathcal{C}$ is y-monotone.

We have already argued that an edge of a reflex chain $\mathcal{R}$ that is neither at the end nor at the beginning of $\mathcal{R}$ is irrelevant for the computation of $kernel(\mathcal{C})$. So let e' be an edge of the convex chain $\mathcal{C}_j$ and let e be the first edge of $\mathcal{C}_j$. For illustration refer to Figure 2b. Since e' and e belong to the same convex chain, e is in $h^+(e')$ and since $\mathcal{C}$ is monotone, e is above the line through e_f. Hence, e belongs to the wedge formed by $h^+(e')$ and $h^-(e_f)$. Furthermore, $0 < \Theta_{\mathcal{C}}(e) \leq \Theta_{\mathcal{C}}(e')$ and, therefore, the line ℓ' through e' intersects the line through e_f in $h^-(e)$ and ℓ' does not intersect $h^+(e) \cap h^+(e_f)$. Since $h^+(e) \cap h^+(e') \cap h^+(e_f)$ is non-empty, $h^+(e')$ contains $h^+(e) \cap h^+(e_f)$ completely. Hence, $h^+(e) \cap h^+(e') \cap h^+(e_f) = h^+(e) \cap h^+(e_f)$ and e' is irrelevant. Since e is also the last edge of $\mathcal{R}_j$, this implies that all edges of $\mathcal{C}$ are irrelevant except for e_f and the end edges of the reflex chains. $\square$

With the help of the above lemma we can now define a sequence S_1 of edges that contains "most" of the relevant edges of $\mathcal{C}$. To this end let $\theta_i = \max_{1 \leq j \leq i} \Theta_{\mathcal{C}}(e_j)$ and let S_1 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_1 is the set of edges whose turning angle is larger than the turning angle of any previous edge.

Lemma 3.3 *If $\mathcal{C}$ is a polygonal chain consisting of reflex chains $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$ and convex chains $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ such that there is no edge e of $\mathcal{C}$ between two consecutive edges e' and e'' in S_1 with $\Theta_{\mathcal{C}}(e') - \Theta_{\mathcal{C}}(e) \geq \pi$, then $kernel(\mathcal{C})$ is the intersection of $h^+(e')$, with $e' \in S_1$, and $h^+(e_l(\mathcal{R}_i))$, $1 \leq i \leq m$.*

Proof: We prove by induction on the edges of $\mathcal{C}$ that $\bigcap_{1 \leq j \leq i} h^+(e_j)$ contains the intersection of the halfplanes $h^+(e_r)$, with $e_r \in S_1$ and $r \leq i$, and $h^+(e_l(\mathcal{R}_k))$, with $1 \leq k \leq m$. The claim is obvious if we consider only e_1. Hence, consider edge e_i and assume that the claim has been shown for the subchain of $\mathcal{C}$ consisting of the edges $e_1, \ldots, e_{i-1}$. As e_i is irrelevant if it doesn't belong to a convex chain, we can assume that e_i is part of convex chain $\mathcal{C}_j$. If $\Theta_{\mathcal{C}}(e_i) > \theta_{i-1}$, then e_i belongs to S_1 and we are done. Otherwise, let e_k be the edge in S_1 with largest $k < i$. The induction hypothesis implies that

$$\bigcap_{1 \leq j \leq i} h^+(e_j) \supseteq \bigcap_{e_j \in S_1, j < k} h^+(e_j) \cap \bigcap_{1 \leq j \leq m} h^+(e_l(\mathcal{R}_j)) \cap \bigcap_{k \leq j \leq i} h^+(e_j).$$

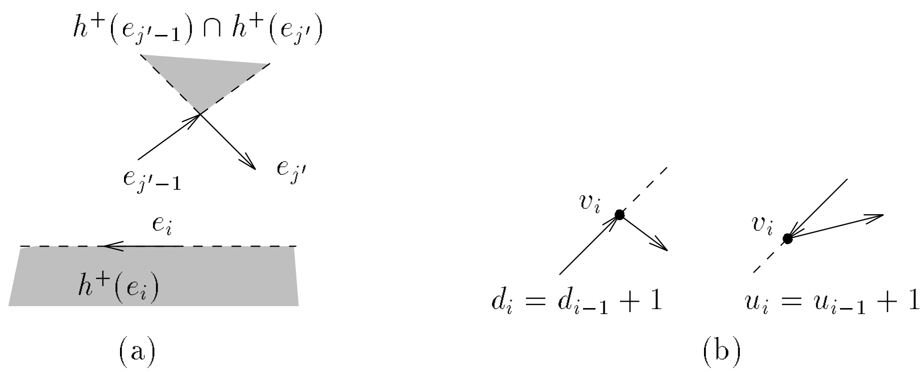

Figure 3: *The kernel of $\mathcal{C}$ is empty if $\Theta_{\mathcal{C}}(e_i) - \Theta_{\mathcal{C}}(e_{j'}) \geq \pi$.*

By Lemma 3.2 $\bigcap_{k \leq j \leq i} h^+(e_j)$ contains

$$h^+(e_k) \cap \bigcap_{1 \leq j \leq m} h^+(e_l(\mathcal{R}_j))$$

since $\Theta_{\mathcal{C}}(e_j) \leq \Theta_{\mathcal{C}}(e_k)$, for $k < j \leq i$, and there is no edge e_j, $k < j \leq i$, with $\Theta_{\mathcal{C}}(e_k) - \Theta_{\mathcal{C}}(e_j) \geq \pi$ which completes the induction step and, hence, the proof. $\square$

Lemma 3.4 *Let e_i and $e_{i'}$ be two consecutive edges in S_1. 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 Figure 3a). $\square$

Corollary 3.5 *If $\mathcal{C}$ is a polygonal chain consisting of reflex chains $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$ and convex chains $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ and $\Theta_{\mathcal{C}}(e_f(\mathcal{C})) - \Theta_{\mathcal{C}}(e_l(\mathcal{R}_i)) \geq \pi$, for some $1 \leq i \leq m$, then $kernel(\mathcal{C})$ is empty.*

Proof: Let e_j be the first edge with $\Theta_{\mathcal{C}}(e_f(\mathcal{C})) - \Theta_{\mathcal{C}}(e_j) \geq \pi$ and e_k be the edge in S_1 with maximum $k < j$. Since $\Theta_{\mathcal{C}}(e_k) > \Theta_{\mathcal{C}}(e_f(\mathcal{C}))$, we have $\Theta_{\mathcal{C}}(e_k) - \Theta_{\mathcal{C}}(e_j) \geq \pi$. By Lemma 3.4 this implies that $kernel(\mathcal{C})$ is empty. $\square$

Now suppose we mirror $\mathcal{C}$ at the y-axis and revert the orientation of the edges and call this new chain $\mathcal{C}'$. 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 $\mathcal{C}$. Similarly, an edge belongs to a reflex (convex) chain of $\mathcal{C}'$ if and only if its preimage belongs to a reflex (convex) chain of $\mathcal{C}$. Hence, we can define a sequence S_2 of edges of $\mathcal{C}'$ analogously to the sequence S_1 of $\mathcal{C}$ only now starting with the image of e_n. But the image of the *last* edge of a reflex chain of $\mathcal{C}$ is now the *first* edge of a reflex chain of $\mathcal{C}'$ and vice versa. If we take into account that we have $\bigcap_{e \in S_1} h^+(e) \subseteq \bigcap_{1 \leq i \leq m} e_f(\mathcal{R}_i)$, then, if we mirror back and again change the orientation, we have analogously $\bigcap_{e \in S_2} h^+(e) \subseteq \bigcap_{1 \leq i \leq m} e_l(\mathcal{R}_i)$. Therefore, we obtain

$$\begin{aligned} kernel(\mathcal{C}) &= \bigcap_{e \in S_1} h^+(e) \cap \bigcap_{1 \leq i \leq m} h^+(e_l(\mathcal{R}_i)) \\ &= \bigcap_{e \in S_1} h^+(e) \cap \bigcap_{e \in S_2} h^+(e). \end{aligned} \tag{1}$$

4 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. With the observations from the previous section it seems that the following approach to computing the kernel is a reasonable try.

1. Compute S_1 and S_2;
2. Merge S_1 and S_2 into a single sequence S;
3. Intersect the halfplanes in S;

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

4.1 Computing S_1

The computation of S_1 consists of two steps. First we have to compute $\Theta_{\mathcal{C}}(e_i)$, and then θ_i, for $1 \leq i \leq n$. Recall that $\theta_i = \max_{1\leq j\leq i} \Theta_{\mathcal{C}}(e_j)$. 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 $\Theta(e_i) \leq \pi$ and $\Theta(e_{i+1}) > \Theta(e_i) + \pi$. Similarly, we say vertex v_{i+1} is an *upturn* if $\Theta(e_i) > \pi$ and $\Theta(e_{i+1}) < \Theta(e_i) - \pi$ (see Figure 3b). Let the number of upturns up to and including vertex v_i be u_i and the number of downturns d_i.

Lemma 4.1 *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 $\Theta_{\mathcal{C}}(e_{i+1}) = \Theta_{\mathcal{C}}(e_i) + incr(e_i, e_{i+1})$. In order to do so we distinguish four cases depending on $\Theta(e_i)$.

1. $\Theta(e_i) \leq \pi$ and $\Theta(e_{i+1}) < \Theta(e_i) + \pi$.
   By definition we have $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i)$, $d_{i+1} = d_i$ and $u_{i+1} = u_i$. Hence,
$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i+1}) &= \Theta_{\mathcal{C}}(e_i) + incr(e_i, e_{i+1})\\ &= \Theta(e_i) + (u_i - d_i)2\pi + \Theta(e_{i+1}) - \Theta(e_i)\\ &= \Theta(e_{i+1}) + (u_{i+1} - d_{i+1})2\pi.\end{aligned}$$
2. $\Theta(e_i) \leq \pi$ and $\Theta(e_{i+1}) > \Theta(e_i) + \pi$.
   We have $incr(e_i, e_{i+1}) = \Theta(e_{i+1}) - \Theta(e_i) - 2\pi$, $d_{i+1} = d_i + 1$, and $u_{i+1} = u_i$. Hence,
$$\begin{aligned}\Theta_{\mathcal{C}}(e_{i+1}) &= \Theta_{\mathcal{C}}(e_i) + incr(e_i, e_{i+1})\\ &= \Theta(e_i) + (u_i - d_i)2\pi + \Theta(e_{i+1}) - \Theta(e_i) - 2\pi\\ &= \Theta(e_{i+1}) + (u_{i+1} - (d_i + 1))2\pi\\ &= \Theta(e_{i+1}) + (u_{i+1} - d_{i+1})2\pi.\end{aligned}$$

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

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

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

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

which completes the proof. □

Hence, the computation of $\Theta_{\mathcal{C}}(e_i)$ can be reduced to computing $t_i = u_i - d_i$. Assume that t_i is in the interval $[-k, k]$, $1 \leq k \leq n$. In the following we show that if $kernel(\mathcal{C})$ is non-empty, then $k = 1$.

In order to compute t_i, for all $1 \leq i \leq n$, we make use of an optimal $O(\log\log n)$ CRCW-PRAM algorithm that solves the *all nearest smaller values* (*ANSV*) problem which is defined as follows [2]. Given a sequence of n elements $(a_1, \ldots, a_n)$ 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 element with maximum $j < i$ such that $a_j < a_i$ 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 up- and downturns. The first step is that for each downturn we find the right closest up- or downturn. In order to do so, processor i writes $n - i$ into array cell a_i if v_i is an up- or downturn and $n + 1$ otherwise. Clearly, an application of the ANSV-algorithm yields the closest right up- or downturn, for each vertex. If v_i is a downturn and its right match v_j an upturn, then v_i and v_j are called *matched*. If there is no up- or downturn to the right of downturn v_i, then we consider v_i as matched nevertheless. Now we compute, for each matched downturn v_i, its left closest unmatched up- or downturn. Each of these steps can be carried out in $O(\log\log n)$ steps with $n/\log\log n$ processors [2].

We claim that for the remaining (unmatched) vertices t_i is contained in the interval $[-k+1, k]$. The proof is by contradiction. If there is no unmatched downturn, then $t_i \geq 0$ and the claim holds. Otherwise, we observe that the minima of the sequence (t_i) are achieved at downturns. So assume that there is an unmatched downturn v_i such that $t_i = -k$. Its closest up- or downturn v_j to the right exists and is a downturn since otherwise v_i and v_j were matched. But this implies that $t_j = -k - 1$ in contradiction to our assumption that $t_j \in [-k, k]$.

The above procedure is now iterated one more time, matching upturns with downturns and finding the closest left unmatched up- or downturn, for each matched downturn. With this iteration the possible range for the t_i values of the remaining up- or downturns is again reduced by one. If there are any unmatched up- or downturns left after the iteration, then this implies that either there is an i with $t_i > 1$ or we started out with a $t_i < -1$. In the first case we have $\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) + t_i 2\pi > 4\pi$ which implies that $kernel(\mathcal{C})$ is empty [4, 6]. In the second case we have $\Theta_{\mathcal{C}}(e_i) = \Theta(e_i) - t_i 2\pi < -2\pi$ and, hence, $\Theta_{\mathcal{C}}(e_f(\mathcal{C})) - \Theta_{\mathcal{C}}(e_i) > 2\pi$ which implies by Corollary 3.5 that $kernel(\mathcal{C})$ is also empty in this case.

Hence, we can assume that all t_i's are contained in $[-1, 1]$ and the above procedure is executed at most two times. Any downturn that is matched in the second step is known to have a t_i value of 0 and any upturn is known to have a t_i value of 1. Since each matched pair P' in the first step has a pointer to an unmatched pair P the t-value of P' can be computed in constant time per vertex. In order to obtain the t_i values for all vertices we just have to apply the ANSV-algorithm one more time so that each vertex can find its left closest up- or downturn. All of these operations can be carried out in time $O(\log\log n)$.

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 4.1. In order to obtain S_1 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 [2].

Finally, we have to check if there is an edge e between two consecutive edges e' and e'' of S_1 with $\Theta_{\mathcal{C}}(e') - \Theta_{\mathcal{C}}(e) \geq \pi$ by Lemma 3.3. This can be done by yet another application of the ANSV-algorithm in order to find, for each edge, the closest edge in S_1 to the left and then compare the two turning angles. If there is one edge e with left closest neighbour e' in S_1 such that $\Theta_{\mathcal{C}}(e') - \Theta_{\mathcal{C}}(e) \geq \pi$, then $kernel(\mathcal{C})$ is empty.

The sequence S_2 can be computed and checked analogously.

4.2 Merging S_1 and S_2

The important property of S_1 and S_2 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 the merged sequence S seems possible. Unfortunately, the edges of S_1 and S_2 are scattered among the edges of $\mathcal{C}$ which we assume to stored in an array A. In order to compact the edges of S_1 and S_2 into a contiguous part of A we need at least $\Omega(\log|S_1|/\log\log n)$ time [7] which we cannot afford since S_1 (or S_2) 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_1, its left closest match $lcm_{S_1}(e)$ in S_1 and the same for edges not in S_2. We create two new arrays A_1 and A_2 with $A_i[j] = e_j$ if e_j is in S_i and $A_i[j] = lcm_{S_i}(e_j)$ otherwise. Note that $\bigcap_{e \in A_i} h^+(e) = \bigcap_{e \in S_i} h^+(e)$, for $i = 1, 2$. Hence, we can merge the two arrays A_1 and A_2 into one array A' of size $2n$ in time $O(\log\log n)$.

4.3 Intersecting S_1 and S_2

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

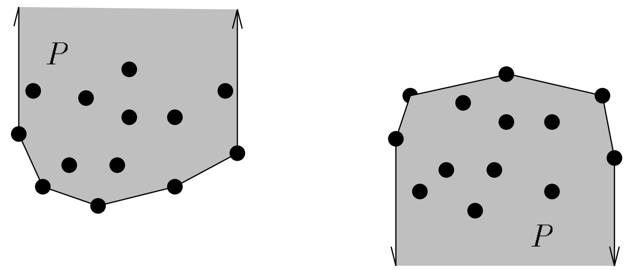

Figure 4: *The lower and upper hull of a point set P.*

to compute the intersection of the halfplanes associated with the edges in $E_+ \cup E_-$ we apply a dualization method as described below.

5 Halfplane Intersection and the Convex Hull

In this section we show how to reduce the intersection of halfplanes that are sorted according to 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 [5]

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

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 $\mathcal{L}_+$ 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 lower convex hull of $D(\mathcal{L}_+)$ is above $D(p)$. The *lower convex hull* of $D(\mathcal{L}_+)$ is defined as the set of points that are above the convex hull of $D(\mathcal{L}_+)$. Similarly, the *upper convex hull* of $D(\mathcal{L}_-)$ is defined as the set of points that are below the convex hull of $D(\mathcal{L}_-)$ (see Figure 4). We denote the boundary of the lower convex hull of $D(\mathcal{L}_+)$ by $\mathcal{H}_+(\mathcal{L}_+)$ and the boundary of the upper convex hull of $D(\mathcal{L}_-)$ by $\mathcal{H}_-(\mathcal{L}_-)$.

If we consider line ℓ' through the vertices v_1 and v_2 on $\mathcal{H}_+(\mathcal{L}_+)$, then all the points of $D(\mathcal{L}_+)$ are above ℓ' except v_1 and v_2 which lie on ℓ'. So ℓ' corresponds to a point p' in the original plane that is above all lines in $\mathcal{L}_+$ and is the intersection point of $D(v_1)$ and $D(v_2)$. In other words, p' is a vertex of $\bigcap \mathcal{H}_+$. This proves the following observation.

Observation 5.1 *The lines through the edges of $\mathcal{H}_+(\mathcal{L}_+)$ correspond to the vertices of $\bigcap \mathcal{H}_+$.*

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 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 $y = a_1 x + a_2$, 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 [8] 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; therefore, a special data structure—called the *bridge tree*—is used in [8]. 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 $\bigcap \mathcal{H}_-$.

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 lower hull of $D(\mathcal{L}_+)$ (or the upper hull of $D(\mathcal{L}_-)$).

(ii) Given a point q, report the tangents to the lower hull of $D(\mathcal{L}_+)$ (or the upper hull of $D(\mathcal{L}_-)$) passing through q.

(iii) Given a line ℓ, report the intersection of ℓ with the lower hull of $D(\mathcal{L}_+)$ (or the upper hull of $D(\mathcal{L}_-)$).

(iv) Given a direction d, report the extremal points of the lower hull of $D(\mathcal{L}_+)$ (or the upper hull of $D(\mathcal{L}_-)$) in direction d.

In the following it is our aim to show that the same types of queries can 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 lower 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 lower hull of $D(\mathcal{L}_+)$ nor the upper hull of $D(\mathcal{L}_-)$.

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

Lemma 5.2 *The line ℓ intersects $\mathcal{K}$ if and only if two of the four tangents of $D(\ell)$ to $\mathcal{H}_+(\mathcal{L}_+)$ and $\mathcal{H}_-(\mathcal{L}_-)$ do not intersect the interior of $\mathcal{H}_+(\mathcal{L}_+) \cup \mathcal{H}_-(\mathcal{L}_-)$.*

Proof: The line ℓ intersects $\mathcal{K}$ if and only if there are two points p_1 and p_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 $D(p_1)$ and $D(p_2)$ and $D(p_1)$ and $D(p_2)$ are tangents to $\mathcal{H}_+(\mathcal{L}_+)$ and/or $\mathcal{H}_-(\mathcal{L}_-)$. Furthermore, $D(p_1)$ and $D(p_2)$ belong to $D(\mathcal{K})$ and, hence, do not intersect the interior of $\mathcal{H}_+(\mathcal{L}_+)$ or $\mathcal{H}_-(\mathcal{L}_-)$. □

Since there are only four tangents through $D(\ell)$ to the lower hull of $D(\mathcal{L}_+)$ and the upper 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 Figure 5).

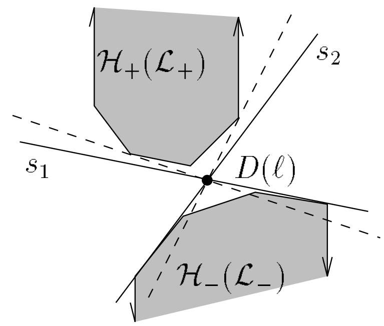

Figure 5: *The tangents to the lower hull of $D(\mathcal{L}_+)$ and the upper hull of $D(\mathcal{L}_-)$ correspond to the intersection points of ℓ.*

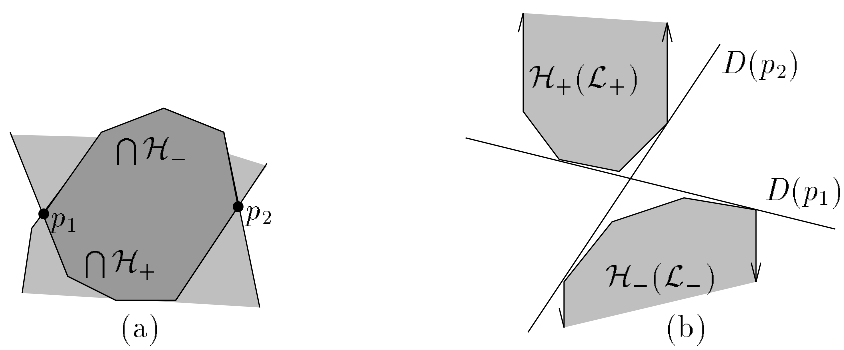

Figure 6: *The duality between $\bigcap\mathcal{H}$ and the lower hull of $D(\mathcal{L}_+)$ and the upper hull of $D(\mathcal{L}_-)$.*

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

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 lower hull of $D(\mathcal{L}_+)$ and the upper hull of $D(\mathcal{L}_-)$ (see Figure 6).

Note that the lower hull of $D(\mathcal{L}_+)$ is on a different side of t_1 (t_2) as the upper hull of $D(\mathcal{L}_-)$. In order to compute t_1 and t_2 we make use of the internal structure of the bridge tree for the the upper hull of $D(\mathcal{L}_-)$ which 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 upper 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 upper hull of $D(\mathcal{L}_-)$. This structure is recursively applied to all $\mathcal{P}_i$.

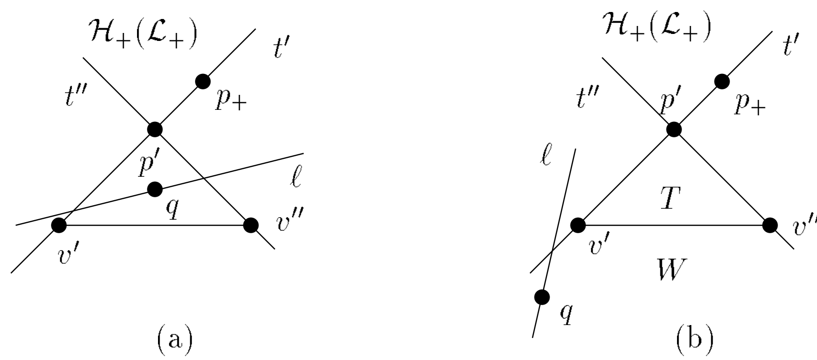

Figure 7: *q can neither be in T nor below T.*

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 vertices $\mathcal{P}_{i_0}$ 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 $\mathcal{P}_{i_1}$.

Let $v_1, \ldots, v_k$, $k \leq 2n^{1/6}$, 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 lower 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 lower hull of $D(\mathcal{L}_+)$. With $n^{1/3}$ processors we can check in constant time which tangents have all the k vertices 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 5.3 *The support points of the tangents t_1 and t_2 at the upper 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 upper 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 Figure 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 Figure 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 5.3 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.

Lemma 5.4 *If p is a point above t', then p_x is between v'_x and w'_x.*

Figure 8: *Only the vertices between v' and w' can be above t'.*

Proof: Let x' be a vertex of $\mathcal{H}_-(\mathcal{L}_-)$ 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 upper hull of $D(\mathcal{L}_-)$ in contradiction of our choice of w' (see Figure 8a). 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 upper hull of $D(\mathcal{L}_-)$ (see Figure 8b). □

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 lower hull of $D(\mathcal{L}_+)$ intersects the upper 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 $\mathcal{H}_+(\mathcal{L}_+)$ with t_i and $q_{i,-}$ be the intersection point of $\mathcal{H}_-(\mathcal{L}_-)$, for $i = 1, 2$. We denote the part of the boundary of $\mathcal{H}_+(\mathcal{L}_+)$ between $q_{1,+}$ and $q_{2,+}$ by $\mathcal{H}^*_+(\mathcal{L}_+)$ and the part of the boundary of $\mathcal{H}_-(\mathcal{L}_-)$ between between $q_{1,-}$ and $q_{2,-}$ by $\mathcal{H}^*_-(\mathcal{L}_-)$. 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 Figure 9).

Lemma 5.5 *The points in $\mathcal{K}$ are duals of lines that do not intersect $\mathcal{H}^*_+(\mathcal{L}_+)$, $\mathcal{H}^*_-(\mathcal{L}_-)$, $r_{1,+}$, $r_{2,+}$, $r_{1,-}$, or $r_{2,-}$.*

Proof: Let ℓ be a line that does not intersect $r_{1,+} \cup \mathcal{H}^*_+(\mathcal{L}_+) \cup r_{2,+}$. ℓ is below $r_{1,+} \cup \mathcal{H}^*_+(\mathcal{L}_+) \cup r_{2,+}$ and, hence, below $\mathcal{H}_+(\mathcal{L}_+)$. Similaly, if ℓ does not intersect $r_{1,-} \cup \mathcal{H}^*_-(\mathcal{L}_-) \cup r_{2,-}$, then ℓ is above $r_{1,-} \cup \mathcal{H}^*_-(\mathcal{L}_-) \cup r_{2,-}$ and, hence, above $\mathcal{H}_-(\mathcal{L}_-)$. 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 $\mathcal{H}_+(\mathcal{L}_+)$. If ℓ intersects $r_{1,+}$ and is below $q_{1,+}$, then ℓ intersects $\mathcal{H}_-(\mathcal{L}_-)$ since any line that is below $q_{1,+}$ and has a steeper

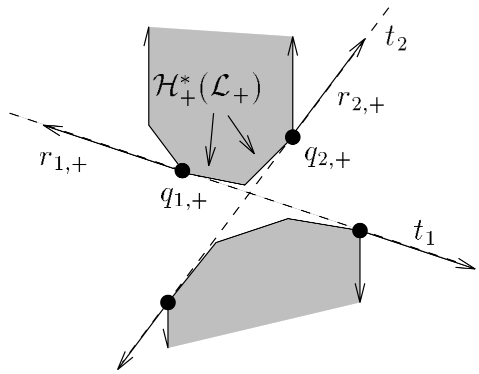

Figure 9: *Defining* $q_{i,+}$, $r_{i,+}$, *and* $\mathcal{H}^*_+(\mathcal{L}_+)$.

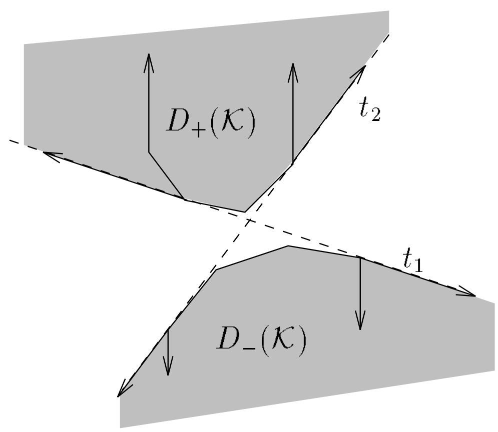

Figure 10: *The dual of* $\mathcal{K}$.

slope than t_1 intersects $\mathcal{H}_-(\mathcal{L}_-)$. Similar arguments hold if ℓ intersects $r_{2,+}$, $r_{1,-}$, or $r_{2,-}$. □

We denote the region above $r_{1,+} \cup \mathcal{H}^*_+(\mathcal{L}_+) \cup r_{2,+}$ by $D_+(\mathcal{K})$ and the region below $r_{1,-} \cup \mathcal{H}^*_-(\mathcal{L}_-) \cup r_{2,-}$ 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 Figure 10).

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 is 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 lower hull of $D(\mathcal{L}_+)$ and the upper hull of $D(\mathcal{L}_-)$ and, hence, the query time is still $O(\log n/(\log p + 1) + 1)$.

6 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 [4]. We give a different proof of their main theorem and 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 [8], we also address the question of how to answer the following four queries for the kernel of a polygon $\mathbf{P}$ in the primal plane given a query structure in the dual plane.

(i) Given a point p, test whether p is in the $kernel(\mathbf{P})$.

(ii) Given a point p, report the tangents to the $kernel(\mathbf{P})$ passing through p.

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

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

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

References

[1] M. J. Atallah and D. Z. Chen. Optimal parallel algorithm for visibility of a simple polygon from a point. In *Proc. 5th ACM Symposium on Computational Geometry*, pages 114–123, 1989.

[2] 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.

[3] 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.

[4] R. Cole and M. Goodrich. Optimal parallel algorithms for polygon and point-set problems. In *Proc. 4th ACM Symposium on Computational Geometry*, pages 201–210, Champaign, Illinois, 1988.

[5] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, Berlin Heidelberg New York, EATCS monographs on theoretical computer science edition, 1987.

[6] 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, July 1979.

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

[8] 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.