Kinetic and Dynamic Data Structures for Convex Hulls and Upper Envelopes[*]

Giora Alexandron[†] Haim Kaplan[‡] Micha Sharir[§]

December 20, 2005

Abstract

Let S be a set of n moving points in the plane. We present a kinetic and dynamic (randomized) data structure for maintaining the convex hull of S. The structure uses $O(n)$ space, and processes an expected number of $O(n^2\beta_{s+2}(n)\log n)$ critical events, each in $O(\log^2 n)$ expected time, including $O(n)$ insertions, deletions, and changes in the flight plans of the points. Here s is the maximum number of times where any specific triple of points can become collinear, $\beta_s(q) = \lambda_s(q)/q$, and $\lambda_s(q)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols. Compared with the previous solution of Basch, Guibas and Hershberger [8], our structure uses simpler certificates, uses roughly the same resources, and is also dynamic.

[*]Work by Haim Kaplan was partially supported by Israel Science Foundation (ISF) grant no. 548-00. Work by Micha Sharir was partially supported by NSF Grant CCR-00-98246, by a grant from the U.S.-Israeli Binational Science Foundation, by a grant from the Israel Science Fund, Israeli Academy of Sciences, for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

[†]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: `gioraa@post.tau.ac.il`

[‡]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: `haimk@post.tau.ac.il`

[§]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA. E-mail: `michas@post.tau.ac.il`

1 Introduction

The *Kinetic Data Structure* (KDS) framework, introduced by Basch, Guibas and Hershberger [8], proposes an algorithmic approach, together with several quality criteria, for maintaining certain geometric configurations determined by a set of objects, each moving along a semi-algebraic trajectory of constant description complexity (see below for a precise definition). Several interesting algorithms have been designed, using this framework, over the past few years, including algorithms for maintaining the convex hull of a set S of (moving) points in the plane [8], the closest pair in such a set [8], a point in the center region of such a set [2], kinetic planar subdivisions [1, 3, 6], kinetic medians and *kd*-trees [4], maintaining the extent of a moving point set [5], kinetic collision detection [7, 16, 19], shooting a moving target [9], kinetic discrete centers [13], kinetic connectivity for unit disks, rectangles, and hypercubes [15, 17], kinetic geometric spanners [18], and kinetic separation of convex polygons [20]; see also [11, 14].

Typically, a geometric algorithm for computing such a configuration determined by a set S is normally designed for the *static* case, where the objects are stationary. When the objects move, the combinatorial representation of the configuration may change at certain *critical times*, when certain "events" occur (e.g., a new vertex of the convex hull may appear, an old vertex may disappear, the closest pair of points changes, etc.). The goal is to design a data structure that can efficiently keep track of these changes, and maintain (a discrete representation of) the correct configuration at all times. Thus the algorithm has to keep track of these critical events, and fix the configuration when they happen. One easy solution is to compute all the possible discrete values of the configuration at all times in advance, given the flight plans of the moving objects. In most cases, though, this would cause a large memory consumption. Furthermore, since objects may change their flight plan at times not known in advance, this pre-calculation may be useless, and fixing the structure after such a change might be too expensive.

The crux in designing an efficient KDS is finding a set of *certificates* that, on one hand, ensure the correctness of the configuration currently being maintained, and, on the other hand, are inexpensive to maintain. When the motion starts, we can compute the closest failure time of any of the certificates, and insert these times into a global event queue. When the time of the next event in the queue matches the current time, we invoke the KDS repair mechanism, which fixes the configuration and the failing certificate(s). In doing so, the mechanism will typically delete from the queue failure times that are no longer relevant, and insert new failure times into it.

To analyze the efficiency of a KDS, we distinguish between two types of events: *internal* and *external*. *External events* are events associated with real (combinatorial) changes in the configuration, thus forcing a change in the output. *Internal events*, on the other hand, are events where some certificate fails, but the overall desired configuration still remains valid. These events arise because of our specific choice of the certificates, and are essentially an overhead incurred by the data structure. If the ratio between the number of internal events to the number of external events is no more than polylogarithmic in the number of input objects, the KDS is said to be *efficient*.[1] Other parameters of the KDS that one would like to minimize are the following.

- The processing time of a critical event by the repair mechanism. If this parameter is no more

[1]Basch et al. [8] considered a KDS to be efficient, if the ratio between the number of internal events to the number of external events is bounded by a small power of the number of input objects. In our definition of efficient KDS, we only allow a degradation factor that is a polylogarithmic function of the number of input objects. We impose similar more stringest restrictions on the other performance parameters of the structure.

than polylogarithmic in the number of input objects, we say that the KDS is *responsive.*

- The maximum number of events at any fixed time in the queue that are associated with one particular object. When this parameter is no more than polylogarithmic in the number of input objects, we say that the KDS is *local.* Locality typically implies that changes in flight plans can be handled efficiently.

- The space used by the data structure. If this is larger than the number of input objects by at most a polylogarithmic factor, we say that the KDS is *compact.*

In addition, which is one of the central issues considered in this paper, one might wish to design a KDS that is also *dynamic*, meaning that it can also efficiently support insertions and deletions of objects.

In their paper, Basch et al. [8] developed a KDS that maintains the convex hull of a set of moving points in the plane, which meets all these criteria, namely, it is compact, efficient, local, and responsive. Specifically, their structure processes $O(n^{2+\varepsilon})$ events, for any $\varepsilon > 0$, each in $O(\log^2 n)$ time. (The number of events has been slightly improved in a later work [5], to $O(n\lambda_s(n))$, where s is the number of times any fixed triple of points can become collinear.) To achieve locality, their algorithm uses a fairly complicated set of certificates. Furthermore, Basch et al. have focused only on kinetization, and did not consider insertions and deletions of points. The motivation for our work is twofold: (i) to simplify the certificates used by [8], and (ii) to obtain a dynamic algorithm that still meets the four quality criteria mentioned above.

Our results. In this paper we present an efficient *dynamic* KDS for maintaining the convex hull of a set of moving points in the plane, which also supports insertions and deletions of points. Our certificates are simpler than those of [8], and the performance of our algorithm is comparable with that of [8].

We assume that each moving point i is given as a pair $(a_i(t), b_i(t))$ of semi-algebraic functions of time of *constant description complexity.* That is, each function is defined as a Boolean combination of a constant number of predicates involving polynomials of constant maximum degree. We present our result in the dual plane, where each point is mapped to the moving non-vertical line $y = a_i(t)x + b_i(t)$, and the goal is to maintain the upper and lower envelopes of this set of moving lines. For simplicity, and without loss of generality, we will only consider the maintenance of the upper envelope.

The main idea in our solution is to maintain the lines sorted by slope in a data structure similar to the stationary data structure of Overmars and van Leeuwen [21]. This is in contrast with the data structure of Basch et al. [8] that does not maintain the lines sorted by slope, but rather keeps them in a tree in some arbitrary order.

Because of some technical difficulties in the analysis, which are discussed at the end of Section 3.3 (these difficulties arise due to lack of tight bounds on the complexity of a single level in planar arrangements), we have to use a *treap* [22] as the underlying tree. Our data structure is therefore randomized, and its performance bounds hold only in expectation.

The data structure of Overmars and van Leeuwen [21] exploits the following simple observation.

Proposition 1.1. *Given two sets of lines L and R, such that any line in L has a smaller slope than that of any line in R, the upper envelope of L and the upper envelope of R have exactly one* *common intersection point q. The envelope is attained by lines of L to the left of q, and by lines of R to the right of q.*

Overmars and van Leeuwen use this to develop a divide-and-conquer algorithm that computes the upper envelope of a set of *stationary* lines in $O(n \log n)$ time, and maintains it, after each insertion or deletion, in $O(\log^2 n)$ time. We follow the same idea, using a *treap* as the underlying tree, in which the lines are stored (in inorder) in their increasing slope order. Let n denote the total number of insertions, and let s denote the number of times any three input points can become collinear. Write $\beta_q(n) = \lambda_q(n)/n$, where q is any constant, and where $\lambda_q(n)$ is the maximum length of a Devenport-Schinzel sequence of order q on n symbols (see [23]). We show that our structure processes an expected number of $O(n^2\beta_{s+2}(n) \log n)$ events,[2] each in $O(\log^2 n)$ expected time, that it has size $O(n)$, and that each line participates in only $O(\log n)$ "certificates" maintained by the structure. In the terminology defined above, our structure is compact, efficient, local, and responsive.

We present the algorithm in three stages. First, we describe the classical dynamic algorithm of Overmars and van Leeuwen for stationary lines [21], upon which our solution is built. Second, we make this algorithm kinetic, by designing a set of simple certificates and an efficient algorithm for maintaining them as the lines move. In this special case, the bound on the number of events slightly improves to $O(n^2\beta_s(n) \log n)$. Third, we make the algorithm dynamic, by showing how to perform insertions and deletions efficiently, adapting and enhancing the basic technique of [21].

2 Preliminaries

In this section we inroduce our framework and notation, by briefly reviewing the data structure of Overmars and van Leeuwen [21] for dynamically maintaining the upper envelope of a set of lines. We describe this structure here in its original stationary context. In the subsequent sections we will make the structure both kinetic and dynamic.

We denote by $S = \{\ell_1, \dots \ell_n\}$ the set of lines in the data structure, sorted in order of increasing slopes, so that ℓ_k is the line with the k-th smallest slope. We store the lines at the leaves of a balanced binary search tree T in this order. Slightly abusing the notation, we also use ℓ_k to denote the node of T containing ℓ_k. Later, we take T to be a *treap* (see [22] and below), but for now any kind of balanced search tree will do. Denote the root of T by r. For a node $v \in T$, denote the left and right children of v by $\ell(v)$ and $r(v)$, respectively, and denote the parent of v by $p(v)$. Denote the set of lines in the leaves of the subtree of v by $S(v)$.

Each node $v \in T$ stores a sorted list of the lines that appear in the upper envelope $E(v)$ of $S(v)$, in their left-to-right order along the envelope, which is the same as the increasing order of their slopes. To facilitate fast implementation of searching, splitting, and concatenation of upper envelopes, we represent each such sorted list as a balanced search tree. Abusing the notation slightly, we denote by $E(v)$ both the upper envelope of the lines in $S(v)$ and the tree representing it.

After sorting the lines of S in the increasing order of their slopes, we build T and the secondary structures $E(v)$, for each $v \in T$, in the following bottom-up recursive manner. For a node v,

[2]In case the number of insertions and deletions, say m, is larger than the maximum number, n, of points in the data structure at any fixed time, the bound on the total number of events is in fact $O(mn\beta_{s+2}(n) \log n)$. One can easily establish this by splitting time into $O(m/n)$ intervals, each containing $O(n)$ updates, use our analysis for each interval, and sum up the bounds in all intervals.

we build $E(v)$ from $E(\ell(v))$ and $E(r(v))$. First we compute the intersection $q(v)$ of $E(\ell(v))$ and $E(r(v))$, by simultaneous binary search over $E(\ell(v))$ and $E(r(v))$, in the manner described in [21]. Then we split $E(\ell(v))$ and $E(r(v))$ at $q(v)$, and concatenate the part of $E(\ell(v))$ that lies to the left of $q(v)$ with the part of $E(r(v))$ that lies to the right of $q(v)$, to obtain $E(v)$.

Using standard search tree machinery, after we split $E(\ell(v))$ and $E(r(v))$ at $q(v)$, the trees representing $E(\ell(v))$ and $E(r(v))$ are destroyed. For that reason, and to save space, Overmars and van Leeuwen [21] store at each node v *only the part of $E(v)$ that does not appear on $E(p(v))$*. One can then reconstruct $E(v)$ on the fly from $E(p(v))$, and from the piece stored at v.

The operations of finding $q(v)$, splitting and concatenating $E(\ell(v))$ and $E(r(v))$, take $O(\log n)$ time each. Therefore, we can build the entire structure in $O(n \log n)$ time. The size of the primary tree T, including the portions of the envelopes $E(v)$ stored at each node v, is $O(n)$. To see this, observe that, for each line ℓ, there is at most one node v, ancestor of ℓ, where ℓ is stored and where it is not adjacent to $q(v)$.

To support insertions and deletions of lines, each time we traverse an edge of the tree from a node v to one of its children, we construct the envelopes $E(\ell(v))$ and $E(r(v))$ from $E(v)$. Later on when we traverse the same edge going from the child back to v we reconstruct $E(v)$ from the potentially new values of $E(\ell(v))$ and $E(r(v))$. The overall cost of an insertion or deletion is $O(\log^2 n)$.

To simplify the presentation in the subsequent sections, we will consider upper envelopes stored at various nodes of the structure as if they are stored there in full, and will ignore the issues related to this more space-efficient representation. Nevertheless, the bounds that we will state will take this improved representation into account.

3 Making the Data Structure Kinetic

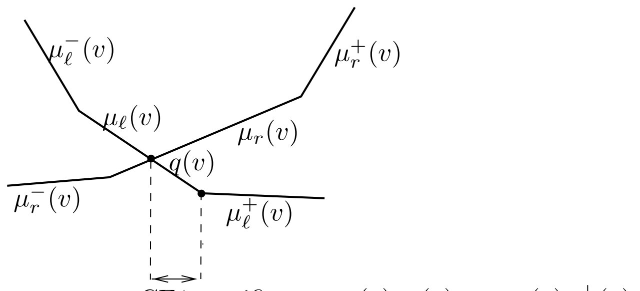
We now show how to maintain the upper envelope E of S, using the structure of Section 2, when the lines are moving along known trajectories, which are assumed to be semi-algebraic functions of time of constant description complexity, and known to the algorithm, except that at certain times the motion ("flight plan") may change (and then the algorithm is notified about the change). Note that now the increasing slope order of the lines $\ell_1, \ldots, \ell_n$ may change over time. So when we refer to ℓ_k we mean the line with the k^{th} smallest slope at some particular time, which will always be clear from the context.

Fix an internal node $v \in T$. We need the following notation. Denote the two lines from $E(\ell(v))$ and $E(r(v))$ that intersect at $q(v)$ as $\mu_\ell(v)$ and $\mu_r(v)$, respectively. Denote the line in $E(\ell(v))$ immediately preceding (resp., succeeding) $\mu_\ell(v)$ as $\mu_\ell^-(v)$ (resp., $\mu_\ell^+(v)$). Similarly, we denote the lines immediately preceding and succeeding $\mu_r(v)$ in $E(r(v))$ by $\mu_r^-(v)$ and $\mu_r^+(v)$, respectively; see Figure 1.

We denote the intersection point of two lines a and b by ab. We write $ab <_x cd$ if the x-coordinate of ab is smaller than the x-coordinate of cd.

To ensure the validity of the structure as the lines are moving, we use two types of certificates, denoted by CT and CE. These are predicates, each involving a small number of lines. As long as all certificates remain true, the validity of the structure is ensured. Each certificate contributes a *critical event* to a global event queue Q, which is the first future time when the certificate becomes invalid (if there is such a time).

(CT) *Certificates that ensure the validity of T.*



CE1 certificate: $\mu_\ell(v)\mu_r(v) <_x \mu_\ell(v)\mu_\ell^+(v)$

Figure 1: One of the four CE-certificates for guaranteeing the validity of $E(v)$.

For each pair of consecutive lines ℓ_k, ℓ_{k+1} in T, we have a CT-certificate that asserts that the slope of ℓ_k is smaller than or equal to the slope of ℓ_{k+1}.

(CE) *Certificates that ensure the validity of the envelopes $E(v)$.*
For each node v, we maintain the following (at most) four certificates (see Figure 1); recall that $\mu_\ell(v)\mu_r(v) = q(v)$.

1. $\mu_\ell(v)\mu_r(v) <_x \mu_r(v)\mu_r^+(v)$,
2. $\mu_\ell(v)\mu_r(v) <_x \mu_\ell(v)\mu_\ell^+(v)$,
3. $\mu_\ell(v)\mu_r(v) >_x \mu_r(v)\mu_r^-(v)$,
4. $\mu_\ell(v)\mu_r(v) >_x \mu_\ell(v)\mu_\ell^-(v)$.

Alternatively, we can use certificates which guarantee that 1) $\mu_r(v)\mu_r^+(v)$ is above the line $\mu_\ell(v)$, 2) $\mu_\ell(v)\mu_\ell^+(v)$ is below the line $\mu_r(v)$, 3) $\mu_r(v)\mu_r^-(v)$ is below the line $\mu_\ell(v)$ and , 4) $\mu_\ell(v)\mu_\ell^-(v)$ is above the line $\mu_r(v)$.

The proof of the following lemma is straightforward.

Lemma 3.1. *As long as all CT and CE certificates are valid, the lines are stored at the leaves of T from left to right in increasing order of their slopes, and, for each node $v \in T$, $E(v)$ stores the correct upper envelope of $S(v)$.* □

3.1 Handling critical events

By a CT or CE *critical event* we mean a failure event of one of the current CT or CE certificates. A CT certificate fails when the slopes of two consecutive lines ℓ_k and ℓ_{k+1} in T become equal. If, right after the failure, the slope of ℓ_{k+1} becomes smaller than the slope of ℓ_k, we have to update T as follows. Let $w = LCA(\ell_k, \ell_{k+1})$ be the lowest common ancestor of the two leaves containing ℓ_k and ℓ_{k+1} (see Figure 2).

We swap ℓ_k and ℓ_{k+1}, and then delete from Q the two CT events associated with ℓ_k and ℓ_{k-1}, and with ℓ_{k+1} and ℓ_{k+2}, and add to Q up to three new CT events: between ℓ_{k-1} and the new ℓ_k, between the new ℓ_{k+1} and ℓ_{k+2}, and between ℓ_k and ℓ_{k+1}, if their slopes become equal again at some future time. In addition, this swap might affect upper envelopes at the nodes on the two paths

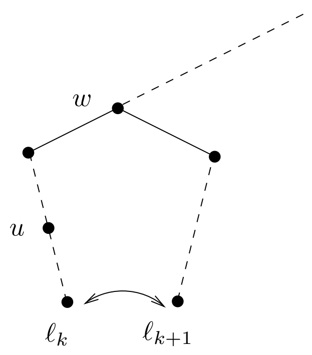


Figure 2: Handling a CT event, at which the order of slopes of lines in two consecutive leaves changes. We swap the lines between the leaves, and recompute the envelopes on the paths to the lowest common ancestor.

from ℓ_k and from ℓ_{k+1} to the root.[3] Hence, for each node u on either of the paths, we recompute $E(u)$ *from scratch* in a bottom-up fashion. In particular, it means that, for each such node u at which $E(u)$ has changed, we may have to delete from Q the at most four CE events associated with u, and replace them by at most four new CE events.

When a CE certificate fails at some node v, $E(v)$ is no longer valid. The following changes can take place: If the certificate $\mu_\ell(v)\mu_r(v) <_x \mu_r(v)\mu_r^+(v)$ fails, the line $\mu_r(v)$ is removed from $E(v)$. If $\mu_\ell(v)\mu_r(v) <_x \mu_\ell(v)\mu_\ell^+(v)$ fails, the line $\mu_\ell^+(v)$ is added to $E(v)$ between $\mu_\ell(v)$ and $\mu_r(v)$. Similarly, if $\mu_\ell(v)\mu_r(v) >_x \mu_r(v)\mu_r^-(v)$ fails, the line $\mu_r^-(v)$ is added to $E(v)$ between $\mu_\ell(v)$ and $\mu_r(v)$, and if $\mu_\ell(v)\mu_r(v) >_x \mu_\ell(v)\mu_\ell^-(v)$ fails, the line $\mu_\ell(v)$ is removed from $E(v)$. Because of the continuity of the motion of the lines, only these local changes can occur at a failure of a CE certificate.

We restore $E(v)$ by inserting or deleting the appropriate line at the appropriate location. We replace the four old CE certificates associated with v by four new certificates, to reflect the fact that either $\mu_r(v)$ or $\mu_\ell(v)$ has changed, as did its predecessor and successor in the respective sub-envelope. We also delete from Q the failure times of the old certificates, and insert into Q the failure times of the new certificates.

The change in $E(v)$ may also cause $E(w)$ to change at ancestors w of v. We propagate the change from v up towards the root, until we reach an ancestor w of v for which $E(w)$ is not affected by the change at v. Let w be an ancestor of v at which $E(w)$ changes. Let $p(w)$ be the parent of w, and let $s(w)$ be the sibling of w. If the line that joins or leaves $E(w)$ also joins or leaves $E(p(w))$, we change $E(p(w))$ accordingly. In addition, if the change replaces the line in $E(w)$ on which the intersection of $E(w)$ and $E(s(w))$ occurs, or one of lines adjacent to it on $E(w)$, we also replace the CE certificates associated with $p(w)$, and replace the corresponding failure times in Q.

[3]In fact, if the swap is between ℓ_k and ℓ_{k+1}, where $k \neq 1$ and $k+1 \neq n$, then no change in the upper envelope can happen for nodes z on the path from w to the root. This is because, right before the event, only one of the lines ℓ_k or ℓ_{k+1} can be on $E(z)$, as is easily checked. On the other hand if the swap is between ℓ_1 and ℓ_2 or between ℓ_{n-1} and ℓ_n, it may also affect every ancestor z of w since both lines may occur on the envelope of z right before the swap.

3.2 Performance analysis

Using the terminology of [8] (see also the introduction), we show that the resulting data structure is *compact, local, responsive, and efficient.*

Compactness. We want to show that the size of the data structure is small. Clearly, we have a linear number of CT certificates, and a linear number of CE certificates, so our event queue Q is of linear size. The size of the primary tree T and of all the trees $E(v)$ is $O(n)$, if we store partial envelopes in the manner outlined in Section 2. Therefore our KDS can be implemented in linear space, and is thus *compact*.

Locality. We want to show that each line ℓ is involved in a small number of certificates, and that any change in the flight plan of ℓ can be quickly encoded into the data structure. Each line ℓ participates in only $O(\log n)$ certificates. Indeed, it participates in at most two CT certificates, one with the line with the next larger slope, and one with the line with the next smaller slope (if such lines exist). In addition, ℓ may participate in at most four CE certificates in each of its $O(\log n)$ ancestors in T. It follows that the data structure is *local*, and that one can update the flight plan of a line ℓ in $O(\log^2 n)$ time. (Usually, such changes in flight plans assume a non-dynamic scenario, in which any such change in the motion of any line ℓ keeps its motion continuous. However, we will later show that our structure can also be made dynamic, which also allows us to implement "abrupt" changes in the flight plans by simply deleting the respective line, and re-inserting it with the new flight plan. See Section 4 for details.)

Responsiveness. We want to show that when we reach a critical event, we can quickly update the certificates maintained by the structure, so as to restore and maintain its validity. Specifically, we show that the time needed to process a critical event is $O(\log^2 n)$, which thus makes the structure *responsive*. When a CT certificate fails, we recompute the upper envelope $E(v)$ at $O(\log n)$ nodes v, along the two paths from the leaves storing the two respective lines to the root. At each such node v, recomputing $E(v)$ takes $O(\log n)$ time, so we spend $O(\log^2 n)$ time in recomputing all these envelopes. In addition, for each node v in which we recompute $E(v)$, the four CE certificates associated with v may change. For each such node v, we have to delete from Q the failure times of the old certificates, and insert into Q the failure times of the new ones. Since $O(\log n)$ such certificates may change, updating them and the queue Q takes $O(\log^2 n)$ time.

When a CE certificate fails, we may have to fix the envelope at $O(\log n)$ nodes, along the path from the node where the certificate fails to the root. At each such node v, we delete one line or insert one line into $E(v)$, so these updates take $O(\log^2 n)$ time. This in turn may cause, as above, $O(\log n)$ other CE certificates to change, which we handle as above, in a total of $O(\log^2 n)$ time.

The most interesting and involved part of the analysis is to show that the data structure is *efficient*, in the sense of obtaining an upper bound on the total number of critical events that is comparable with the bound on the total number of real combinatorial changes in the overall upper envelope.

3.3 Bounding the number of critical events

To analyze the total number of critical events that our data structure processes, we refine a technique of Basch et al. [8], in which time is considered as an additional (static) dimension, which allows us to represent each critical event as a vertex of an appropriate upper envelope of *bivariate* functions, where these envelopes are the graphs of the sub-envelopes $E(v)$, as they evolve over time.

In more detail, we parametrize the moving lines as surfaces in the 3-dimensional xty-space. For each line $\ell \in S$, its surface σ_ℓ is the locus of all points (x, t, y), such that (x, y) lies on ℓ at time t. Note that σ_ℓ is a ruled surface, and that it is *xt-monotone*, so that we can regard it as the graph of a function of x and t, which, with a slight abuse of notation, we denote as $y = \sigma_\ell(x, t)$. For any node v of T, we denote by $E_3(v)$ the upper envelope of the bivariate functions σ_ℓ, for $\ell \in S(v)$.

If we assume that the motions of the lines are semi-algebraic of constant description complexity, then the surfaces σ_ℓ are also semi-algebraic of constant description complexity. The intersection curve of a pair of surfaces is the trace of the moving intersection point between the two respective lines, and an intersection point of three surfaces represents an event where the three respective lines become concurrent. It follows that the number of changes in the time-evolving upper envelope of the lines is upper bounded by the combinatorial complexity of the upper envelope of their surfaces. Note that the above assumptions on the motion of the lines, including the assumption of general position, imply that any triple of surfaces intersect in at most s points, where s is some constant.

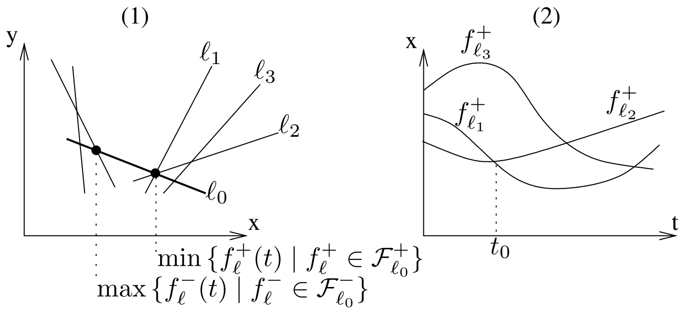
The following argument shows that the complexity of the upper envelope of n such surfaces is $O(n^2\beta_{s+2}(n))$, where $\beta_q(n) = \lambda_q(n)/n$, and $\lambda_q(n)$ is the maximum length of a Davenport-Schinzel sequence of order q on n symbols [23]. Fix a line ℓ_0. For each line $\ell \neq \ell_0$ with a larger (resp., smaller) slope, let $f_\ell^+(t)$ (resp., $f_\ell^-(t)$) denote the x-coordinate of the intersection point $\ell_0 \cap \ell$ at time t. Let $\mathcal{F}_{\ell_0}^+$ (resp., $\mathcal{F}_{\ell_0}^-$) denote the set of all functions $f_\ell^+(t)$ (resp., $f_\ell^-(t)$). Note that in general these functions are partially defined: At times t' where the slopes of ℓ and ℓ_0 become equal, one function, say f_ℓ^+, ceases to be defined, and the other function f_ℓ^- starts being defined. If this happens several times, the functions have disconnected domains of definition, and then we regard each such function as multiple partial functions, each with a connected domain of definition. By our assumptions on the motion, two lines have identical slopes at most a constant number of times, so the number of functions in $\mathcal{F}_{\ell_0}^+$ (resp., $\mathcal{F}_{\ell_0}^-$) is $O(n)$. Furthermore, since three lines become concurrent at most s times, each pair of functions in $\mathcal{F}_{\ell_0}^+$ (resp., $\mathcal{F}_{\ell_0}^-$) intersect in at most s points. It follows that, at any time t, the portion of ℓ_0 that appears on the upper envelope (if there exists such a portion) is a connected interval, delimited on the right by $\min\{f_\ell^+(t) \mid f_\ell^+ \in \mathcal{F}_{\ell_0}^+\}$, and on the left by $\max\{f_\ell^-(t) \mid f_\ell^- \in \mathcal{F}_{\ell_0}^-\}$; see Figure 3.

The complexity of each of these lower and upper envelopes is at most $O(\lambda_{s+2}(n)) = O(n\beta_{s+2}(n))$ [23], which thus also bounds the number of times these envelopes "run into" each other (see [23]), causing ℓ_0 to disappear or re-appear on the envelope. Repeating this analysis to each line ℓ_0 yields the asserted overall bound.

To sum up, we conclude that the total number of changes in the upper envelope of our set of moving lines (the so-called external events), over time, is $O(n^2\beta_{s+2}(n))$.

Remark. The complexity of the upper envelope of the set of ruled surfaces defined by the lines is in fact bounded by the complexity of the lower envelopes $\min\{f_\ell^+(t) \mid f_\ell^+ \in \mathcal{F}_{\ell_0}^+\}$, over all lines ℓ_0. (The corresponding upper envelopes $\max\{f_\ell^-(t) \mid f_\ell^- \in \mathcal{F}_{\ell_0}^-\}$ are not really needed for the preceding analysis, but we define them since they will be used in the proof of Lemma 3.2 below.) To see this, consider a vertex p on the upper envelope of the surfaces. This vertex appears as a vertex of the lower envelope $\min\{f_\ell^+(t) \mid f_\ell^+ \in \mathcal{F}_{\ell'}^+\}$, where ℓ' is the line with the smallest slope (at the time of concurrency) among the three lines defining p. □

We next derive an upper bound on the number of events that our data structure handles (the so-called internal events), which is not much larger than the bound just derived. By our assumption



(1)
(2)
y
x
ℓ_1
ℓ_3
ℓ_2
ℓ_0
$\min\{f_\ell^+(t) \mid f_\ell^+ \in \mathcal{F}_{\ell_0}^+\}$
$\max\{f_\ell^-(t) \mid f_\ell^- \in \mathcal{F}_{\ell_0}^-\}$
x
t
$f_{\ell_3}^+$
$f_{\ell_1}^+$
$f_{\ell_2}^+$
t_0

Figure 3: (1) The lines intersecting ℓ_0 at time t_0. At t_0, $f_{\ell_1}^+(t)$, $f_{\ell_2}^+(t)$, and $f_{\ell_3}^+(t)$ are defined, and both $f_{\ell_1}^+(t)$, and $f_{\ell_2}^+(t)$ have the minimum x-coordinate. (2) The arrangement of the functions $\mathcal{F}_{\ell_0}^+$ in the tx-plane. The complexity of the lower envelope of these functions, and the complexity of the upper envelope of $\mathcal{F}_{\ell_0}^-$, asymptotically bound the total number of CE events that involve the line ℓ_0.

on the motion, the slopes of two lines can coincide at most $O(1)$ times. Therefore, the total number of CT events is $O(n^2)$.

The main part of the analysis is to bound the number of CE events. Consider such an event that occurs when a CE certificate at some node v of T fails. Note that, at this event, three lines of $S(v)$ become concurrent, and the point of concurrency lies on the upper envelope $E(v)$. Hence, we can charge the event to a vertex of the corresponding bivariate upper envelope $E_3(v)$.

(Note that we can apply this charging only to events that are extracted from the front of Q and are processed as events that update the structure. Events that are inserted into Q and are removed later during another update of the structure do not represent real envelope vertices. Nevertheless, the number of such events that can be generated during an update is only $O(\log n)$, so the overall number of these spurious events is at most $O(\log n)$ times the overall number of real CE events, which we now proceed to bound.)

Not every vertex of $E_3(v)$ corresponds to a CE event at v. Each charged vertex is an intersection of three surfaces, such that at least one of them corresponds to a line in $S(\ell(v))$, and at least one of them corresponds to a line in $S(r(v))$. Recall that the intersection corresponds to the event where the three lines defining these surfaces become concurrent, at a point on the upper envelope $E(v)$. To bound the number of CE events at v, we need to bound the number of such "bichromatic" vertices of $E_3(v)$.

Let $P(v)$ denote the multiset of pairs of lines (ℓ, ℓ'), for which there exists some time at which $\ell \in S(\ell(v))$ and $\ell' \in S(r(v))$ simultaneously. The multiplicity of a pair (ℓ, ℓ') in $P(v)$ is taken to be the number of times t, such that, just before time t, either ℓ was not in $S(\ell(v))$ or ℓ' was not in $S(r(v))$, and, just after time t, $\ell \in S(\ell(v))$ and $\ell' \in S(r(v))$. In other words, the multiplicity of (ℓ, ℓ') is the number of maximal connected time intervals during which $(\ell, \ell') \in S(\ell(v)) \times S(r(v))$. The following main technical lemma bounds the total number of events encountered in v, in terms of $|P(v)|$.

Lemma 3.2. *Let $P(v)$ be the multiset of pairs of lines $(\ell, \ell') \in S(\ell(v)) \times S(r(v))$, as defined above. Let m be the maximum number of lines under v at any fixed time. Let s be the maximum number of times a triple of lines become concurrent. Then the total number of CE events that are encountered* *at v is $O(|P(v)|\beta_{s+2}(m))$.*

Proof. We apply a similar argument to the one used above. Fix a line ℓ_0, and for each line $\ell \neq \ell_0$, let $f^+_{\ell_0,\ell}(t)$ (resp., $f^-_{\ell_0,\ell}(t)$) be defined if the slope of ℓ is greater (resp., smaller) than that of ℓ_0, in which case it is equal to the x-coordinate of the intersection point $\ell_0 \cap \ell$ at time t. (Thus, for any time t where ℓ_0 and ℓ are not parallel, exactly one of these functions is defined.)

Fix a node v of the tree, and let ℓ_0 be a line in $S(\ell(v))$. Define $\mathcal{F}^+(\ell_0, v)$ to be the set of all functions $f^+_{\ell_0,\ell}$, for lines ℓ in $S(r(v))$, where the domain of definition of any $f^+_{\ell_0,\ell}$ is further restricted to those times at which ℓ_0 is stored at $S(\ell(v))$ and ℓ at $S(r(v))$. We define the family $\mathcal{F}^-(\ell_0, v)$ in complete analogy, for lines ℓ_0 stored at $S(r(v))$. As before, a function in either collection with a disconnected domain of definition is represented as several "sub-functions", each with a connected domain of definition.

Consider now an event encountered at v, where three lines ℓ_0, ℓ_1, ℓ_2 become concurrent on $E(v)$ at time t_0, with at least one line belonging to $S(\ell(v))$ and at least one belonging to $S(r(v))$. Suppose first that one of the lines, say ℓ_0, is stored at $S(\ell(v))$, and that the other two, ℓ_1, ℓ_2, are stored at $S(r(v))$. Then $f^+_{\ell_0,\ell_1}(t) = f^+_{\ell_0,\ell_2}(t)$, and this intersection lies on the *lower envelope* of the set $\mathcal{F}^+(\ell_0, v)$ (refer to Figure 3). A symmetric property holds when ℓ_0 is stored at $S(r(v))$ and ℓ_1, ℓ_2 are stored at $S(\ell(v))$, in which case we get a vertex of the *upper envelope* of the set $\mathcal{F}^-(\ell_0, v)$.

Fix a line ℓ_0, and consider an interval I of time where ℓ_0 is stored at $S(\ell(v))$. Let $N(\ell_0, v)$ denote the number of lines that are stored at $S(r(v))$ at some time in I. If a line leaves and re-enters the right subtree multiple times during I, we count each of its appearances as a different line in $N(\ell_0, v)$ (this is in accordance with the definition of $P(v)$). The complexity of the lower envelope of $\mathcal{F}^+(\ell_0, v)$ during I is at most $\lambda_{s+2}(N(\ell_0, v)) \leq N(\ell_0, v)\beta_{s+2}(N(\ell_0, v))$, which can be slightly improved to $O(N(\ell_0, v)\beta_{s+2}(m))$ [23], where s is, as above, the maximum number of times a triple of lines become concurrent, and where m is the maximum number of lines under v at any fixed time. The sum of these bounds, over all lines ℓ_0 and all intervals I, is at most $O(|P(v)|\beta_{s+2}(m))$. Applying a symmetric argument for lines ℓ_0 that are stored at $S(r(v))$ completes the proof of the lemma. ☐

A slightly improved bound. Agarwal et al. [5] proved a slightly tighter bound of $O(n^2\beta_s(n))$ on the number of changes in the convex hull of n moving points. Using their technique we can also establish this tighter bound on the complexity of the upper envelope of a set of n ruled surfaces defined by n moving lines as above, but only when the motion of the lines is restricted so that no line ever becomes parallel to the y-axis. (In case the moving lines are duals of moving points, their motion does indeed obey this restriction.) The same technique also allows us to improve the bound in Lemma 3.2 to $O(|P(v)|\beta_s(m))$, but it does not extend to the dynamic setting of Section 4.

The idea is to bound the number of points of the lower envelope $\min\{f^+_\ell(t) \mid f^+_\ell \in \mathcal{F}^+_{\ell_0}\}$ that correspond to points of the upper envelope of the ruled surfaces by the length of a particular Davenport-Schinzel sequence Ψ of order s on $|\mathcal{F}^+_{\ell_0}|$ symbols (a length bounded by $\lambda_s(|\mathcal{F}^+_{\ell_0}|) \leq \lambda_s(n)$).

Consider the sequence Ψ of lines whose functions attain the envelope $\min\{f^+_\ell(t) \mid f^+_\ell \in \mathcal{F}^+_{\ell_0}\}$, ordered by increasing time of their appearances on the envelope. Each element a in Ψ corresponds to a function $f^+_a \in \mathcal{F}^+_{\ell_0}$ and a maximal time interval $[t(a), t'(a)]$ such that $f^+_a(t) = \min\{f^+_\ell(t) \mid f^+_\ell \in \mathcal{F}^+_{\ell_0}\}$ for all $t \in [t(a), t'(a)]$. We remove from Ψ every occurrence of a line a such that the intersection point of a and ℓ_0 never appears on the upper envelope of the lines during the time interval $[t(a), t'(a)]$. (So for every occurrence of a line a that remains in Ψ there is a time $t \in [t(a), t'(a)]$ in which the intersection of a with ℓ_0 was on the upper envelopes of the lines.) We then also remove duplicates from Ψ (that is, any occurrence of a symbol a that immediately follows another occurrence of a). Clearly, the length of Ψ (after these trasformations) bounds the number of vertices of the lower envelope $\min\{f_\ell^+(t) \mid f_\ell^+ \in \mathcal{F}_{\ell_0}^+\}$ that correspond to vertices of the upper envelope of the ruled surfaces.

To see that Ψ is a Davenport-Schinzel sequence of order s, consider an occurrence of the symbol a followed by an occurrence of the symbol b. The definition of Ψ implies that at some time t_1 the intersection of a with ℓ_0 appeared on the upper envelope of the lines, and therefore was above the line b, and, analogously, at some later time t_2 the intersection point of b and ℓ_0 was above the line a. See Figure 4.

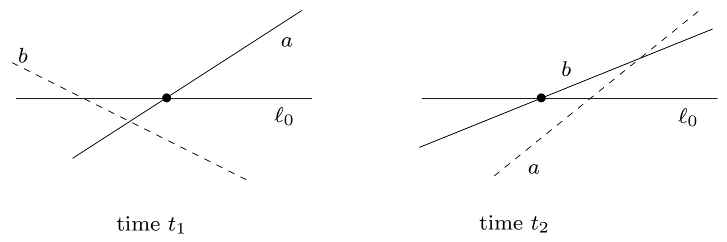


Figure 4: An alternation of a and b in Ψ.

Since lines never become parallel to the y-axis, we can interpret the lines as continuously moving points in the dual plane. With an appropriate duality transform, it follows that in the dual plane we obtain that the triangle $\triangle\ell_0 ab$ is oriented clockwise at time t_1, and counterclockwise at time t_2. Therefore, between times t_1 and t_2, the points a, b, and ℓ_0 must become collinear. Going back to the primal plane, we obtain that between times t_1 and t_2 the lines a, b and ℓ_0 are concurrent. Since a fixed triple of lines cannot become concurrent more than s times, we obtain that Ψ is a Davenport-Schinzel sequence of order s.

Note that the argument breaks down in the dynamic case, when the lines can appear or disappear, in which case all we can show is that Ψ is a Davenport-Schinzel sequence of order $s + 2$. $\square$

A technical difficulty. The next goal is to bound the quantities $|P(v)|$. Since the lines keep swapping between the nodes of T, the sets $P(v)$ keep acquiring new pairs. The difficulty in the analysis stems from the fact that if a line ℓ enters, say, the left subtree $\ell(v)$ of a node v, it creates $|S(r(v))|$ new pairs with the lines stored at $r(v)$, all of which are to be added to $P(v)$. That is, we pay for each swap a price that is rather small if v is low in the tree, but which may become quite expensive when v is close to the root. More precisely, the sum $\sum_v |P(v)|$, over all nodes v of T, is $O\left(\sum_v |S(v)| \cdot (|S(v)| + M(v))\right)$, where $M(v)$ is the number of swaps performed between the left and right subtrees of v.

To appreciate the difficulty in bounding this sum, consider the slopes of the lines as functions of time. These n functions define an arrangement $\mathcal{A}$ in the slope-versus-time plane. Each swap of lines between the k-th and the $(k+1)$-st leaves of T corresponds to a vertex of $\mathcal{A}$ where the k-th and the $(k+1)$-st levels of $\mathcal{A}$ meet. Now even in the simplest case, where the slopes are linear functions of time, the best upper bound known for the complexity of the k-th level is $O(nk^{1/3})$ [12], and the situation becomes much worse for classes of more general curves (see, e.g., [10]).

Consider, for example, the root r of T, and set $k = n/2$. Assume that our arrangement of slope functions is such that the complexity of its $(n/2)$-level is large, say $\Theta(n^{4/3})$. Then each vertex v

at level $n/2$ of the arrangement of the slope functions corresponds to a swap between the left and right subtrees of r, and thus adds $n/2$ new pairs to the multiset $P(r)$. In total, we would have $|P(r)| = \Theta(n^{7/3})$. Plugging this bound into Lemma 3.2 already yields a bound on the number of internal events that is much larger than the near-quadratic bound on the number of external events.

On the other hand, the *average* number of vertices in a level of the arrangement of the slope functions is only $O(n)$. Thus, if we could substitute $M(v) = O(n)$ at each node v, we would get $\sum_v |P(v)| = O(n) \cdot \sum_v |S(v)| = O(n^2 \log n)$.

Before proceeding, we remark that the best known *lower bounds* for the complexity of a single level (in any arrangement of well-behaving curves) are very close to linear [24], and the prevailing conjecture is that the upper bounds are also near-linear. In this case, the calculation just given, appropriately modified, yields a near-quadratic bound on the number of internal events, in which case the refined analysis, given in the rest of the paper, is not needed.

3.4 Treaps

The preceding discussion means that, with a lack of good bounds on the complexity of any single level in an arrangement $\mathcal{A}$ of functions of low complexity in the plane (namely, our slope-versus-time functions), our approach falls short of proving a good bound on the number of internal events, if the underlying tree T causes levels of $\mathcal{A}$ with large complexity to appear near the root. To overcome this difficulty, and exploit the fact that, on average, levels have linear size, we make T a *treap* [22]. Intuitively, using a treap allows us to make the height of a “bad level” in T a random variable, so that, on average (over the choice of the priorities that define the treap), swaps at that level would occur rather low in the tree, and consequently would not be too expensive.

In more details, a *treap* is a randomized search tree with optimal *expected* behavior. Each node v in the treap has two fields $rank(v)$ and $priority(v)$. The treap is a search tree with respect to the *ranks*, and a heap with respect to the *priorities*. We use integer ranks from 1 to n, that index the given lines in the increasing order of their slopes. We assume that the priorities are drawn independently and uniformly at random from an appropriate continuous distribution, so that, with probability 1, the set of priorities defines a random permutation of the nodes.[4] Note that, once we draw the priorities, the resulting treap T is uniquely determined.

We turn our underlying tree T into a treap as follows. A node v of rank k stores the line $\mu(v) = \ell_k$, which is the line with the k-th smallest slope. We now denote by $S(v)$ the set of lines stored at all nodes in the subtree rooted at v, including $\mu(v)$ itself, and we define $E(v)$ to be the upper envelope of the new set of lines $S(v)$.

Since now every node of T stores a line, rather than just the leaves, we need to slightly modify the algorithm. To understand the data structure stored at a node v and the associated certificates, think of each node v as split into two nodes, v_{low} and v_{high}. The left child of v_{low} is $\ell(v)_{\text{high}}$ and the right child of v_{low} is a leaf containing $\mu(v)$. The left child of v_{high} is v_{low} and the right child of v_{high} is $r(v)_{\text{high}}$. See Figure 5. In the transformed tree, lines are only stored at the leaves. Now we compute $E(v_{\text{low}})$, and $E(v_{\text{high}})$, and the CE certificates associated with them, in the same manner as in the preceding algorithm, where the lines were stored only at the leaves. We store at v the portion of $E(v_{\text{low}})$ that does not appear in $E(v_{\text{high}})$, and the portion of $E(v_{\text{high}})$ that does not appear in $E(p(v_{\text{high}})_{\text{low}})$. We associate with v the CE certificates associated with v_{low} and the CE

[4] In practice, integers drawn at random from a sufficiently large range are good enough. See [22].

certificates associated with v_{high}.[5] Note that $E(v)$ is equal to $E(v_{high})$. In what follows we regard this node splitting as implicit in the description of the algorithm, which is formulated in terms of the original unsplit nodes of the treap.

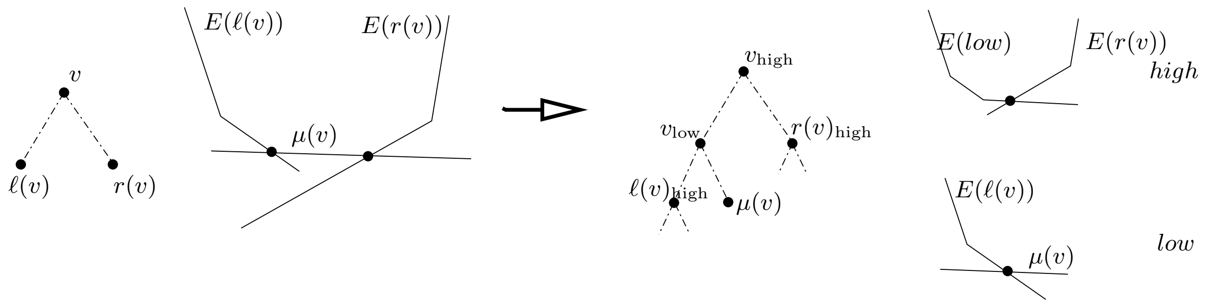


Figure 5: Splitting v into two nodes, v_ℓ and v_h.

3.4.1 Handling critical events

The main modification of the preceding analysis for the case of treaps is in handling CT events. Consider a CT event, involving a swap between two lines ℓ_k and ℓ_{k+1} whose slopes are equal at the critical time t. Let v be the node containing ℓ_k, and v' the node containing ℓ_{k+1}; then $rank(v) = k$ and $rank(v') = k + 1$. It follows that either v' is the leftmost descendant of $r(v)$, or v is the rightmost descendant of $\ell(v')$. When processing the swap, we place ℓ_{k+1} in v and ℓ_k in v', without changing the structure of the treap. Then we recompute the envelopes $E(w)$, for all nodes w on the path between v and v', and update the CE events associated with each such node w. Finally, we delete from Q the CT events previously associated with ℓ_k and ℓ_{k+1}, and insert into Q new CT events between ℓ_k and ℓ_{k+2}, between ℓ_{k+1} and ℓ_{k-1}, and between ℓ_k and ℓ_{k+1} (if their slopes become equal again at some future time).

Handling CE events is done in essentially the same way as in Section 3, and we omit the easy details.

3.4.2 Performance analysis for treaps

The same argument as in Section 3.2 shows that our data structure can be implemented in linear space. The analysis of [22] shows that the depth of any node in a treap is on average (over the draw of the priorities) $O(\log n)$. This fact immediately implies that any line participates in an expected number of $O(\log n)$ certificates at any given time, and that, in any CT or CE critical event, the expected number of nodes v that need to be updated is $O(\log n)$, and thus the expected time it takes to process a critical event, or a change in the flight plan of a line, is $O(\log^2 n)$. Hence, the new data structure is compact, local, and responsive, in an expected sense.

Number of critical events in the case of treaps. We bound the expected number of critical events using the approach suggested in Section 3.3. The following version of Lemma 3.2 holds when

[5]Note that v_{low} has only two certificates associated with it, since the envelope of $r(v_{\text{low}})$ is a single line. Hence the maximum number of certificates associated with each original node in the treap is six.

a line is stored at every node of T. The proof follows that of Lemma 3.2, and is omitted.

Lemma 3.3. *Let $P(v)$ be the multiset of pairs of lines (ℓ, ℓ'), such that (i) $\ell \neq \ell'$, (ii) $\ell \in S(\ell(v))$ or $\ell = \mu(v)$, and (iii) $\ell' \in S(r(v))$ or $\ell' = \mu(v)$, where the multiplicity of a pair is the number of maximal connected time intervals during which (ℓ, ℓ') satisfy (i)–(iii). Let m be the maximum number of lines under v at any fixed time (including also $\mu(v)$). Let s be the maximum number of times where any fixed triple of lines becomes concurrent. Then the total number of CE events that are encountered at v is $O(|P(v)|\beta_{s+2}(m))$.*

This lemma reduces the problem of bounding the expected number of events to the problem of bounding the expected value of the sum $\mathcal{P} = \sum_{v\in T} |P(v)|$ of the sizes of the multisets $P(v)$, over all nodes v. Recall that the sets $P(v)$ are affected only by the initial sets $S(v)$ at the begining of the motion and by the swaps that take place at CT critical events.

We perform the analysis in two steps. First, we bound the expected initial value of $\mathcal{P}$. Then we bound the expected contribution of each swap to $\mathcal{P}$. We denote by π the permutation of the nodes when we order them by increasing priority. Specifically, $\pi(v)$ is the number of nodes with priorities smaller than the priority of v (if the priorities are drawn from any appropriate continuous distribution, the probability of a tie is 0, and we will ignore this possibility). In the following we refer to a line ℓ_k simply by its index k, that is, by its rank in the list of lines sorted by slope. We also denote by $v(k)$ the node containing line k, which is the node of rank k of the treap. Note that $v(k)$ is always the same node but the line that it contains may change over time through swaps.

Bounding the initial value of $\mathcal{P}$ is trivial. Indeed, a pair (i, j), with $i < j$, appears in exactly one set $P(v)$: If i is a descendant of j then (i, j) belongs (only) to $P(v(j))$. Symmetrically, if j is a descendant of i then (i, j) belongs (only) to $P(v(i))$. Finally, if neither of them is a descendant of the other, then (i, j) belongs only to $P(v)$, where v is the lowest common ancestor of i and j. Hence, initially, we have $\sum_v |P(v)| = \binom{n}{2}$.

We now estimate the expected contribution of a swap between two consecutive lines, say line ℓ of rank $m-1$ before the swap, and line ℓ' of rank m before the swap. After the swap line ℓ has rank m and line ℓ' has rank $m-1$. Node $v(m-1)$, the node of rank $m-1$ in the heap, contains ℓ before the swap and ℓ' after the swap. Similarly, node $v(m)$ contains line ℓ' before the swap and line ℓ after the swap. Clearly, either $v(m-1)$ is the rightmost leaf descendant of $\ell(v(m))$, or $v(m)$ is the leftmost leaf descendant of $r(v(m-1))$. The two cases are symmetric, so we only handle the first case. See Figure 6.

As a result of the swap between ℓ and ℓ', line ℓ' creates a new pair with every line j in the subtree of $v(m)$. These pairs are added to $P(v(k))$, where $v(k)$ is the lowest common ancestor of $v(j)$ and $v(m-1)$. Similarly, for every j in the subtree of $\ell(v(m))$, other than $m-1$, we get a new pair of ℓ and j that contributes to $P(v(m))$. Therefore we will estimate the expected number of new pairs created by ℓ in $v(m)$ after the swap; the expected number of new pairs created by ℓ' is the same. Moreover, as is easily verified, no new pairs are formed with elements $j > m$, nor with elements j to the left of the subtree of $\ell(v(m))$.

For $j < m-1 < m$, define $A_{j,m}$ to be an indicator random variable, that is 1 if and only if $v(j)$ is a descendant of $v(m)$. As just argued, if j and ℓ form a new pair in $P(v(m))$ (again, recall that line ℓ resides in $v(m)$ after the swap), then j must be a descendant of that node, and vice versa. Hence, the expected number of new pairs created by ℓ is

$$\sum_{j|j<m-1} \mathbf{E}(A_{j,m}).$$

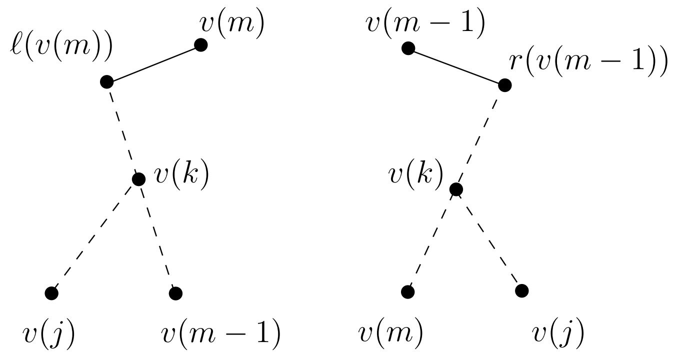


Figure 6: Line ℓ' residing in $v(m)$ swaps with line ℓ in $v(m-1)$.

To compute $\mathbf{E}(A_{j,m})$, we have to calculate the probability that $v(j)$ is a descendant of $v(m)$. This happens if and only if $\pi(y) < \pi(m)$, for all nodes y such that $j \le rank(y) \le m-1$. This probability is equal to the probability that the nodes of ranks between j and m (inclusive) are arranged in π such that the node of rank m is last. That is,

$$\mathbf{E}(A_{j,m}) = \frac{(m-j)!}{(m-j+1)!} = \frac{1}{m-j+1}.$$

The expected number of new pairs is then

$$\sum_{j|j<m-1} \mathbf{E}(A_{j,m}) = \sum_{j|j<m-1} \frac{1}{m-j+1} = \sum_{3\le j<m} \frac{1}{j} = O(\log m) = O(\log n).$$

Since the number of new pairs created by ℓ' is the same as the number of new pairs created by ℓ, we conclude that the expected contribution of each swap to $\mathcal{P}$ is $O(\log n)$.

Our assumptions on the motion implies that the total number of swaps is $O(n^2)$. Therefore, we get that all swaps generate $O(n^2 \log n)$ additional pairs, so in total $\mathcal{P} = O(n^2 \log n)$. Using Lemma 3.3, our structure thus processes $O(n^2\beta_{s+2}(n) \log n)$ events, and is therefore *efficient*.

In summary, we have the following result.

Theorem 3.4. *Let S be a fixed set of n lines moving in the plane. Assuming that the motion of each point is semi-algebraic of constant description complexity, we can maintain the upper envelope of S in a randomized structure of linear size that processes an expected number of $O(n^2\beta_s(n) \log n)$ events, each in $O(\log^2 n)$ expected time, where s is the number of times where any fixed triple of lines can become concurrent. Each line participates at any given time in $O(\log n)$ certificates that the structure maintains.*

So far, this matches (but, as we argue, simplifies) the KDS structure of [8]. In the main contribution of the paper, presented in the next section, we turn this structure into a dynamic KDS that also supports insertions and deletions of lines.

4 Making the Data Structure Kinetic and Dynamic

We next adapt the treap data structure so that it can also efficiently support insertions and deletions of lines into/from the structure. First, we review the algorithms in [22] for inserting and deleting elements into/from a treap. To insert a new line ℓ, we create a new leaf, in a position determined by its rank. Then we draw a random priority for ℓ from the given distribution, and rotate the node storing ℓ up the tree, as long as its priority is larger than the priority of its parent. While rotating the node of ℓ up, we also re-compute the envelope of every node involved in a rotation from the envelopes of its children and from the line that it stores. After ℓ is located in its right place, we recompute the envelopes on the path from ℓ to the root in a bottom-up manner. The expected logarithmic depth of the treap implies that insertion takes $O(\log^2 n)$ expected time.

The implementation of a delete operation is similar. Let m be the line to be deleted. We keep rotating the edge connecting m to its child of largest priority, until m becomes a leaf. We then discard m and recompute the envelopes of all nodes involved in the rotations, in a bottom-up manner, until we reach the first node that did not contain m on its envelope or we reach the root. As in the case of insertion, deletion also takes $O(\log^2 n)$ expected time.

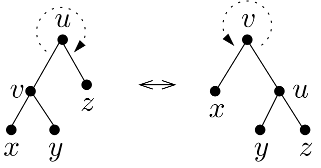


Figure 7: A rotation around the edge (u, v). The new nodes u and v retain the identities of the old nodes, as shown.

Consider for example a right rotation around an edge $(v, u = p(v))$ as shown in Figure 7. Node v before the rotation changes its right child to be u, and node u changes its left child to be y, previously the right child of v. The rotation introduces new pairs associated with v (and removes pairs associated with u). Similarly, a left rotation around $(v = p(u), u)$ introduces new pairs associated with u (and removes pairs associated with v). Therefore, we need to re-analyze the *efficiency* of the data structrure, when insertions and deletions are allowed, to take these changes into account. Using Lemma 3.3, we need to estimate the number of new pairs that are generated during an insertion or a deletion. We show below that the expected number of such pairs is $O(n)$, for each update operation. Hence, if $O(n)$ such operations are performed, starting with the empty set, they generate an expected number of $O(n^2)$ pairs, and thus create only $O(n^2\beta_{s+2}(n))$ new CE events. In other words, the bound on the expected number of internal events remains asymptotically the same.

We analyze deletion in detail; the analysis of insertion is analogous and hence omitted. Assume that m is the line to be deleted. We examine the rotations that bring m down, and bound the expected number of new pairs created by these rotations.

Let v be the node containing the line m. Let σ^ℓ denote the rightmost path from $\ell(v)$ to a leaf, and let σ^r denote the leftmost path from $r(v)$ to a leaf. Each edge on σ^ℓ and σ^r corresponds to a rotation. That is, as shown in Figure 8, a right rotation around the edge between v and $\ell(v)$ changes the left child of v to the next node on σ^ℓ. In this case, for each line x in $A_1 = S(\ell(\ell(v)))$, including $\mu(\ell(v))$, and for each line y in $S(r(v))$, the rotation introduces a new pair (x, y) in $P(\ell(v))$. These are the only new pairs that are generated. A left rotation around $(v, r(v))$ has a symmetric effect, and it changes the right child of v to be the next node along σ^r (emphasizing it yet another time, this is in accordance with the way nodes retain their identity in a rotation).

Let $p^\ell_1, \ldots, p^\ell_s$ be the nodes on σ^ℓ, and let $p^r_1, \ldots, p^r_t$ be the nodes on σ^r, in their top-down order.

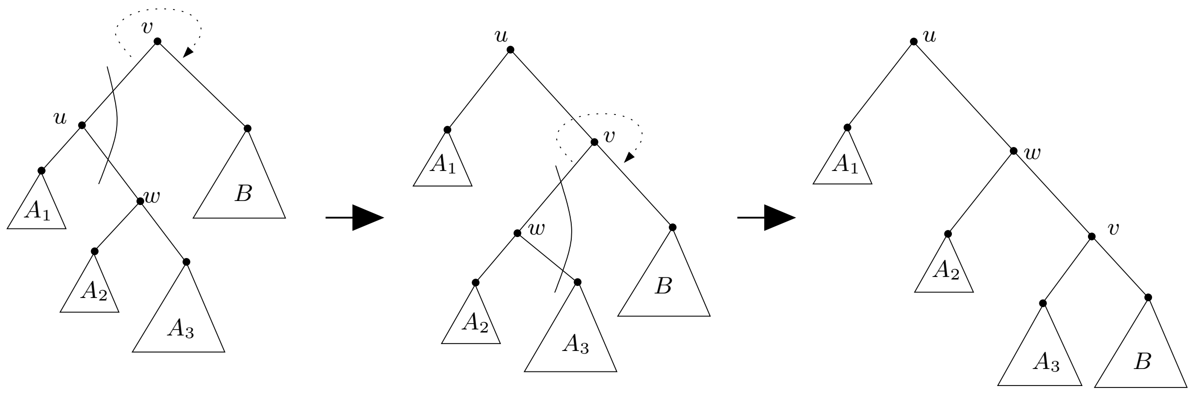


Figure 8: Rotating m down.

Set $A = S(\ell(v))$ and $B = S(r(v))$. Furthermore, set, for each i, $A_i = S(\ell(p_i^\ell))$, and $B_i = S(r(p_i^r))$. Suppose that all right rotations are performed first. As just discussed, the first right rotation creates the subset of new pairs $(A_1 \cup \{\mu(p_1^\ell)\}) \times B$, all added to $P(p_1^\ell = \ell(v))$, the second creates the subset $(A_2 \cup \{\mu(p_2^\ell)\}) \times B$, and so on. Similarly, if all left rotations are performed first, then the first left rotation creates the subset $A \times (B_1 \cup \{\mu(p_1^r)\})$, all of whose elements are added to $P(p_1^r = r(v))$, the second creates the subset $A \times (B_2 \cup \{\mu(p_2^r)\})$, and so on. See Figure 8. It is easy to see that if right rotations and left rotations are mixed, then each right rotation creates only a subset of the pairs it would have created if performed before all left rotations, and the same holds for left rotations. Therefore, regardless of the order of the rotations, the total number of new pairs is dominated by $|A \times B|$.

For $i < m < j$, define $B_{i,m,j}$ to be an indicator random variable, which is 1 if and only node $v(m)$ is the lowest common ancestor of $v(i)$ and $v(j)$. The expected size of $A \times B$, for a fixed node $v(m)$, is

$$\sum_{i,j|i<m<j} \mathbf{E}(B_{i,m,j}) \ .$$

For $B_{i,m,j}$ to be 1, $v(m)$ must have the largest priority among all nodes x, such that $i \le rank(x) \le j$. The probability of this event is equal to the probability that $v(m)$ ends up last in a random permutation of the nodes $\{x \mid i \le rank(x) \le j\}$. That is

$$\mathbf{E}(B_{i,m,j}) = \frac{(j-i)!}{(j-i+1)!} = \frac{1}{j-i+1},$$

for any $i < m < j$. Summing up over all such pairs i and j, we get that

$$\sum_{i,j|i<m<j} \mathbf{E}(B_{i,m,j}) = \sum_{i,j|i<m<j} \frac{1}{j-i+1} \le \sum_{2\le k\le n} (k-1)\frac{1}{k+1} = O(n).$$

That is, we have shown that the expected increase in the sum $\sum_v |P(v)|$, caused by inserting or deleting an element (at place m) is $O(n)$. Following the preceding discussion, we thus obtain:

Theorem 4.1. *Let S be a fully dynamic set of n lines moving in the plane, where the motion of each line is semi-algebraic of constant description complexity. Assume also that S is subject to*

$O(n)$ insertions and deletions. We can maintain the upper envelope of S in a randomized structure of linear size, that processes an expected number of $O(n^2\beta_{s+2}(n)\log n)$ events, each in $O(\log^2 n)$ expected time, where s is the number of times where any fixed triple of lines can become concurrent. Each line participates at any given time in $O(\log n)$ certificates that the structure maintains.

The standard duality that we have used between lines and points yields the primal version of the preceding theorem.

Theorem 4.2. *Let S be a fully dynamic set of n lines moving in the plane, where the motion of each line is semi-algebraic of constant description complexity. Assume also that S is subject to $O(n)$ insertions and deletions. We can maintain the convex hull of S in a randomized structure of linear size, that processes an expected number of $O(n^2\beta_{s+2}(n)\log n)$ events, each in $O(\log^2 n)$ expected time, where s is the number of times where any fixed triple of points can become collinear. Each point participates at any given time in $O(\log n)$ certificates that the structure maintains.*

Remarks. (1) Note that, in accordance with the discussion in Section 3, here we need to use $\beta_{s+2}(n)$ in the bounds, rather than the slightly improved factor $\beta_s(n)$ that has been derived in the non-dynamic case.
(2) In Theorems 4.1 and 4.2, we assume that there are only $O(n)$ insertions and deletions. If the number of insertions and deletions is $N \gg n$, then the preceding analysis shows that the number of additional external events created by these updates is $O(Nn\beta_{s+2}(n))$. This, times a logarithmic factor due to swaps, is easily seen to also bound the number of internal events, which makes the structure efficient also in this case.

References

[1] P.K. Agarwal, J. Basch, M. de Berg, L. Guibas, and J. Hershberger, Lower bounds for kinetic planar subdivisions, *Discrete Comput. Geom.* 24 (2000), 721–733.

[2] P. K. Agarwal, M. de Berg, J. Gao, L. J. Guibas, and S. Har-Peled, Staying in the middle: Exact and approximate medians in $\mathbf{R}^1$ and $\mathbf{R}^2$ for moving points, manuscript, 2003.

[3] P. K. Agarwal, J. Erickson, and L. Guibas, Kinetic binary space partitions for intersecting segments and disjoint triangles, *Proc. Ninth Annu. ACM-SIAM Sympos. Discrete Algo.* (1998), 107–116,

[4] P.K. Agarwal, J. Gao, and L. Guibas, Kinetic medians and *kd*-trees, *Proc. European Sympos. Algo.* (2002), 5–16. Lecture Notes in Comput. Sci., 2461, Springer Verlag, Berlin, 2002.

[5] P. K. Agarwal, L. Guibas, J. Hershberger, and E. Veach, Maintaining the extent of a moving point set, *Discrete Comput. Geom.* 26 (2001), 353–374.

[6] P. K. Agarwal, L. Guibas, T. M. Murali, and J. S. Vitter, Cylindrical static and kinetic binary space partitions. *Comput. Geom. Theory Appls.* 16 (2000), 103–127.

[7] J. Basch, J. Erickson, L. Guibas, J. Hershberger, and L. Zhang, Kinetic collision detection between two simple polygons, *Comput. Geom. Theory Appls.* 27 (2004), 211–235.

[8] J. Basch, L. J. Guibas, and J. Hershberger, Data structures for mobile data, *J. Algorithms* 31 (1999), 1–28.

[9] M. de Berg, Kinetic dictionaries: how to shoot a moving target, *Proc. European Sympos. Algo.* (2003), 172–183, Lecture Notes in Comput. Sci., 2832, Springer Verlag, Berlin, 2003.

[10] T.M. Chan, On levels in arrangements of curves, II: A simple inequality and its consequences, *Proc. 44th IEEE Sympos. Foundat. Comput. Sci.*, 2003, 544–550.

[11] A. Czumaj and Ch. Sohler, Soft kinetic data structures, *Proc. Twelfth Annu. ACM-SIAM Sympos. Discrete Algo.* (2001), 865–872.

[12] T. Dey, Improved bounds for planar k-sets and related problems, *Discrete Comput. Geom.* 19 (1998), 373–382.

[13] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu, Discrete mobile centers, *Discrete Comput. Geom.* 30 (2003), 45–63.

[14] L. Guibas, Kinetic data structures: a state of the art report. *Robotics: the Algorithmic Perspective* (WAFR 1998), 191–209, A.K. Peters, Natick, MA, 1998.

[15] L. Guibas, J. Hershberger, S. Suri, and L. Zhang, Kinetic connectivity for unit disks. *Discrete Comput. Geom.* 25 (2001), 591–610.

[16] J. Hershberger, Kinetic collision detection with fast flight plan changes, *Inform. Process. Lett.* 92 (2004), 287–291.

[17] J. Hershberger and S. Suri, Simplified kinetic connectivity for rectangles and hypercubes, *Proc. Twelfth Annu. ACM-SIAM Sympos. Discrete Algo.* (2001), 158–167.

[18] M. I. Karavelas and L. Guibas, Static and kinetic geometric spanners with applications, *Proc. Twelfth Annu. ACM-SIAM Sympos. Discrete Algo.* (2001), 168–176.

[19] D. Kirkpatrick, J. Snoeyink, and B. Speckmann, Kinetic collision detection for simple polygons, *Internat. J. Comput. Geom. Appls.* 12 (2002), 3–27.

[20] D. Kirkpatrick and B. Speckmann, Separation sensitive kinetic separation structures for convex polygons, *Proc. Sympos. Discrete Comput. Geom.* (Tokyo, 2000), 222–236, Lecture Notes in Comput. Sci., 2098, Springer Verlag, Berlin, 2001.

[21] M. H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Computer Syst. Sci.* 23 (1981), 166–204.

[22] R. Seidel and C. R. Aragon, Randomized search trees, *Algorithmica* 16 (1996), 464-497.

[23] M. Sharir and P.K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[24] G. Tóth, Point sets with many k-sets, *Discrete Comput. Geom.* 26 (2001), 187–194.