To appear in the 1997 *ACM-SIAM Symposium on Discrete Algorithms*

Data Structures for Mobile Data

Julien Basch *Leonidas J. Guibas*
Computer Science Department
Stanford University
Stanford, CA 94305, USA
{jbasch,guibas}@cs.stanford.edu

John Hershberger
Mentor Graphics Corp.
1001 Ridder Park Drive
San Jose, CA 95131, USA
john_hershberger@mentorg.com

1 Introduction

We present a set of novel data structures for the efficient maintenance of various attributes of mobile data. For example, given n points continuously moving in the plane, we give methods for maintaining their convex hull or their closest pair. We call these attributes *configuration functions* of the mobile data. Since motion is common with objects in the physical world, the examples we discuss in this paper come primarily from computational geometry and are motivated by problems like collision detection in robotics or animation, visibility determination in computer graphics, etc. Our techniques, however, are more generally applicable to the processing of discrete events associated with any kind of continuously changing data. We call our data structures *kinetic*, to distinguish them from their more classical static or dynamic (in the other sense, as we explain below) counterparts, and we abbreviate the term "kinetic data structure" to KDS for short. We will call *kinetization* the process of transforming an algorithm on static data into a data structure that is valid for continuously changing data.

The problems of convex hull and closest pair maintenance have been exhaustively studied in computational geometry [5, 6, 10, 13, 14, 17, 19], but almost exclusively in the context of static objects with operations like insertion and deletion. Our emphasis instead is on the maintenance of such configuration functions under *continuous motions* of the given objects. Though in principle the continuous motion of a single object can be approximated, after a discrete sampling of time, by deleting it and reinserting it at a new position at each time step, this method is clearly ill-adapted to our purposes and wasteful of computation. The aim of our technique is to take full advantage of the coherence present in continuous motions so as to process a minimal number of combinatorial events. In this respect, the way of analyzing our data structures is akin to the *dynamic computational geometry* framework introduced by Atallah [4] in order to study the number of combinatorially distinct configurations of a given kind (e.g., convex hull or closest pair) that arise during the continuous motion of geometric objects. Unlike Atallah's scheme, however, our data structures do not require us to know the full motion of the objects in advance. Thus they are better suited to real-world situations in which objects can change their motion on-line because of interactions with each other, external impulses, etc.

We assume that each moving object has a posted *flight plan* that gives full or partial information about its current motion. As mentioned above, flight plans can change. A flight plan *update* can occur because of interactions between our object and other moving objects, the environment, etc. For example, a collision between two moving objects will in general result in updates to the flight plans of both objects. The interface between our kinetic data structures and the object motions is through a global event queue. Thus our techniques most closely resemble sweep-line and -plane methods in computational geometry, except that in our case the dimension being swept over is time. A key aspect of our data structures is that we have a "narrow interface" to the motion. What we mean by this is that the kinds of events we have in our event queue correspond to possible combinatorial changes involving a constant (and typically small) number of objects each. For example, in the case of 2-D convex hull maintenance, one type of event we will use is "the points A, B, C become collinear" or, equivalently, "the triangle ABC reverses sign (orientation)". Indeed, it will turn out that the correctness of whatever configuration function we maintain can be guaranteed with a conjunction of such low-degree algebraic sign conditions involving a bounded number of objects each—we

will call these conditions the *certificates* of the KDS.

At any one time, our event queue will contain several KDS events corresponding to times when certificates might change sign. The times for these events are calculated using the posted flight plans of the objects involved. If, because of other events, the flight plan of an object is updated, then all certificates involving that object must be located and have their 'sign change' time recalculated according to the new plan. In this way the event queue adapts to the evolving motions of the objects. We can deal in the same way with objects whose flight plan is only partially known. In our "sign of the triangle ABC" example above, given some partial bounds on the positions and velocities of the points A, B, C, we can easily calculate a time interval Δt during which we can be sure that the sign of ABC does not change. Thus we can schedule an event to occur after Δt time units, and at that point we can recheck the sign of ABC and proceed similarly (after updating our knowledge of the motions of the participating points). In general our philosophy will be that each moving object needs to be aware of all the events in the event queue that involve it and the validity assumptions about its motion on which these events are based. If the motion of the object changes so that any of these assumptions is no longer valid, then it is the responsibility of the object to take the steps necessary to have these events rescheduled at the times appropriate for its new motion.

We will analyze and evaluate a kinetic data structure by counting the worst-case number of combinatorial events that need to be processed when the object motions are fully known and parameterizable by what we call *pseudo-algebraic* functions of time. These are functions with the property that each of the elementary predicates involved in the kinetization changes sign at most a bounded number of times—very much in the spirit of situations in which Davenport-Schinzel sequences have been used [22]. We will call this number the *cost* of the KDS. We make a distinction between *external events*, i.e., those affecting the configuration function we are maintaining (e.g., convex hull or closest pair), and *internal events*, i.e., those processed by our structure because of its internal needs, but not affecting the desired configuration function. Our aim will be to develop kinetic data structures for which the total number of events processed by the structure in the worst case is asymptotically of the same order as, or only slightly larger than, the number of external events in the worst case. This is reasonable, as the number of external events is a lower bound on the cost of any algorithm for maintaining the desired configuration. A KDS meeting this condition will be called *efficient*. We define the *size* of a KDS to be the maximum number of events it needs to schedule in the event queue at any one time. We call a structure *compact*, if its size is roughly linear in the number of moving objects. Finally, we define the *degree* of a KDS as the maximum number of events in the event queue that depend a single object, and call a KDS *local* when that number is polylogarithmic in the number of moving objects involved. This property is crucial for fast handling of flight plan updates.

To summarize, our kinetic data structures are different from classical dynamic data structures: though we can (and often want to) accommodate insertions and deletions, our focus is on continuous motions and not modifications. We can use Atallah's framework of dynamic computational geometry to get lower bounds on the amount of work we have to do. But our structures are on-line and can be used to implement correct simulations even when the object flight plans change because of interactions between the objects themselves or the objects and the environment, or even when only partial information about the motions is available. Furthermore, we provide some general tools for the kinetization of static algorithms that lead to KDSs that are easy to analyze and perform well.

1.1 An illustrative example

To make the issues above more concrete, let us consider the following simple 1-D situation. Given a set of points moving continuously along the y-axis, we are interested in knowing at all times which is the topmost point (the largest, if we think of the points as numbers). If two points collide, we allow them to pass each other without interaction. Suppose further that we know that the points are moving with constant velocities (but possibly a different one each), starting from an arbitrary initial configuration.

If we draw the trajectories of the points in the ty-plane (where the t axis is horizontal and denotes time), then our problem is nothing but computing the upper envelope of a bunch of straight lines in the plane (or at least the part of it that is after the initial time t_0). This upper envelope computation can be trivially done in $O(n \log n)$ time with a divide and conquer algorithm (this bound holds even if points can appear and disappear at arbitrary times, but then it is not trivial [12]). In the worst case, the number of times during the motion that the topmost point changes is $\Theta(n)$. Thus we have a method for computing the configuration function of interest in time that is is only a logarithmic factor higher than the maximum number of changes in the configuration function itself.

For our purposes, however, this solution is unsatisfactory, because it is really based on knowing in advance the full motions of the points. What we seek is a strategy that works on-line and can accommodate flight plan updates. So suppose instead that we try to maintain the

sorted order of our points along the y-axis, on-line. For every pair of points that are currently consecutive along the y-axis we schedule an event that is the first time when these points cross (or if, as above, our knowledge of the motions is incomplete, we schedule an event based on our estimate of how long we can be sure that the relative order of the points does not change). In a manner entirely reminiscent of sweep-line algorithms, when two adjacent points pass through each other, this destroys two old adjacencies and creates two new ones along the sorted list. Thus we de-schedule (up to) two events and schedule (up to) two new events. In this process we always maintain the sorted list of points, and in particular we always know the topmost one as well. Unfortunately, although the kinetic data structure obtained is local, it may have to process $\Theta(n^2)$ events even when the points have the simple motion described above (imagine that half the points are stationary, and the other half pass over them). Thus the number of internal events here is an order of magnitude greater than those affecting the configuration function we are interested in—this solution is not efficient.

A third structure, and one that lets us meet all our objectives, is to maintain the moving points in a heap, with the root being the topmost (maximal) element. The kinetization of the heap is as follows. As the points move, each of the links in the heap may generate an event in the event queue corresponding to when the two points involved change their order. Notice that, if we assume that no two pairs of points meet at the same time (non-degeneracy), then when a parent and a child point in the heap change their order, we can just interchange their locations in the heap and still have a valid heap on all the data—all the other heap inequalities are still valid, and so are the ones involving the crossing points, because at the moment they cross their y-coordinates are the same (here we are making strong use of the continuity of the motions). As in the sweep-line-like argument above, when a swap of two elements happens in the heap, up to four adjacency (parent/child) relationships can change in the heap, so we may have to de-schedule four events and reschedule four more. This describes our *kinetic heap*, which maintains the topmost element at all times.

But how many events does the kinetic heap have to process in the worst case, when the points move with constant velocities? This question turns out to be surprisingly non-trivial; we can show by a potential argument that the kinetic heap under linear point motions processes $O(n \log^2 n)$ events, and thus is a data structure meeting our requirements (the proof is omitted from this version of the paper).

To prepare ourselves for the solutions to the other problems we will present below, let us also consider the following fourth solution to the kinetic maximum maintenance problem. Consider first an algorithm that computes the maximum of n (static) numbers. The algorithm computes the maximum recursively, by partitioning the numbers into two approximately equal-sized groups (arbitrarily), computing the maximum of each subgroup, and then comparing the two winners to select the final true maximum. If viewed from the bottom up, this is exactly a tournament for computing the global leader. In the end this algorithm has performed $O(n)$ comparisons that prove that the maximum it computed is indeed the true maximum. Now imagine that our numbers start varying—our points can move. As long as each of the comparisons the algorithm made stays valid, the identity of the maximum element cannot change.

A general kinetization strategy we will use consists of taking the certificates of correctness in the computation performed by our static algorithm—the comparisons in this case—and associating with each of them an event in the global queue that describes when that certificate will (may) be violated in the future. When a violation happens, we hope that there will be a simple and efficient way to update the output of the algorithm and the set of certificates to be maintained. In our example, suppose that a particular comparison involved in the maximum computation flips. This comparison is between the leaders of two subgroups at a certain level of the tournament tree. If the winner changes, then this winner has to be percolated up the tournament tree, till it is either defeated or declared the overall maximum. But because a tournament tree is balanced, this computation takes only $O(\log n)$ time and can affect also at most $O(\log n)$ existing certificates (a constant number of de-schedulings and new schedulings per level). We call this fourth structure a *kinetic tournament*.

If our points move with constant velocities, how many events will our kinetic tournament have to process? The key insight to answering this question is to realize that the kinetic tournament is implementing a divide-and-conquer algorithm for the computation of the upper envelope of n straight lines in the ty-plane (the point trajectories). For example, the comparisons performed over time at the top level for declaring the final leader are exactly those needed to merge the upper envelopes of the two subgroups of the lines. The overall cost of the merge is easily seen to be $O(n)$. Thus this divide-and-conquer way of implementing the upper envelope computation has a worst case cost satisfying the recurrence $C(n) = 2C(n/2) + \Theta(n)$, which solves to $C(n) = O(n \log n)$. The number of kinetic tournament events (reschedulings, etc.) is proportional to the number of times the identity of one of the contestants at a node of the tournament tree changes. Each such identity change corresponds to an intersection in one of the sub-envelopes computed by the divide-and-conquer algorithm, and hence is counted by the $O(n \log n)$ bound

on $C(n)$. Therefore the kinetic tournament accomplishes our goal of maintaining on-line the maximum of a set of moving points, and it is an efficient, compact, and local KDS. If we use a priority queue to store the relevant events and perform a discrete-time simulation, then the event counts for all the structures described here can be made into run-times with an extra $O(\log n)$ factor (the priority queue cost).

1.2 Previous results and summary of the work

A number of works in the early eighties [4, 8, 16] considered the problem of *computing* a configuration function of moving points. In all cases, the motion was considered fully known, and the problem was typically cast and solved in one dimension higher. The method of Edelsbrunner and Welzl [8] for computing the k-th order statistic of a set of points moving at constant speed along the x-axis (introduced as a motivation for computing the k-level of an arrangement of lines) is most similar to a KDS.

More recently, questions concerning the maintenance of the Voronoi diagram of moving points (or its dual, the Delaunay triangulation) have received extensive attention [7, 9, 11, 20]. The significance of our work is best understood in comparison. The Delaunay triangulation contains a proof of its correctness involving only four-point certificates for each of the edges of the triangulation. In that sense, it is what we might call a *self-certifying* structure. As such, its kinetization is immediate: we need only maintain a certificate for each of the edges. Whenever any certificate changes sign, we know that we can update the triangulation (and the corresponding certificate structure) by an edge-flip on the failing edge. The structure has no internal events, hence the issue of efficiency does not arise. It is also well known that the Delaunay triangulation can be used to compute both the convex hull and the closest pair, so that we readily have a common kinetic data structure to maintain these configuration functions (closest pair maintenance requires in addition a kinetic tournament on the edge lengths), but this solution has two drawbacks: it is not local (a point can be a vertex of linearly many triangles), nor known to be efficient (the tightest upper bound known on the number of changes to the Delaunay triangulation of points in algebraic motion is roughly cubic in the number of points [11], whereas the convex hull and the closest pair can change roughly a quadratic number of times). In general, one can view the process of kinetization as 'sufficiently augmenting a configuration function to make it self-certifying.'

Algorithms for collision detection in robotics by Lin and Canny [15] and Ponamgi et al. [18] exploit temporal coherence to maintain the minimum distance between all pairs of moving objects, but their approach retests the validity of separating planes at every step, and recalculates these separators from scratch when the old ones fail.

We have applied the methodology described above to a number of problems in 2-D computational geometry. In this paper, we present in some detail efficient, compact, and local kinetic data structures for two important and common configuration functions, giving representative examples of the kinetization process. Convex hull maintenance (Section 2) calls upon some deep theorems of combinatorial geometry to prove the efficiency of the structures we develop. Closest pair maintenance (Section 3) requires the development of a novel static algorithm, and specialized data structures to handle events efficiently. In Section 4, we take up some further issues generated by this framework for mobile data and present plans for further work.

Due to lack of space, we omit some proofs and algorithmic details, which will appear in the full paper. Other kinetizations will also be presented in the full paper, including vertical cell decomposition of moving shapes, smallest vertical distance between axis-aligned moving rectangles, and other configuration functions for moving points.

2 2-D convex hull

In this section, we present an efficient kinetic data structure to maintain the convex hull of a set of moving points in the plane. Following our general strategy for kinetization, we first describe a static algorithm and its certificate structure, simplify these certificates to attain certain desirable properties, and then show how to maintain the certificate structure once the points start moving.

Before we proceed, we dualize the problem, as the algorithm is a bit more natural to describe in the dual setting. We focus here on computing the upper convex hull, and dualize each point (p, q) to the line $y = px + q$. In the dual, the goal is to maintain the upper envelope of a family of lines whose parameters change in a continuous, predictable fashion. We will perform the kinetization of the $O(n \log n)$ divide and conquer algorithm mentioned in Section 1.1 for the analysis of the kinetic tournament: we divide the set of n lines into two subsets of roughly equal size, compute their upper envelopes recursively, and then merge the two envelopes. To focus on the merge step, we first study how to maintain the upper envelope of two convex piecewise linear univariate functions.

2.1 Upper envelope of convex functions

Consider two convex piecewise linear univariate functions, one red and one blue, each given by a linked list of vertices and edges ordered from left to right. As the supporting lines are the primary elements in our problem, we denote by ab the vertex at the intersection of lines a and b along one of the functions. For such a vertex, there is an edge in the other envelope that is either below or above it, which we call its *contender* edge and denote $\texttt{ce}(ab)$. The computation of the joint upper envelope consists of sweeping the two functions from left to right, and outputting in sequence each vertex that is above its contender edge, and each red-blue intersection (easily discovered in the process). We assume the presence of sentinel vertical lines at infinity to avoid special cases for the extremes. We denote by $\chi(\cdots)$ the color (red or blue) of a vertex or edge, and assume that the merged list of red and blue vertices is maintained as a doubly linked list ordered by x-coordinates, with the fields $ab.\texttt{next}$ and $ab.\texttt{prev}$ to navigate this list.

Hence, the comparisons done by the sweep are of two types: x-certificates proving the horizontal ordering of vertices, denoted by $<_x$, and y-certificates proving the vertical position of a vertex with respect to an edge, denoted by $<_y$. Unfortunately, if we were to keep all these comparisons as certificates, the kinetic data structure thus obtained would not be local, as a given edge could be the contender of linearly many vertices from the other envelope. We thus build an alternative list of certificates that also involves comparisons between line slopes, denoted by $<_s$. The following table gives this modified list of certificates for a given configuration. The first column contains the name of a certificate, the second column contains the comparison that this certificate guarantees, and the third column contains additional conditions for this certificate to be present in the KDS.

Cert.	*Comparison*	*Condition(s)*
$\texttt{x}[ab]$	$ab <_x ab.\texttt{next}$	$\chi(ab) \neq \chi(ab.\texttt{next})$
$\texttt{yli}[ab]$	$ab <_y$ or $>_y \texttt{ce}(ab)$	$b \cap \texttt{ce}(ab) \neq \emptyset$
$\texttt{yri}[ab]$	$ab <_y$ or $>_y \texttt{ce}(ab)$	$a \cap \texttt{ce}(ab) \neq \emptyset$
$\texttt{yt}[ab]$	$\texttt{ce}(ab) <_y ab$	$a <_s \texttt{ce}(ab) <_s b$
$\texttt{slt}[ab]$	$a <_s \texttt{ce}(ab)$	$\texttt{ce}(ab) <_y ab$
$\texttt{srt}[ab]$	$\texttt{ce}(ab) <_s b$	
$\texttt{sl}[ab]$	$b <_s \texttt{ce}(ab)$	$b <_s \texttt{ce}(ab)$
		$ab <_y \texttt{ce}(ab)$
$\texttt{sr}[ab]$	$\texttt{ce}(ab) <_s a$	$\texttt{ce}(ab) <_s a$
		$ab <_y \texttt{ce}(ab)$

The certificates have the following meaning: (1) The exact x-ordering of vertices is recorded with $\texttt{x}[\cdots]$ certificates. (2) Each intersection is surrounded by $\texttt{yli}[\cdots]$ and $\texttt{yri}[\cdots]$ certificates ("y left/right intersection"). (3) If an edge is not part of the upper envelope, some certificates remember its slope relative to the edges that cover it, with three "tangent" certificates ($\texttt{yt}[\cdots], \texttt{slt}[\cdots], \texttt{srt}[\cdots]$) or one proving there is no tangent ($\texttt{sl}[\cdots]$ or its symmetric $\texttt{sr}[\cdots]$). Illustrations of the certificates appear in Figure 1.

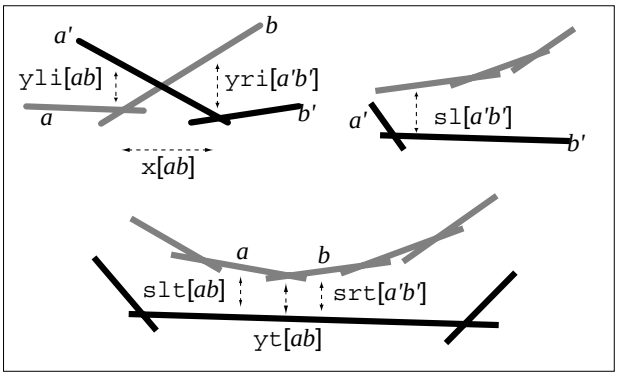

Figure 1: Illustration of the certificate definitions.

Lemma 2.1 *Consider a configuration of two convex piecewise linear functions and the certificate list for their upper envelope as defined above. This upper envelope has the same combinatorial description in any other configuration in which all these certificates have the same signs.*

Proof: Omitted from this version of the paper. □

Hence, the certificate list described above is sufficient to maintain the upper envelope. As in the case of any kinetic data structure, all these certificates are placed in a global event queue, where each certificate is stamped with the time at which it is scheduled to change outcome. When the first event of the queue changes outcome, this requires the modification of some certificates; it can be checked that any such event requires only the modification of $O(1)$ certificates (see Figure 2 for a partial list of cases).

When a y-certificate changes outcome, this modifies the output: either two neighbor vertices merge into one, or the reverse. Hence, for the purpose of the recursive construction, it is necessary to be able to handle such local structural changes in the input, and it can be checked that this also changes $O(1)$ certificates.

2.2 Divide and conquer upper envelope

To kinetize the divide and conquer algorithm, we keep a record of the entire computation in a balanced binary tree. A node in this tree is in charge of maintaining the upper envelope of the two upper envelopes computed by its two children. If an event triggers a change in the output of a node, this node passes on the event to its parent, as a local structural change in the input, and

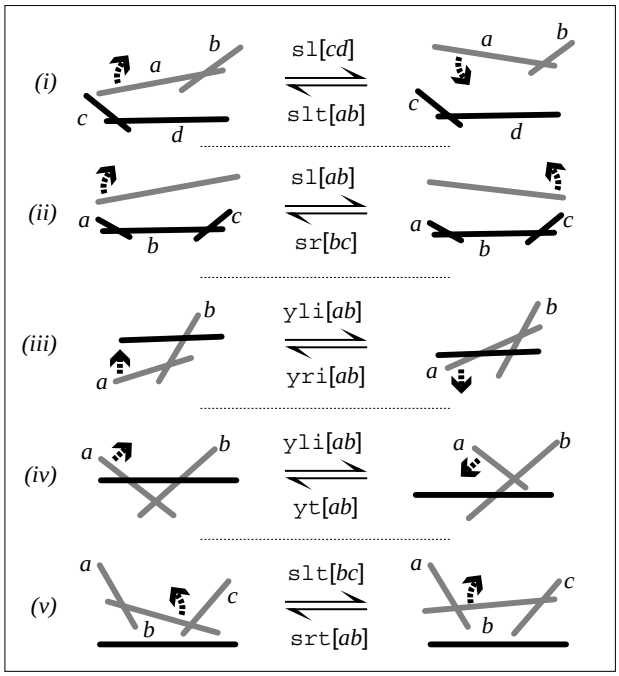

Figure 2: A partial list of events. The certificate that becomes invalid is indicated for each transition. There are three additional cases not shown: *(i)* and *(iii)* in mirror image, and the event corresponding to the invalidation of an x-certificate.

so on to upper levels of the computation tree while this change remains visible.

As in the case of the one-dimensional kinetic tournament data structure for known motion, we analyze efficiency by considering time as an additional static dimension and charging each event to a feature of a three-dimensional structure with known worst case complexity. The primal version of the problem is ill-suited for such an analysis, as the static structure described by the convex hull over time is not the convex hull of the trajectories of the underlying points. On the other hand, in the dual, the structure described by the upper envelope over time is exactly the upper envelope of the set of pseudo-algebraic surfaces described by the underlying lines. We can thus use results proving near-quadratic complexity for the upper envelope of pseudo-algebraic surfaces [21]. We also make use of the recent result of Agarwal, Schwarzkopf, and Sharir [1] about the near-quadratic complexity of the overlay of the projections of two upper-envelopes, to obtain sharp bounds on the number of events due to x-certificates.

Theorem 2.2 *The KDS for maintaining the convex hull is efficient, compact, and local.*

Proof: We first focus on the events attached to a specific node of the computation tree that involves a total of n red and blue lines. Consider time as a static third dimension: a line describes a pseudo-algebraic surface. The blue (red) family of lines is now a family of bivariate pseudo-algebraic functions. Looking at the upper envelopes of the blue and red families, and at their joint upper envelope in turn, we observe that a red-blue vertex on this surface corresponds to a change of sign of a y-certificate (a "y-event") in the kinetic interpretation. A monochromatic vertex corresponds to the appearance/disappearance of an edge triggered by some descendant in the computation tree. As our functions are pseudo-algebraic and satisfy Davenport-Schinzel type conditions, their upper envelope has complexity $O(n^{2+\epsilon})$ for any $\epsilon > 0$ [21], and therefore the number of events due to y-certificate sign changes is bounded by this quantity.

Consider now the events corresponding to the x reordering of two vertices of different colors (called "x-events"). In the 3-dimensional setting, a blue envelope vertex becomes an edge of the blue surface upper envelope. Hence, an x-event corresponds to a point (x, t) above which there is an edge in both the blue and the red upper envelopes. In other words, each x-event is associated with a bichromatic vertex in the overlay of the projections of the red and blue upper envelopes on the xt-plane ($y = 0$). If there are n bivariate surfaces in total, the complexity of this overlay is also $O(n^{2+\epsilon})$ for any $\epsilon > 0$ [1]. Hence, there are at most that many x-events.

Finally, each pair of lines becomes parallel a constant number of times, so there are $O(n^2)$ slope events attached to the node we have been focusing on up to now.

Getting back to the full computation tree, we conclude that the total number of events $C(n)$ satisfies the recurrence $C(n) = 2C(n/2) + O(n^{2+\epsilon})$, and therefore $C(n) = O(n^{2+\epsilon})$. In the worst case, the convex hull of n points in algebraic motion changes $\Omega(n^2)$ times. Hence our KDS is efficient. A straightforward counting argument proves compactness and locality. □

3 Closest pair in 2-D

In this section we describe a kinetic data structure for maintaining the closest pair in a set S of moving points in the plane. The static algorithm on which the kinetic data structure is based is a sweep-line algorithm, augmented to record its history in a combinatorial structure. As the points move, the kinetic algorithm updates this combinatorial structure so that it always reflects what the sweep-line algorithm would do if applied to the current configuration of points.

3.1 The sweep-line algorithm

The static closest-pair algorithm is based on the idea of dividing the space around each point into six 60° wedges. It is a trivial observation that the nearest neighbor of each point is the closest of the nearest neighbors in the six wedges. We show that a relaxed definition of nearest neighbor in each wedge (using one dimension of separation instead of two) is still sufficient to find the closest pair. The relaxed definition lets us compute neighbors efficiently, and aids in the kinetization of the algorithm.

We define the *dominance wedge* of a point p, call it $Dom(p)$, to be the right-extending wedge bounded by the lines through p that make $\pm 30°$ angles with the x-axis. The dominance wedge is defined to be open on the bottom and closed on top (it includes its upper boundary, but not its lower boundary). We define $Circ(p, r)$ to be the circle with radius r centered on point p. The distance between two points p and q is simply denoted pq.

Our algorithm uses all three right-extending wedges bounded by the vertical line through p and by the $\pm 30°$ lines, but we frame our arguments in terms of the single dominance wedge that contains the point $(\infty, 0)$. The same arguments apply to the other wedges by rotation.

Let the closest pair of points in S be (a, b), with a to the left of b (or below b, if their x-coordinates are equal). For notational convenience, we write this as $a \prec b$. Without loss of generality, assume that $b \in Dom(a)$; if this is not the case, then consider the $\pm 60°$ rotated plane that puts b in $Dom(a)$. Figure 3 illustrates the proof ideas behind the following lemmas.

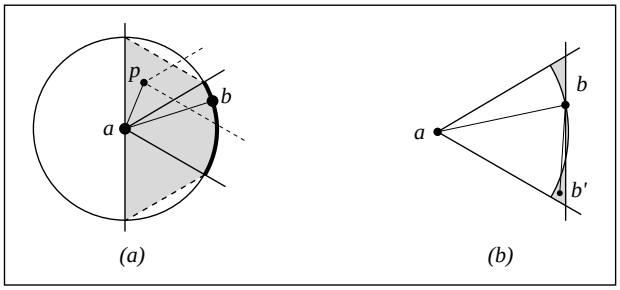

Figure 3: If (a, b) is the closest pair and $b \in Dom(a)$, then: (a) there is no point p to the right of a that dominates b, as such p would lie in the shaded region and be closer to a than b is; (b) point b is also the leftmost point in $Dom(a)$—any point b' left of b would be closer to b than a is.

Lemma 3.1 *Point b is not contained in $Dom(p)$ for any third point p with $a \prec p$.*

Lemma 3.2 *The leftmost point of S in $Dom(a)$ is b.*

For any point p, let $Maxima(p)$ consist of the points of S on the boundary of

$$\bigcup_{\substack{q \in S \\ p \prec q}} Dom(q).$$

We define the set of *candidates* associated with p, $Cands(p)$, to be the set $Maxima(p) \cap Dom(p)$. We denote the leftmost of these by $lcand(p)$. See Figure 4. By Lemmas 3.1 and 3.2, we have $b = lcand(a)$ for the closest pair (a, b).

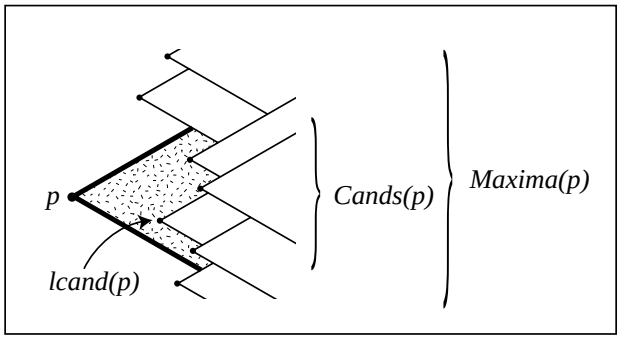

Figure 4: The sets of points $Maxima(p)$ and $Cands(p)$, and the leftmost candidate $lcand(p)$.

The sweep-line algorithm performs the following steps three times, once on the untransformed points of S, once on S rotated around the origin by $+60°$, and once on S rotated by $-60°$. Each of these rotations brings one of the three families of right-extending wedges into the central position, bounded by $\pm 30°$ lines.

1. Initialize a y-ordered list of points *Maxima* to $\emptyset$.
2. For each point $p \in S$ from right to left,
   (a) Set $Cands(p) = Maxima \cap Dom(p)$.
   (b) Set $lcand(p)$ to be the leftmost element of $Cands(p)$.
   (c) Delete the points of $Cands(p)$ from *Maxima*.
   (d) Insert p into *Maxima* at its proper place in y-order.

At the end of this procedure, repeated for all three directions, one of the three sets of $(p, lcand(p))$ pairs it produces contains the closest pair (a, b). The algorithm can be implemented to run in $O(n \log n)$ time, as the full paper will show.

3.2 Kinetization

To make the sweep-line algorithm kinetic, we need to transform it into a static data structure that represents the action of the sweep-line algorithm. We also need a set of certificates to show that the data structure is valid for the current set of points.

We define the *maxima diagram* to be the union, over all points p, of the boundary of the part of $Dom(p)$ that lies outside $\cup_{q \in Maxima(p)} Dom(q)$. Each point of S is the left endpoint of two segments in the maxima diagram that extend from p to the boundaries of $Dom(q)$ and $Dom(q')$, for two points q and q' in $Maxima(p)$. We say that q and q' are the *targets* of p in the maxima diagram.

We use as certificates three sorted orders: the projections of the points in S on the x-axis and on the lines that make an angle of $\pm 60°$ with the x-axis. Each point belongs to up to six certificates, involving its two neighbors in each of the three sorted orders. We also use certificates for a kinetic tournament, described below.

Lemma 3.3 *If two configurations of S have all three orders equivalent, then for each p, $Maxima(p)$, $Cands(p)$ and $lcand(p)$ are the same in the two configurations.*

Proof: By induction from right to left. □

The maxima diagram can undergo a linear number of changes when a pair of points swaps in one of the three linear orders. However, the changes to the maxima diagram can be represented in an implicit data structure that requires only $O(\log n)$ updates per swap. For this purpose, we keep two auxiliary data structures, representing two separate one-to-many relations.

For each $p \in S$, we keep $Cands(p)$, the intersection of $Maxima(p)$ with $Dom(p)$, as a sequence of points ordered by y coordinate. This sequence is stored in a balanced binary tree and supports the usual searching and update operations. In addition, each node of the tree has a pointer to its parent in the tree, and the root of the tree for $Cands(p)$ points to p. Thus each point of $q \in S$ can find the point $p \in S$ whose candidate it is, $q \in Cands(p)$, in $O(\log n)$ time. Each node in a $Cands()$ tree also keeps track of the leftmost point in its subtree, and so the root of $Cands(p)$ records $lcand(p)$. The parent pointers can be maintained as part of the standard tree update operations, within the same asymptotic time bound, as can the "leftmost" fields. As part of our algorithm, we will make sure that the "leftmost" fields are maintained correctly whenever the x-order of points changes.

The second auxiliary data structure, $Parents(p)$, records in an ordered sequence all the points for which p is a target in the maxima diagram. The sequence can be divided into the points above p, denoted $Parents_a(p)$, and the points below p, denoted $Parents_b(p)$. In each of the two subsequences, the points appear in the order in which their edges hit $Dom(p)$, which is the same as their x-order. The sequence $Parents(p)$ is stored in a balanced binary tree with parent pointers, so for each of the two edges extending from a point q in the maxima diagram, we can find the point p for which $q \in Parents(p)$ in logarithmic time. $Parents(p)$ is required to support the $\pm 60°$ exchanges described below, though it is not needed for the x exchanges.

The following algorithmic sketch shows how to update all the affected $Cands()$, $Parents()$, and $lcand()$ fields when two points p and q exchange positions in the x-order of S. Without loss of generality, assume that $p \prec q$ (p is left of q) before the exchange. Furthermore, assume that p is below q at the instant of exchange (similar pseudo-code applies if p is above q). See Figure 5.

1. If $p \in Parents(q)$, specifically in $Parents_b(q)$, then
   (a) Split off the portion of $Cands(q)$ inside $Dom(p)$ and join it to the top of $Cands(p)$.
   (b) Let u be the point such that $q \in Parents_a(u)$. Delete q from $Parents_a(u)$ and insert it into $Parents_a(p)$.
   (c) Let v be the new bottom point of $Cands(q)$, if any, or else the point such that $q \in Parents_b(v)$. Delete p from $Parents_b(q)$ and insert it into $Parents_b(v)$.
2. Let p' and q' be the points such that $p \in Cands(p')$, and $q \in Cands(q')$. If $p' = q'$, then update $lcand(p')$ starting from p and q in the tree for $Cands(p')$.

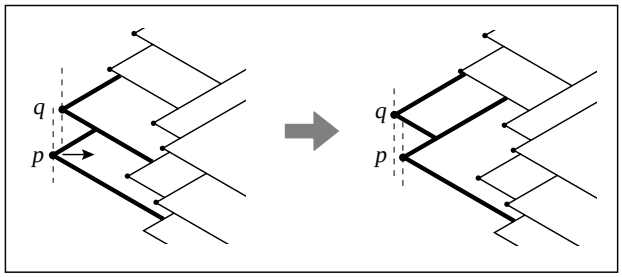

Figure 5: An x event and the change in the $Cands$ sets.

Lemma 3.4 *After the preceding procedure for updating the $Cands()$, $Parents()$, and $lcand()$ fields when two points of S exchange in x-order, the data structure correctly represents the maxima diagram for the current configuration of S, and the $lcand()$ fields are correct.*

The following pseudo-code tells how to update the affected fields when two points p and q exchange positions in the $+60°$-order of S (at the instant of exchange, the line through p and q makes an angle of $-30°$ with the x-axis). Without loss of generality, assume that p is left of q. There are two cases, depending on whether q enters or exits from $Dom(p)$. In this abstract we give only the pseudo-code for the case in which q enters $Dom(p)$. (See Figure 6.) The code for the case in which q exits $Dom(p)$ inverts the action performed in the first case.

1. If $p \in Parents_a(q)$ then

(a) Let v be the point such that $q \in Cands(v)$. Delete q from $Cands(v)$ and insert it into $Cands(p)$.

(b) Let t be the leftmost point in $Parents_b(q)$ that is to the right of p, if any, or else the point such that $q \in Parents_a(t)$. (Recall that x-order in $Parents_b(q)$ is equivalent to the order in which edges hit $Dom(q)$.) Delete p from $Parents_a(q)$ and insert p into $Parents_a(t)$.

(c) Split off the subsequence of $Parents_b(q)$ whose points are to the left of t (and hence left of p) and join it onto the bottom of $Parents_b(p)$.

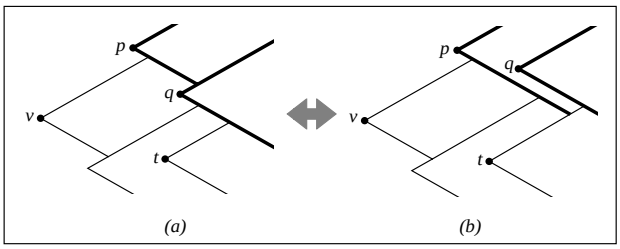

Figure 6: A 60° event. (a→b) q enters $Dom(p)$; (b→a) q exits $Dom(p)$.

Lemma 3.5 *After the preceding procedure for updating the Cands(), Parents(), and lcand() fields when two points of S exchange in the $+60°$-order, the data structure correctly represents the maxima diagram for the current configuration of S, and the lcand() fields are correct.*

The procedure for exchanging two points in the $-60°$-order is symmetric to the one for $+60°$-order exchanges. There are $O(n^2)$ exchanges in each of the three orders.

Each of the update operations needed to restore the auxiliary data structures $Cands()$, $Parents()$, and $lcand()$ takes $O(\log n)$ time: each involves a constant number of standard operations on binary trees.

The final element of our kinetic data structure is a kinetic tournament on the $3n$ distances corresponding to $(p, lcand(p))$ pairs (this adds $3n$ certificates to our KDS). The root of the tournament tree contains the closest pair at any time during the running of the algorithm. Note that when $lcand(p)$ changes, it triggers a discontinuity of the associated distance in the kinetic tournament, but bounds like those in Section 1 apply even in this case.

Theorem 3.6 *The kinetic data structure for the closest pair problem is efficient, compact, and local.*

4 Conclusion and further issues

We have presented a new framework for maintaining attributes (configuration functions) of objects in continuous motion. This framework introduces an on-line, combinatorial approach to changes in the configuration function, avoids a discretization of time, and sets the ground for using sophisticated algorithmic techniques to maintain these configurations in what we call kinetic data structures. We measure the quality of a KDS by its locality and efficiency. By working through three examples, we have demonstrated the generality of the kinetization procedure, which transforms a static algorithm into its kinetic counterpart.

The algorithms described in this paper have recently been implemented by Craig Silverstein (convex hull) and Li Zhang (closest pair) [23, 24]. In both cases, finite precision arithmetic generated errors in the exact sequencing of events, which were due to two main causes. The first was keeping a global clock for event times—as more and more bits are required to record the current time, precision is lost as the simulation proceeds. A possible fix to this problem is to store only time differences between events in the priority queue. The second was the operation of taking square roots to solve the second degree equations that arise in determining event times in the convex hull case (and in the closest pair case when balls bounce against each other). To address these problems, it proved fruitful, in the convex hull case, to resort to the approximate model mentioned in the introduction, i.e., to schedule an event at a date that is a conservative estimate of the actual date at which it will happen, and, from that point, reschedule it at a more precise date given the updated knowledge of the positions of the points (iterative convergence).

In conclusion, we mention a number of issues that need further work:

Although in the analyses of the two examples discussed in this paper (convex hull and closest pair) we have assumed that each point follows a fixed pseudo-algebraic flight plan, it seems that in general it is important to make the number of flight plan changes (globally, or on a per object basis) a parameter of the analysis. This will become necessary, even if our actual objects never change flight plans, whenever we want to compose kinetizations. For example, the separation of the closest pair among continuously moving points changes continuously, even if the actual pairs realizing the distance change from time to time. If this distance itself is to become an input to another kinetic algorithm, its flight plan has to be updated whenever the underlying realizing pair changes. An instance of this phenomenon is already present inside our kinetization of the closest pair algorithm in Section 3.

Experiments on random inputs showed that our kinetic convex hull algorithm has an overhead of internal events that is of the same order as the number of exter-

nal events [23], whereas our kinetic closest pair algorithm always processes $\Theta(n^2)$ internal events [24]. Hence, ideally, the measure of efficiency should not compare the worst case number of internal events to the worst case number of external events, but the worst case ratio of the actual number of internal events to the actual number of external events for any flight plan. It appears much more difficult to develop good algorithms with respect to this measure. Even if an exact analysis is difficult, heuristics that prune unneeded internal events are likely to prove important in practice.

We can view our kinetization process as starting from a proof of correctness of a static configuration function, and then 'animating this proof through time.' Not all proofs are equally good for this use. Our locality requirement favors proofs that have a small number of predicates involving each particular datum. Thus it will generally be advantageous to start with 'shallow proofs'—proofs of small depth—for the static problem, such as one gets, for example, from *parallel* algorithms for solving the static version. Techniques already developed in parallel computational geometry [3], or in parametric searching [2], may prove to be useful.

In a real time system, it is possible that there is not sufficient time to completely process an event before the next event appears. If the kinetic structures are to be used in such a context, it is crucial to be able to maintain partially correct structures, with a mechanism for processing multiple events efficiently and correctly as a batch.

Acknowledgments. We wish to thank Pankaj Agarwal, Rajeev Motwani, G.D. Ramkumar, Craig Silverstein, and Li Zhang for useful discussions. Leonidas Guibas acknowledges support by ARO–MURI grant 5-23542, and NSF grants CCR-9215219 and IRI-9306544.

References

[1] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[2] Pankaj K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

[3] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry*. Prentice-Hall, 1993.

[4] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11:1171–1181, 1985.

[5] Sergei N. Bespamyatnikh. An optimal algorithm for closest pair maintenance. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 152–161, 1995.

[6] Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms (SODA '95)*, pages 263–272, 1995.

[7] O. Devillers, M. Golin, K. Kedem, and S. Schirra. Revenge of the dog: Queries on Voronoi diagrams of moving points. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 122–127, 1994.

[8] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[9] J.-J. Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. *Internat. J. Comput. Geom. Appl.*, 1(1):23–32, 1991.

[10] M. Golin, R. Raman, C. Schwarz, and M. Smid. Randomized data structures for the dynamic closest-pair problem. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 301–310, 1993.

[11] L. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Proc. 17th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, volume 570 of *Lecture Notes in Computer Science*, pages 113–125. Springer-Verlag, 1991.

[12] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[13] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[14] S. Kapoor and M. Smid. New techniques for exact and approximate dynamic closest-point problems. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 165–174, 1994.

[15] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[16] T. Ottmann and D. Wood. Dynamical sets of points. *Comput. Vision Graph. Image Process.*, 27:157–166, 1984.

[17] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[18] Madhav K. Ponamgi, Ming C. Lin, and Dinesh Manocha. Incremental collision detection for polygonal models. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages V7–V8, 1995.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[20] T. Roos. Voronoi diagrams over dynamic scenes. *Discrete Appl. Math.*, 43:243–259, 1993.

[21] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.

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

[23] C. Silverstein. Personal communication. 1996.

[24] L. Zhang. Personal communication. 1996.