Kinetic Data Structures

A State of the Art Report

Leonidas J. Guibas, *Stanford University, Stanford, CA 94305*
e-mail: guibas@cs.stanford.edu

Suppose we are simulating a collection of continuously moving bodies, rigid or deformable, whose instantaneous motion follows known laws. As the simulation proceeds, we are interested in maintaining certain quantities of interest (for example, the separation of the closest pair of objects), or detecting certain discrete events (for example, collisions) which may, in turn, alter the motion laws of the objects. In this paper we present a general framework for addressing such problems and the tools for designing and analyzing relevant algorithms, which we call kinetic data structures. *We discuss kinetic data structures for a variety of fundamental geometric problems, such as the maintenance of convex hulls, Voronoi and Delaunay diagrams, closest pairs, and intersection and visibility problems. We also briefly address the issues that arise in implementing such structures robustly and efficiently. The resulting techniques satisfy three desirable properties: (1) they exploit the continuity of the motion of the objects to gain efficiency, (2) the number of events processed by the algorithms is close to the minimum necessary in the worst case, and (3) any object may change its 'flight plan' at any moment with a low cost update to the simulation data structures. For computer applications dealing with motion in the physical world, kinetic data structures lead to simulation performance unattainable by other means. In addition, they raise fundamentally new combinatorial and algorithmic questions whose study may prove fruitful for other disciplines as well.*

1 Introduction

Motion is ubiquitous in the world around us. Computer disciplines, such as computer graphics, robotics, and vision, that deal with modeling the physical world must in particular deal with the modeling of motion. This may involve estimating motion parameters from sensor data, local and global motion planning and control, collision checking, physics-based simulation, character animation, rendering, etc. Unlike the modeling of shape, which has been well studied in the above disciplines over the past three decades, the analysis and modeling of motion in a cohesive fashion is still in its infancy. A key difficulty is that in modeling motion we rarely have all the motion data available at once. Motion takes place over the time dimension, and the motion parameters of a system of moving objects, whether real or virtual, can change drastically because of events and interactions between the objects that are hard to predict far into the future. Useful motion models must incorporate this basic on-line character of the problem.

In the work presented here, we will be interested in the simulation of a complex system of multiple moving objects and the efficient maintenance of various attributes of this system as the motion evolves. For example, we may be interested in maintaining (the separation of) the closest pair of objects, because we want to detect collisions between them. We may want to maintain a binary space partition (BSP) of the space containing the objects, in order to quickly obtain visibility information for rendering. Or we may want to maintain a minimum spanning tree (MST) connecting the moving objects, as a way of linking them through a wireless communications network. Closest pairs, BSPs, and MSTs are all concepts whose fundamental importance is well-known to Computational Geometry, and on which extensive research has been done. The novelty in our setting is that the defining objects are not static but in *continuous motion*. We mean 'continuous' both in the sense of non-stop, as well as in the mathematical sense — objects do not jump from one part of space to another. We propose to study a new class of algorithmic techniques, which we call *kinetic data structures* (or KDSs for short), that exploit this conti-

nuity or coherence of the motion for maintaining these geometric structures of interest more efficiently than any known alternatives. We will refer to the attribute being maintained as the *configuration function* of the system (or CF for short). Note that each of the above CFs has a combinatorial description (e.g., the graph structure of the MST) that changes only at discrete times, when certain events occur (but the separation of the closest pair, or the total cost of the MST are themselves continuous functions of time).

Now, in principle, the continuous motion of each object can be approximated, after a discrete sampling of time, by deleting and reinserting it in a new position at each time step. But this incremental updating method, which is widely used in practice, benefits from continuity only indirectly and can be used only when the number of moving objects is small, because of its inherent inefficiency. In particular, the trouble with any fixed-rate sampling of the evolving system is that it either oversamples the system (wasting resources), or undersamples it (possibly missing critical periods) — since the events of interest to the configuration function typically occur in highly irregular patterns over time. The aim of our new technique is to take advantage of the coherence present in continuous motions so as to process a minimal number of combinatorial events, yet still maintain the CF correctly. In this respect, the way of analyzing our data structures is related to the *dynamic computational geometry* framework introduced by Atallah [10] in order to study the number of combinatorially distinct configurations of a given kind (e.g., convex hull or closest pair) that arise during known simple motions of the geometric objects. However, unlike Atallah's setting in which motions are predetermined, our data structures do not require us to know the full motion of the objects in advance. Thus they are much better suited to real-world situations in which objects can change their motion on-line because of interactions with each other, external impulses, etc.

Though our emphasis will be on exploiting the continuous motion of the present objects, we must also deal with new objects entering and old objects exiting the simulation. Such discrete changes can be handled by using dynamic data structures which are well-developed in Computational Geometry — these must then be combined with our kinetic structures. The need for handling discrete changes arises as well in *composing* kinetic data structures with each other. For example, a kinetic client of a kinetic convex hull algorithm for moving points will see a combinatorial change to the convex hull as a discrete update (a new point enters the hull, or an old point leaves).

The remaining subsections of the paper are as follows. Section 2 introduces the key idea on which our development of good KDSs is based: that of *animating proofs through time*. The motion model and qualities of good KDSs are discussed in sections 3 and 4 respectively. The following four sections survey the currently known results and discuss several ideas towards the future development of KDSs for important configuration functions in geometry, including classical problems in arrangements, intersections, visibility, etc., all not yet studied in the kinetic setting. Section 9 returns to the dynamic computational geometry framework and discusses new approaches for bounds on CF combinatorial changes under various probabilistic models. Section 10 discusses the issues that arise in implementing efficient schedulers for the event queues that arise in kinetic simulations and Section 11 explores various additional topics related to the practical implementation of KDSs. Finally Section 12 concludes by discussing the theoretical and applied significance of this work.

Although the work presented here is focussed primarily on rigid motions in a geometric context, it is worth remarking the the ideas of kinetic data structures extend to any context in which a continuous attribute evolves through a sequence of discrete events. Thus the conceptual framework we will develop is equally applicable to problems involving deforming objects, or even to completely non-geometric problems, such as the maintenance of shortest paths in a graph with continuously varying edge costs.

2 Animating Proofs Through Time

We will maintain a configuration function by performing an event-driven simulation of the motion. Such methods are based on an event queue which stores events, each with an associated event time. The simulation always proceeds to the next event in the queue; the event is removed from the queue and processed. This in turn may cause additional events to be scheduled or old events to be descheduled. And then this cycle is repeated all over — a situation not unlike that of sweep algorithms in Computational Geometry.

The critical events for us are those that change the combinatorial description of the configuration function. For example, if we are maintaining the closest pair of n moving points in the plane, the instant when the current closest pair AB is replaced by CD as the closest pair, constitutes an event that we must know about and have scheduled in the priority queue. Since discovering and scheduling these events is not easy, we will often find it advantageous to consider a somewhat larger set of events that is easier to maintain. We call the events which change the configuration function *external*, in order to distinguish them from the remaining events we process, which we call *internal*. Any correct method for maintaining the CF must process all external events; the internal ones are there for the convenience of the particular kinetic data structure we are developing.

How are we to find a superset of the set of the external events that is not too large and at the same time is easy to maintain? The key insight of our approach is that we can do so by maintaining through time an evolving *proof of correctness* of the value of the configuration function of interest. Such a proof will consist of a number of elementary conditions on the moving data, which we call *certificates*, that altogether imply (prove) the correctness of the current value of the CF[1]. Each certificate has an earliest failure time, and this failure is scheduled as an event in the event queue. Clearly, as long as no certificate fails, the combinatorial description of the CF cannot change. When a certificate fails, it is the job of the kinetic data structure to repair and update this proof. It may be that the CF has not changed (AB is still the closest pair), but the proof needs to be updated (internal event). Or it may be that both the CF and its proof need to change at such an event (external event).

Since the motion of the objects is continuous, it will be possible to repair a single certificate failure in a well-chosen proof with a relatively modest cost. If this failure is an external event, then the proof will guide us on how to update the configuration function as well. Finding proofs (i.e., certificate sets) that evolve gracefully in time as certificate failures happen is exactly the art of designing good kinetic data structures. Given a configuration function, we can generate a first cut at a proof as follows. Conceptually speaking, we stop the motion, run our favorite static algorithm for computing the value of the CF, and then collect all the tests the algorithm performed together with their outcomes. If this static algorithm is correct, then the outcomes of all these tests is a set of certificates that proves the correctness of the value of the CF. Most proofs generated this way will not animate well, or they will require considerable massaging before they yield good kinetic structures. We will present several examples of this process in later sections of this paper.

Since certificates will generally correspond to tests performed by a static algorithm for the CF, they invariably involve only a small and constant number of the moving objects. For reasons that will become clearer in Sections 3 and 4, throughout this work we will aim to only use certificates that involve a constant number of objects each, as well as to minimize the number of certificates that any one moving object is involved in.

To summarize then, a kinetic data structure maintains a CF by updating over time a proof of correctness of the CF. We saw earlier the difficulties with any scheme for maintaining the CF by doing a fixed-rate sampling of the evolving system. By using the idea of 'proof animation' we are able to do a more intelligent, uneven sampling that is better adapted to the CF, so as to avoid wasteful computation when the CF does not change, while doing the necessary update at the exact times when it does.

3 Motion Models

What information about the moving objects does a kinetic data structure need in order to do its work? The interface happens in the calculation of the certificate failure times. In the simplest setting, each moving object follows a publicly posted *flight plan* specifying its short-term motion. If we look at the evolving system over a period of time when these flight plans or motion laws do not change, then we can calculate for each currently valid certificate its failure time. In a typical case, the flight plans will be polynomial or rational trajectories, and the certificate itself will be a simple algebraic inequality, so the certificate failure time will be the next largest real root of some low-degree polynomial.

Of course the flight plan of an object can change. A flight plan *update* can occur because of interactions

[1] In this paper we will often use the word 'proof' to refer to such a certificate set.

between the 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. But an object can also change its motion law because of its own free will. In order for the simulation to proceed correctly, whenever a flight plan update happens, the KDS must be informed about it. Events that change the flight plans of objects (e.g., collisions) must themselves be scheduled in the event queue before they happen. Thus a KDS for the MST of a set of moving and elastically bouncing balls may need to also implement a closest pair/collision detection KDS as a way to know all the flight plan updates. Or, as we already remarked, when composing KDSs, events that change the output of the first structure can generate flight plan updates for the other.

When a flight plan update for an object happens, all the certificates that involve that object must have their failure times recalculated and their positions in the event queue updated. This makes it desirable to keep the number of certificates involving any particular object small.

Although the simple motion model presented so far may be adequate for certain idealized virtual reality simulations, it is clearly insufficient when tracking moving objects in the real world. It can then be the case that the flight plans only give us partial information about the object motions, or that there are no posted flight plans at all. Instead the KDS has to try to predict the current motion of each object by extrapolating from its past motion. With this incomplete knowledge, the KDS clearly cannot calculate exact failure times for certificates. The best it can do is to calculate for each certificate a 'last time' till which the KDS can be sure that the certificate is still valid, given the partial flight plans and perhaps some *a priori* bounds on the velocity and other motion parameters of each object. Events can be scheduled for those times and the certificate then be re-examined using any updated position/motion information that may have become available in the meantime.

In the real-world context, this raises the issue of having the kinetic algorithm use sensing so as to acquire better motion information about the objects in order to resolve the status of certificates. This then becomes akin to the model of 'data in motion' introduced by Kahan [33]. What is the best way for a kinetic algorithm to use sensing in order to proceed with the simulation? This raises several research issues that have not been adequately explored yet.

Several other motion models incorporating uncertainty and limiting the ability of the KDS to examine the evolving system are worth discussing, but in order not to delay further the presentation of concrete problems, we will omit this topic. In the remaining sections we will assume that we have posted flight plans with full motion information and that all flight plan updates are events scheduled by our KDS or others in the event queue. We will also assume that each motion plan has fixed complexity independent of the number of moving objects[2]; this implies that the cost of calculating a certificate failure time is $O(1)$.

4 Analysis of Kinetic Data Structures

So far we have concentrated on the issue of correctness for a kinetic data structure. The challenges were how not to miss any external events affecting the CF, and how to stay informed of all the flight plan updates. But how shall we evaluate kinetic structures? Storage and time are clearly important, but the on-line nature of the events raises some additional issues akin to those of on-line algorithms. Criteria for measuring the performance of kinetic data structures is the topic of this Section.

Though the exact nature of the flight plans is not relevant to correctness issues, it clearly affects performance. When analyzing kinetic structures we will generally assume that all trajectories followed by the moving objects are *pseudo-algebraic* splines. By this we mean that each trajectory consists of a discrete set of arcs, and the transitions between the arcs correspond to flight plan updates. The arcs themselves must be what we call pseudo-algebraic with respect to the KDS. By this we mean that all certificates used by the KDS can switch from TRUE to FALSE at most a bounded number of times, for any motions following the given arcs. This is a condition akin to the Davenport-Schinzel condition [41] commonly used to

[2]This assumption excludes certain kinds of simulations, such as classical n-body simulations [29], from our simplest framework.

limit the complexity of curve interactions in Computational Geometry. In particular, algebraic arcs of bounded degree are always pseudo-algebraic. On occasion, we will make much stronger assumptions, such as in the usual scenario of linear, constant velocity, motions.

The performance bounds we give depend on n, the number of rigidly moving objects or object parts. Clearly the number of flight plan updates also affects the complexity of the simulation. Exactly how to incorporate that into our framework is a topic for further research. For the KDS analyses in the following sections we will assume that there are no flight plan updates, except in specific situations where we are composing kinetic structures. We will call a quantity 'small' if it is of the order of $O(\text{polylog}(n))$, or even $O(n^\epsilon)$ for some arbitrarily small $\epsilon > 0$.

We propose four major criteria for evaluating these structures.

responsiveness Most obviously, a KDS is good if the cost of processing a certificate failure is small. The responsiveness of a KDS is the worst-case amount of time needed to update its proof after a certificate failure. This may require discovering if the value of the CF has changed and what the new value is, as well as updating the certificate set by removing old certificates that are no longer part of the proof, and adding new certificates that are part of the new proof. All these certificate changes need to be reflected in the event queue as well. We call a KDS *responsive* if the the worst-case amount of time needed for such a proof update is small (in the technical sense just defined).

efficiency A second key performance measure for a KDS is the worst-case number number of events processed. 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 (technically, we require that the ratio of total events to external events is small). This is reasonable, as the number of external events is a lower bound on the cost of any algorithm for maintaining the desired configuration function. A KDS meeting this condition will be called *efficient*.

Note that in this definition we are comparing events over two different motions: the ones maximizing the number of events processed by the KDS, and the ones maximizing the number of external events. In a more refined setting, we call this measure *weak efficiency*. A weakly efficient KDS only guarantees that it never processes many more events than the worst-case number of external events over all allowed motions. We define *strong efficiency* to mean that the worst-case ratio of total events processed to external events (taken over all allowed motions) is small — this is akin to the *competitive ratio* of on-line algorithms [21] and is perhaps a more satisfactory notion of efficiency. Unfortunately so far we have found only few strongly efficient KDSs, and those under highly restrictive motion assumptions. We need to expand our repertory of strongly efficient structures.

locality We already remarked a number of times that it is important to keep low the maximum number of certificates in which any one object appears, in order to allow efficient flight plan updates. The locality of a KDS is the maximum number of certificates in which any one object can ever appear. If that number is small we call the KDS *local*. The existence of local KDSs is an intriguing question for several CFs.

compactness The size of the KDS is the maximum number of certificates ever present in a proof — it also reflects the size of the event queue that needs to be maintained. We call a KDS *compact* if the maximum number of certificates ever present in a proof is of the order of n times a small (in the technical sense) quantity (i.e., if it is nearly linear).

Note that locality implies compactness, but responsiveness and efficiency are unrelated. There exist efficient but unresponsive structures, as well as inefficient but responsive ones[3].

[3] In our early work on KDSs we have focussed on problems for which compact and local KDSs exist, as the next few sections show. We expect that the development of KDSs for problems where the minimum proof size is superlinear to be considerably more challenging and to require revisions in our locality desiderata.

5 An Example KDS

To make the issues above more concrete, and since we do not have the space to present in detail kinetic solutions to the more substantial problems discussed in the sequel, 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 meet, 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 set 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 [32]). 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 based on knowing in advance the full motions of the points: a flight plan update will force a recomputation from scratch of all future maxima. In [12] we show that various other simple solutions also suffer from drawbacks as kinetic structures. Maintaining a sorted list of the points is easy and yields a structure which is responsive, local, and compact, but unfortunately not efficient. Maintaining a heap is only slightly more complicated, and this yields a responsive, efficient, local, and compact structure. But in this case the proof of efficiency is not easy [13], and to have such a simple KDS be so hard to analyze is discouraging.

Let us also consider the following fourth solution to the kinetic maximum maintenance problem, which we call a *kinetic tournament*. The idea is to use a simple divide-and-conquer strategy. The algorithm partitions the points into two approximately equal-sized groups (arbitrarily), and recursively maintains the maximum of each group. A final comparison at the top level yields the global winner. This tournament structure generates $\Theta(n)$ certificates, each asserting an inequality between the leaders of two subtournaments. When one of these comparisons changes, the new winner has to be propagated up the tournament tree to its proper level. Clearly the update cost is $O(\log n)$, and locality is $O(\log n)$ as well.

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 winner 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 $O(n)$ and it follows that this upper envelope computation has a worst case cost satisfying of $C(n) = O(n \log n)$. The number of kinetic tournament events (re-schedulings, 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 a responsive, efficient, compact, and local KDS.

This maximum problem is one of the rare ones for which we know of a strongly efficient KDS, but in a special case only. We present this as our fifth and final solution. If point y_i is moving as $t_i(t) = \alpha_i t + \beta_i$, then let us map it to the static vector (α_i, β_i) in the parameter plane with axes α and β. Clearly the maximum y_i at time t is the vector among these whose dot product with $(t, 1)$ is maximum. Thus it is enough to precompute the convex hull of the points in the (α, β)-plane corresponding to the tips of these vectors, and then simply track the extremum around the convex hull with a rotating tangent whose slope is $(-1, t)$. The cost of detecting each maximum change is constant. This structure is kinetic, because if flight plan update happens, it can be accommodated by updating the convex after deleting the vector corresponding to the old mo-

tion parameters and inserting the vector corresponding to the new ones. Such updates to the convex hull can be handled efficiently, in time $O(\log^2 n)$ [38].

6 Extent Problems

The first class of kinetic problems we will investigate deals with the *extent* of a set of moving objects. By extent we mean various representations of how spread out the objects are in space. We include under extent classical geometric configuration functions such as the convex hull, diameter, width, various types of bounding boxes, minimum enclosing sphere, etc. In the context of moving bodies these configurations are important for knowing if the objects have entered any forbidden regions (e.g., aircraft flying in forbidden airspace), clipping (e.g, speeding up rendering by ignoring objects outside the view port), detecting various exceptional conditions (e.g., some object gets so far away that it is out of communication range), etc.

Convex hulls and upper envelopes

We have developed a responsive, efficient (in the weak sense), local, and compact algorithm for points moving in the plane with algebraic trajectories of bounded degree [12]. Our algorithm works in a divide-and-conquer fashion, like the kinetic tournament of Section 5. It is actually easier to explain the algorithm in the dual setting, where the problem of maintaining the upper convex hull of n moving points in the plane dualizes to the problem of maintaining the upper envelope of n moving lines. Again, we partition the lines into two roughly equal groups which we call red and blue, and then recursively maintain the upper envelope of each group. The goal of the root node of the recursion is to maintain the purple upper envelope of all the lines, given the recursive maintenance of the red and blue envelopes. A certificate structure for this problem can be derived by considering how we can merge the red and blue envelopes into the purple envelope in a static setting. This can be done by a standard sweep-line algorithm and requires two types of tests: x-tests in order to decide the x-ordering of vertices of the red and blue envelopes, and y-tests that compare whether a red or blue vertex is above or below its blue or red *contender* edge (see [12] for details). Altogether these x- and y-certificates, from all levels of the recursion, form a proof of correctness of the current convex hull.

Unfortunately this set of certificates is not local: as many as $\Omega(n)$ blue vertices might be above the same red contender edge (or vice versa). If that red edge undergoes a flight plan update, a large number of certificates will need to have their failure times recomputed. In [12] we show how to modify this certificate set and make it local by adding an additional type of test, a *slope* or s-certificate between pairs of lines. It is then quite straightforward to check that when one of these x-, y-, or s-certificate fails at a node of the recursion tree, it is possible to update the local upper envelope and corresponding certificate set in $O(1)$ time, and the same holds for all higher levels of the tree to which this change needs to be propagated. From these considerations it easily follows that the structure is local, compact, and responsive. Figure 1 shows a simple example of this process. How about efficiency? Even with straight lines motions, we can show that the convex hull can change combinatorially $\Omega(n^2)$ times [5]. Because of our algebraic trajectory assumption, it is clear that the number of events corresponding to s-certificate failures is $O(n^2)$, but the corresponding counts for y- and x-certificates are $O(n^3)$ and $O(n^4)$ respectively. However, by considering the surfaces swept by the lines over time and invoking recent theorems of Combinatorial Geometry about the complexity of upper envelopes [31], as well as the overlay of envelopes [8], we can prove a near-quadratic bound on the number of the y- and x-events as well. Thus the algorithm is efficient ($O(n^{2+\epsilon})$ events in the worst-case), but the proof of this fact requires substantial machinery.

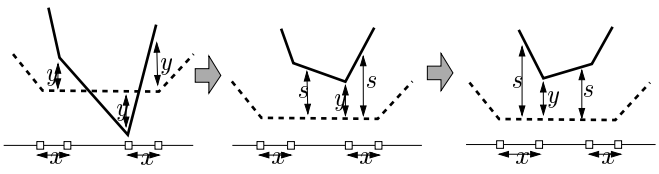

Figure 1: *A set of consecutive kinetic events.*

Efficient kinetic data structures for convex hulls and lower envelopes in higher dimensions $d > 2$ are not yet fully developed. We expect this to pose challenges, even for $d = 3$. To apply the divide-and-conquer strategy presented above to 3-d upper envelopes of moving planes we will have to maintain in the xy-plane the overlay of subdivisions corresponding to the projections of the red and blue upper subenvelopes. In

Section 8 we will see how to maintain a planar subdivision induced by moving segments in the plane in the context of binary space partitions (BSPs). However, those ideas might not work as well in the current context, as the moving elements of the subdivision are themselves derived moving objects (e.g., the projections of lines that are the intersections of the primary moving planes). Also, the overlay subdivision can have $\Theta(n^2)$ complexity, even though each of the blue, red, and purple subdivisions has of course linear complexity. Thus we must find a way to maintain (the relevant parts of) the overlay subdivision implicitly, unless we are willing to consider non-compact structures with $\Theta(n^2)$ certificates. In addition, the trick we used to ensure locality for y-certificates in 2-d extends in the 3-d case only in part, unfortunately. It does not work when we assert that a convex envelope lies below a triangle — in this case all the rim vertices of the envelope seem to require independent z-certificates. Thus both sublinear locality and compactness remain open issues.

We expect progress on these fundamental structures to enable progress on other higher-dimensional kinetic problems as well.

Diameter, width, minimum spanning circle, etc.

Once we can maintain the convex hull of moving points in 2-d, we can also solve a number of other related extent problems. In [5] we show how to maintain the diameter, width, and various flavors of bounding boxes (minimum area, minimum perimeter) with the same overall kinetic performance as the convex hull. These applications show the power of composing kinetic data structures: once we have the convex hull we can maintain antipodal pairs of vertices of the convex hull and compare their distances or separation using a kinetic tournament. We also show that these kinetic algorithms are efficient, as each of the respective configuration functions can undergo $\Omega(n^2)$ changes even for points moving linearly in the plane. It will be quite interesting to consider the same problems in higher dimensions, either based on convex hull techniques, or by other more direct methods, as was done for diameter and width in the static case [18]. The kinetic maintenance of collections of bounding boxes, such as those used in OBB-trees [28] is also very interesting.

Not all 2-d extent problems can be solved this easily. Consider the problem of maintaining the minimum spanning circle (MSC) of a set of n moving points in the plane. It is possible to prove in various ways that the maximum number of times the MSC can change combinatorially is nearly cubic (under the pseudo-algebraicity assumption), but we do not know if this bound is tight. In a space-time diagram the MSC provides a certain kind of envelope of the point trajectories. This makes us hope that some of the vertex charging schemes for lower envelopes, that were recently so successfully used to bound the complexity of lower envelopes of algebraic surfaces [40], might be applicable to this case as well. The key will be to find a way to relate vertices on the envelope (i.e., time instances when the MSC is defined by four cocircular points) to vertices inside the envelope, but not too deeply.

In terms of a kinetic data structure for the MSC, we can use the observation that the MSC is the smallest circumcircle of any of the triangles in the furthest point Delaunay triangulation of the point set (which is always a triangulation of the convex hull). If we can maintain the convex hull and the furthest point Delaunay triangulation of the point set, then we can derive an elegant kinetic structure for the MSC as follows.

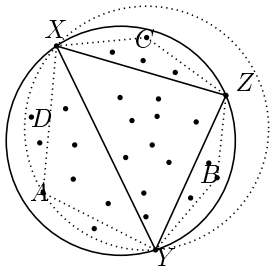

Figure 2: *Maintaining the MSC.*

Consider the dual tree of the furthest point Delaunay triangulation, and root it at the triangle whose circumcircle is the MSC. To be concrete, let this triangle be XYZ, and let the three other triangles that are the children of the root have as their third vertex the points A, B, and C respectively. We claim that as long as the convex hull and the Delaunay triangulation do not change, the MSC also cannot change. The only way for the MSC to change combinatorially is to have one of the points A, B, or C exit the current MSC, and this is a Delaunay event. A point such as D in the figure cannot escape from the MSC without first escaping from

the circumcircle of the Delaunay triangle AXY — and this will also change the Delaunay triangulation. This observation creates a tree of *InCircle* Delaunay certificates of linear size, which together with the certificates needed for the convex hull, proves the correctness of the MSC. When one of the certificates fails, this MSC proof tree can be updated by simple tree rotations. Unfortunately, we do not know if this KDS is efficient. A near-cubic upper bound on the number of total events processed can be proved — the dominant term is the number of Delaunay triangulation changes, for which only an upper bound of the form $O(n^2\lambda_s(n))$ is known (see below).

Range searching

It is worth noticing that the diameter, width, MSC, etc. are all extent measures defined by a small and fixed number of the moving points. This is also the case for the canonical pieces used in various *range searching* geometric structures, including Willard partition trees [24], conjugation trees [24], and many of the newer structures for simplex range searching based on cuttings and shallow cuttings [36]. The efficient maintenance of range searching structures under continuous motion of the objects defining them is another area that needs to be developed.

7 Proximity and Intersection Problems

Proximity and intersection are fundamental modalities in describing events that affect the evolution of a system of objects in motion. Collisions between objects, i.e. transitions from non-intersection to intersection, invariably affect the flight plans of the objects involved. Proximity to other objects will often cause a moving object to change its flight in order to get closer or further away from one of the moving objects, and so on. For complex objects it is often advantageous to approximate them with simpler bounding volumes and then check for intersection between these bounding volumes first. Thus we may want to track which pairs of bounding volumes of objects intersect, as such pairs may need to be checked for collision by a more refined algorithm. In this section we present some preliminary results and research plans for problems in this area.

Voronoi diagrams and Delaunay triangulations

The central role of Voronoi diagrams and Delaunay triangulations is well-established in Computational Geometry. From the kinetic point of view, there is both good and bad news regarding these diagrams. Let us focus on the Delaunay triangulation — these diagrams are equivalent and the events that change one combinatorially are exactly the events that change the other as well. Maintaining the Delaunay triangulation of points moving in a low-dimensional Euclidean space is surprisingly straightforward. We describe the situation for $d = 2$; analogous statements hold in higher dimensions. The key insight that helps is an old theorem of Delaunay himself: a triangulation which is 'locally Delaunay' is globally Delaunay [22]. By locally Delaunay we mean that every edge of the triangulation passes the *InCircle* test — in other words the circumcircle of the triangle on one side of the edge does not contain the third vertex of the triangle on the other side. This set of local conditions gives us a compact set of certificates for the Delaunay triangulation; their failure times become the events to be scheduled in the event queue. When a failure event happens, a 'flip operation' [24] replaces the bad edge by a good edge, and all is well. Thus a KDS for Delaunay is immediate and is clearly responsive, strongly efficient, and compact structure, though of course it is not local (a vertex can have high degree).

That was the good news. The bad news is that despite work on this problem over several decades, we still do not know a tight bound on the number of combinatorial changes in the Delaunay triangulation when the points move pseudo-algebraically, or even linearly. Currently the best known upper bound is $O(n^d\lambda_s(n))$ [30] for some constant s, and the best known lower bound is $\Omega(n^d)$. This gap makes it hard to judge the worst-case efficiency of algorithms that utilize Delaunay or Voronoi KDSs. There is some hope that we can apply recent techniques used to prove a subcubic bound on the complexity of the union of n unit cylinders in $\mathcal{R}^3$ [1] to improve the upper bound for $d = 2$.

Closest pair problems

Tracking the closest pair among a set of moving objects gives us a way to detect collisions among them: clearly the next colliding pair must be the closest pair among all the objects for a small time interval before the collision. Note also that the closest pair of n moving objects can change at most a near-quadratic number of times — we simply plot the separation of each pair as a function of time and take the lower envelope

of these functions. Though Voronoi and Delaunay diagrams contain within them closest-pair information (for example, a kinetic tournament on the Delaunay edges will easily maintain the closest pair), the near-cubic bound on events for these diagrams suggests that other approaches may be significantly more efficient.

We have been able to develop two different kinetic data structures for maintaining the closest pair among n moving points. One structure is for $d = 2$ [12] (though extensions to higher dimensions should be possible), while the other works for all dimensions [15]. Both structures are based on a common insight: to find the closest pair is is sufficient to examine a linear number of point pairs defined by partitioning the space around each point into a fixed number of congruent cones having the point as their apex. These cones have a central axis and each point keeps track of some of its nearest neighbors in each cone. This of course is an old idea going back to A. Yao [16], but in our case the nearest neighbors are defined not in terms of Euclidean distance, but in terms of distance from the apex to the projections of the points on the cone axis — maintaining these neighbors is a much easier 1-d problem. A packing lemma implies that if we have fine enough cones and select enough nearest neighbors in each cone, then we cannot miss the true Euclidean closet pair. For example, in the case $d = 2$, our first approach uses three 60° cones and one neighbor per cone, while the second approach use four 90° cones and three neighbors per cone. It turns out that point membership in these cones, as well as the 1-d nearest neighbors used by these algorithms, can all be obtained from keeping track of the sorted order in the projections of the moving points along a fixed (but of size roughly 2^d) set of directions in the space. Thus these methods clearly process a quadratic set of events in the worst case. Keeping track of which points lie in which cones is accomplished by using a kinetic multidimensional range search tree [15]. With some further insights that we do have not the space to discuss here, these ideas lead to closest pair KDSs which are responsive, efficient, local, and compact.

Several variations on the closest pair problem are interesting and open, even for $d = 2$. First, we do not have a good KDS for the bichromatic closest pair problem [2]. Second, though our closest pair methods extend to the problem of keeping track of the closest among n moving equal radius balls, they do not work when the radii are widely different. The reason is that the key packing lemma we use fails in that case. Of course a Delaunay triangulation corresponding to a weighted Voronoi diagram can still be used in this case [17]. But the issue of whether there is a KDS with a near-quadratic number of events for this problem remains open. Furthermore, it is not clear if it is possible to have a local structure at all. A large ball with many small balls near its boundary suggests that sublinear locality may not be possible — though we have been unable to prove any hard lower bound.

Finally note that our closest pair algorithms have the undesirable property that they process events corresponding to order reversals on the projections of the points along a number of axes. Many such events might be irrelevant, as the points involved are actually very far away in space. It seems intuitively desirable to develop kinetic structures where events always correspond to interactions between points that are sufficiently close. We can accomplish this by partitioning space into bins and processing events only among points in each bin, but at some point the cost of transferring points from bin to bin also becomes significant [35].

Minimum spanning trees

In a number of applications the moving objects are observers and it is important to keep communication among them for effective global motion scheduling, so as to accomplish a task (e.g., explore a building to find a target). One way of doing so is to keep a set of point-to-point wireless communication links between pairs of observers. Among all possible such communication networks, the minimum spanning tree (MST), i.e. the set of links connecting all the observers together with the minimum total Euclidean length, has many desirable properties. How can we efficiently maintain the MST of n points moving around in space?

We do not yet have a good kinetic solution for this problem. We have structures which are responsive, compact, and local, but not necessarily efficient [15]. Our structures maintain either the MST under a polyhedral distance metric, or an approximate Euclidean MST (guaranteed to be a $1 + \epsilon$ approximation of the true Euclidean MST). These structures make use of the kinetic multidimensional range search tree mentioned earlier. Our algorithms exploit again the idea of maintaining a sparse graph among the points which is guar-

anteed to contain all the MST edges, and then keeping track of the MST of this graph. Note that the MST can change either because the underlying sparse graph changes combinatorially, or because the weight relations among the edges of the graph change over time. We have recently [7] been able to obtain a subquadratic algorithm ($O(n^{11/6}\text{polylog}(n))$ currently, but probably better) for maintaining the MST of a (fixed) planar graph with edge weights varying in a pseudoline fashion (pseudo-algebraic of degree 1). This new algorithm is based on recursive graph partitioning using cycle separators and leads to a responsive, local, and compact KDS. The analysis uses tools similar to those exploited by Dey in his recent breakthrough result on the complexity of a level in an arrangement of lines (and pseudolines) [23]. As in the case of Voronoi/Delaunay, we are somewhat hampered by the lack of sharp bounds on the combinatorial number of changes to the MST in both the geometric and graph settings, even under linear point or weight motions (for these cases the currently best known upper bounds are $O(n^3)$ for the Euclidean MST of n linearly moving points [34], and $O(kn^{1/3})$ for the MST of a graph with k vertices and n edges whose weights change linearly, as follows from the technique of [23]).

Incidences and many faces problems

Not every proximity problem deals with points or balls. In fact mixed proximity problems between points and lines or curves raise many interesting kinetic issues of their own. For example, consider the following problem. In the plane we are given n moving points and n moving lines. We wish to track the closest point-line pair (or perhaps just report all point line incidences during the motion). Clearly there can be a near-quadratic number of combinatorial changes to the configuration function. But a local and compact structure may not be possible. The standard many faces construction [25], as well as Erickson's lower bounds on the static version of this problem [27], suggest that a natural certificate set size for this problem is $\Theta(n^{4/3})$. By using ϵ-net techniques we can give a kinetic data structure that processes a near-quadratic number of events and has a proof size of $O(n^{5/3})$ [4]. It would be quite interesting to do better.

Several variations are also worth investigating. These include maintaining the face or faces of the arrangement of the lines containing one or several points, maintaining the zone of another line, and many other kinetic versions of classical arrangement problems. Of course, if we are willing to maintain the full arrangement of the moving lines, these problems become rather easy. The arrangement itself can be maintained by using a vertical decomposition of it — the certificates will simply assert that each trapezoid is well formed. The proof size will be $\Theta(n^2)$ and the worst-case locality $\Theta(n)$ (this is optimal for this problem). An easy counting argument can be used to show that the number of events processed by this kinetic structure will be near-cubic (in space-time we sum the squares of the complexities of the cells of that 3-d arrangement).

Maintaining intersecting pairs

We remarked earlier on the use of bounding volumes for efficient collision detection among moving objects, a method well-established in graphics and robotics [11]. Suppose for concreteness that we are in a 2-d situation and that these bounding volumes are disks. The bounding disks of two objects may intersect, even though the objects themselves do not. One way to approach the problem of collision detection is to maintain the intersecting pairs of disks. Pairs of objects whose disks intersect have to be checked for collision by a more sophisticated algorithm. When a new pair of disks starts intersecting, the corresponding pair of objects is added to the list of those to be thus checked. Similarly, when a pair of disks stops intersecting, the object pair is removed.

This is a more challenging problem than maintaining the closest pair of disks, as in the latter we can assume that all the disks are disjoint. Note that, since for our purposes it is sufficient to just report the changes in intersection status, we do not need to maintain an explicit list of intersecting pairs. This is fortunate, because there can be $\Theta(n^2)$ intersecting pairs among the n disks. One approach for discovering the changes in the intersecting pairs is to maintain a *hitting set* of points for the intersections, i.e., a set of point such that every intersecting pair contains one of these points. The arguments of [42] give an upper bound of $O(n^{5/3})$ and a lower bound of $O(n^{4/3})$ on the size of a minimum hitting set. We can also assume, without loss of generality, that the hitting set points lie in what are called *maximal* cells of the arrangement of the disks (these are cells covered by more disks than any of their neighbors — they are always intersections of a number of the

circles and thus convex). A hitting set can be maintained by (1) tracking the evolution and eventual death of these maximal cells and (2) discovering the creation of new maximal cells, which always arise as a new intersection between two disks. The death of a maximal cell implies that a pair of disks stops intersecting, and that a maximal cell has fissioned into two. The set of intersections that before was hit by a single point will now require two points to be hit (unless one or both of the new maximal cells is a single disk contained in no other, in which case it need not be hit). The tracking of a single maximal cell is the maintenance of the intersection a set of disks and this is akin to the intersection of half-planes or the convex hull problem we have already discussed in Section 6. However, the discovery of new maximal cells will require some new insights for a good KDS.

A different view of this problem that may be helpful can be obtained by lifting the disks with center (x, y) and radius r to the point in 3-d $(x, y, x^2+y^2-r^2)$. If we imagine that the paraboloid of revolution $z = x^2 + y^2$is an opaque surface, then the condition that two disks intersect is exactly that the corresponding 3-d points are mutually visible, despite the paraboloid obstruction. Thus as these points fly around, we are interested in detecting visibility changes among the point pairs. We also note that this intersection maintenance question makes sense for other bounding volumes, including axis-aligned boxes — for which it may be easier.

8 Visibility Problems

The use of visibility information is essential in rendering animations involving very large large models (e.g., architectural walkthroughs) as well as in planning motions related to visibility tasks (model or environment acquisition by one or more mobile robots with cameras, guarding or art gallery type problems, pursuit-evasion algorithms, etc.). In the context of kinetic data structures, these applications pose problems such as the maintenance of the portions of the environment currently visible from one or multiple moving observers, the maintenance of the mutually visible pairs of observers, etc.

Binary space partitions

Binary space partitions (or BSPs) were one of the earlier structures used in computer graphics to obtain visibility information. The application of BSPs to visibility determination is based on the principle that an environment can be described as a collection of clusters, which can be separated from each other in such a way that a visibility ordering can be defined between the clusters. The partition of the clusters is done by means of cutting the underlying space into two halfspaces by a plane. The crucial observation is that clusters that lie in the same halfspace as a given viewpoint can obscure, but never be obscured by, the clusters on the other side — which then provides a visibility order that can be used to correctly render a scene. A recursive application of the partition operation yields a binary space partition tree from which a visibility ordering can be determined by performing a depth-first search in the tree. There have been a few attempts to update BSPs when the objects defining them move [37, 45, 20]. All these prior efforts, however, reduce to deleting moving objects from their earlier positions and reinserting them in their current positions after some time interval has elapsed. Such approaches suffer from all the problems discussed in Section 1.

We have recently been able to obtain kinetic data structures for disjoint moving segments in the plane [6], and disjoint moving triangles in space [3]. These methods are based on defining a BSP by cuts along the given objects or parallel to a particular axis; the cuts are generated according to a random ordering of the objects. The resulting BSP has expected size $O(n \log n)$ and depth $O(\log n)$ in 2-d, and expected size $O(n \log^2 n + k)$ and depth $O(\log n)$ in 3-d, where k denotes the number of intersections between pairs of edges in the projections of the triangles on the xy-plane. As this latter bound suggests, our 3-d structure is based on a kinetic 2-d structure for maintaining the BSP of set of moving and possibly intersecting segments. These BSPs behave very well from the kinetic point of view: we can detect when the current BSP structure becomes invalid due to object motion and update the tree at a cost of $O(\log n)$ per event for the 2-d BSP, and $O(\log^2 n)$ for the 3-d BSP. In the easier to explain 2-d algorithm the events correspond to times when certain critical trapezoids, called *transient* trapezoids [6], collapse by having their two parallel sides coincide. The event counts are $O(n^2)$ and $O(n^2 \lambda_s(n))$ in the 2-d and 3-d cases respectively. These structures are responsive and strongly efficient. Their size is bounded by the BSP size, but they are not local.

Many interesting issues remain open. To obtain the above bounds we need to assume that there is no correlation between the motions of the objects and the fixed random ordering used by the algorithm. An adversary could, of course, design motions aimed at eventually making these BSPs bad, but then the algorithm could respond by evolving over time the ordering through random transpositions and making the corresponding updates to the tree. An even more fundamental open question has to do with BSP optimality, a notoriously difficult problem. Our BSPs above are strongly efficient, but could it not be that there is some other BSP that undergoes many fewer events for the same motions? Can we prove that all 2-d BSPs of a certain type must undergo $\Omega(n^2)$ transitions for some pseudo-algebraic motion of the n disjoint line segments in the plane?

Visibility set

For very large environments, even the fastest hardware depth-buffers cannot render a scene at interactive frame rates. Thus techniques have been developed for computing efficiently a relatively small superset of the potentially visible polygons, called the *visibility set*, and then throwing only those to the depth buffer [43]. A number of approaches have been proposed for maintaining this visibility set as the observer moves [19, 9] in a 2-d environment. Accomodating both observer and environmental motion is clearly an area where further research needs to be done.

9 Probabilistic Event Bounds

Given the rather high worst-case event counts we saw in earlier sections, it makes sense to ask about the number of changes these configuration functions undergo when we consider average, or 'typical' object motions. Defining typical object motions is of course very application dependent. Nevertheless, we have initiated an investigation of how configuration functions change for very simple random point motions in the plane.

Assume n points choose independently a random origin and destination in the unit square and then, in the time interval $[0, 1]$, they move with constant velocity along a straight line segment from their origin to their destination. This will be our model of a random motion in the plane. In [46] it is shown that the convex hull of the points changes $\Theta(\log^2 n)$ times in expectation; the corresponding bounds for closest pair and Delaunay triangulation are $\Theta(n)$ and $\Theta(n^{3/2})$ respectively. The methods used to obtain these results mimic those that have been successfully applied to the same questions in the static setting [39]. For instance, in the case of the convex hull, we compute the conditional probability that three points are on the convex hull given that they are collinear along a line ℓ at time t. We then compute the joint probability density on the product space of lines and times, and use linearity of expectation to complete the calculation. Such probabilistic analyses give simple answers to questions that are much harder to tackle in the worst case. Some preliminary calculations suggest that, in d dimensions and for a similar probabilistic model, the Delaunay triangulation changes $\Theta(n^{1+1/d})$ times and that the closest pair changes $\Theta(n^{2/d})$ times in expectation.

The expected number of events processed by a KDS can also be computed in this model. For example, the convex hull structure we have presented in Section 6 processes $\Theta(n)$ events in expectation, while our 2-d closest point structures process $\Theta(n^2)$ events even in expectation, as the latter have to maintain sorted orderings along one or more axes. There is a simple modification to one of the closest pair algorithms [14] that avoids these sorts and experimentally seems to process roughly $\Theta(n^{3/2})$ events — but we have not been able to prove this formally. For Delaunay, of course, there are no internal events, so the trivial KDS for Delaunay also processes $\Theta(n^{3/2})$ events in expectation.

The case of the minimum spanning tree seems to be especially difficult to analyze in this random setting, and the above methods do not apply. An upper bound of $O(n^{5/2})$ (to be contrasted with the best known worst-case upper bound of $O(n^3 2^{\alpha(n)})$ [34]) follows from the fact that the MST is a subgraph of the Delaunay triangulation and some standard batching argument. A major subgoal in solving this problem is to answer the following static question: given n points chosen independently at random according to a prescribed distribution, what is the expected size of the (graph) diameter of their MST?

10 Kinetic Event Scheduling

In the efficient implementation of kinetic data structures, a key problem is the efficient scheduling and de-scheduling of events (in our implementations this accounts for 70% to 90% of the total cost). The twist

that makes this different from a standard priority queue maintenance problem is that the calculation of the event times is part of our cost. Since our proofs evolve rapidly over time, the accurate calculation of an event time may be wasteful if the corresponding certificate gets removed from the proof before its failure time. We cannot, of course, know the future, but this situation suggests that we compute less accurately event times that are far into the future, and refine our estimates as they get closer and closer to the present time.

Motivated by these issues, we have been studying the following problem. Suppose we maintain a set $\mathcal{S}$ of low degree polynomials $\{f_1(t), f_2(t), \ldots, f_k(t)\}$ — these are associated with the certificates currently active in a KDS. For example, assuming we have points with polynomial motions, if the event we worry about is that 'points A, B, and C become collinear', or equivalently 'the triangle ABC changes sign', then the associated polynomial $f(t)$ will simply be the signed area of the triangle ABC at time t. There is the notion of the current time t_0, and we are interested in finding quickly time t_1, the smallest real root of any of the f_i which is larger than t_0. The time t_1 is the time of the next kinetic event. Then we advance time by setting $t_0 \leftarrow t_1$ and let our KDS update its certificate set and thus the set of active polynomials. If the KDS is responsive, the changes to $\mathcal{S}$ will be relatively small. We then repeat this process all over to continue the simulation.

In order to maintain the next largest real root of this slowly evolving set of polynomials we can of course calculate all the real roots of each polynomial to the required precision for the simulation and insert all these roots into a standard priority queue. But, as remarked above, this may spend root-finding cycles for events that will never happen. We have been investigating an approach based on Sturm sequences in which for each active polynomial $f(t)$ we isolate its roots into a set of intervals whose size increases exponentially as t goes to infinity. This exponentially increasing set of intervals contains all the real roots of f — of course we allow intervals that contain many roots. The leftmost interval of f represents f in a priority queue where the polynomials compete to determine the smallest real root. Both the process of resolving comparisons between such intervals, as well as certain large steps forward in time, can cause us to refine these interval sequences to obtain tighter bounds on the roots of f. We are still developing the theoretical basis of this approach and a preliminary implementation. Note that, assuming polynomial or rational motions, an interval-based approach may allow a simulation using only rational arithmetic.

11 Implementation Issues

In order to validate the use of kinetic data structures in practice, we have implemented a number of the previously discussed structures and compared them to alternative methods for maintaining the same configuration function. This implementation has raised a number of additional research issues that need to be addressed.

Kinetic convex hulls in practice

In addition to the kinetic method presented in Section 6, we implemented the straightforward Delaunay triangulation KDS of Section 7 and a 'brute-force' algorithm. We implemented the Delaunay KDS because it is used in several kinetic problems, but in particular it contains the convex hull as a substructure. For the brute-force structure, we simply calculate the time at which each point will hit (or leave) the convex hull, assuming its current motion remains unchanged. When a point on the hull has a flight plan update, we must recalculate the event times of all the other points[4]. Furthermore, whenever a point enters or leaves the convex hull, all events must be rescheduled.

In order to remain independent of specific implementation details, the cost of a KDS was taken to be a sum over the set of polynomial equations solved, each weighted by an amount corresponding to the difficulty of solving equations of that degree. We ran these three methods with n ranging up to 10000 points on random motions like those in Section 9[5]. We let the simulation run until the convex hull stabilized and there was no more events in the event queue (Figure 3). As Figure 3 shows, the kinetic structure becomes superior to the brute force structure for n less than 100, and it is always much superior to the Delaunay KDS as the latter is hampered by having to solve fourth degree equations (as opposed to second degree for the convex hull KDS). Three snapshots from our implementation are shown in Figure 4. On the bottom is a

[4] When a non-hull point updates its flight plan, we need only recalculate its own event time.

[5] Alternative distributions, such as the uniform unit disk and the Gaussian, gave qualitatively similar results.

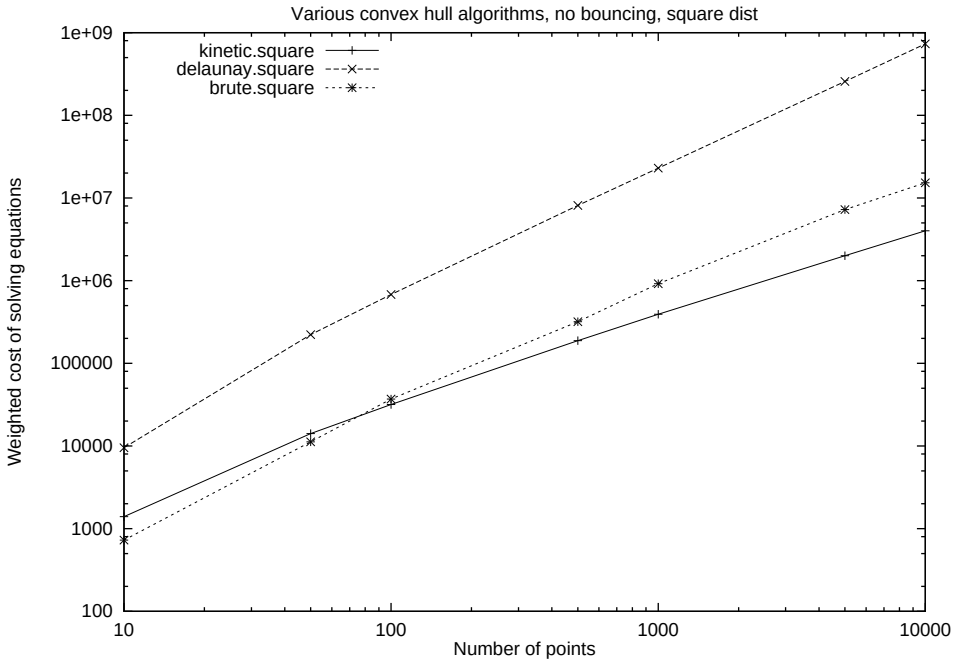

Figure 3: *Cost (weighted sum of the number of equations solved) of the three different methods that maintain the* CONVEX HULL*, when the points have linear motions.*

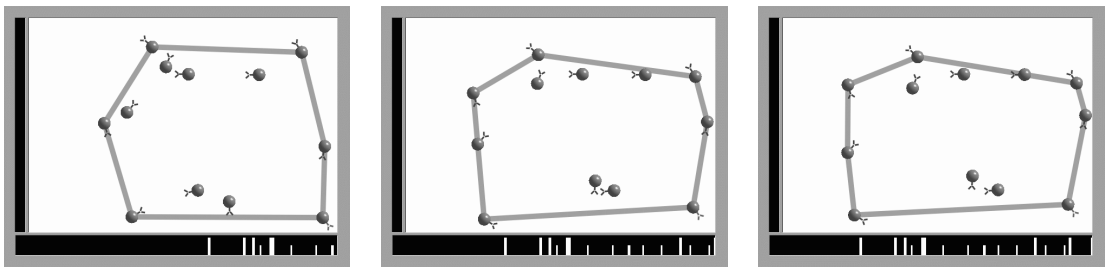

Figure 4: *Three frames from a kinetic convex hull animation.*

time window showing event times. The long bars represent external events, while the short bars show internal events. A public demo of this software is available at `www-graphics.stanford.edu/~jbasch/demokin`.

Several additional experiments are described in [14] and in a forthcoming fuller version of that paper based on current work. Our primary finding is that, with a careful implementation, kinetic data structures do not pose major difficulties in practice and perform well on

several natural motion distributions. For a large number of points, alternative data structures suffer from non-locality or expensive root-finding operations.

Open research issues

The implementation of kinetic structures raises numerous questions that we intend to address.

- We have already discussed the issue of avoiding the precise calculation of event times until they are needed. There is also concern about processing events out of of order, or multiple times, because of numerical errors. With some preliminary efforts in this direction, we were able to run reliable simulations involving 50,000 points with about 400,000 events for the convex hull KDS.
- It is not clear that processing all events in the ideal time order is necessary for a correct simulation. If two events happen at almost the same time, but they involve sets of objects that are far apart from each other and do not otherwise interact according to the CF, it may be acceptable to process these events in either order. Of course we do want to guarantee that all events affecting any particular object happen in the correct sequence. This then would give us a 'local clock' for each object and we would not require global synchronization between all these clocks — an approach reminiscent of a topological sweep [26], but now in the time domain.
- What if too many events happen in a short time interval and we do not have the computational resources to process them all in that interval? Can we batch the processing of nearby events for efficiency? Given a description of a class of motions for the objects (e.g., bounds on velocities, etc.), can we prove that with a given amount of computational resources we will never get into this bottleneck?

12 Conclusions and Impact of the Work

We have presented the notion of kinetic data structures, based on the idea of animating proofs through time. Kinetic structures raise numerous new and interesting combinatorial and algorithmic questions, while at the same time offering the promise of significant impact in all areas of computer science dealing with simulating motion or continuous change in the physical world.

KDSs implement time sweeps, and as such have a lot in common with geometric space sweep methods (event queues, event scheduling, etc.). The novel aspect of kinetic structures is that in the time dimension the future is not known *a priori* — it is decided as we go along the time axis. Such structures are based on proofs or certificate sets that are maintained under continuous motion of the participating objects. They raise questions about what kinds of proofs are possible to certify the correctness of various geometric configuration functions — lower bounds on proof size and locality become interesting problems. At this point in the development of the area, we do not understand deeply which proofs animate well in the sense that they evolve gracefully across certificate failures. There is some evidence that proofs derived from *parallel* algorithms (especially of the kinds used in *parametric searching* [44]) animate better than those coming from sequential algorithms. The intuition is that proofs from parallel algorithms are 'shallower' (in the sense of smaller maximal deduction depth, not in the common mathematical sense) — their parts are more independent and they have relative short deduction chains. These attributes help make proof updates less costly and give good bounds on locality.

Kinetic data structures also motivate the development of better techniques for estimating event counts (CF changes) in both the worst and average cases. In addition, the current analyses of KDSs need to be refined to include the number of flight plan updates in the complexity. The interaction between kinetic and dynamic structures needs to be better developed as well — we saw that many problems require both. Finally, more careful analyses are needed for the cases where only some objects are moving, or large groups of objects are moving rigidly together.

Acknowledgements.

The author wishes to thank Pankaj Agarwal, Julien Basch, John Hershberger, Micha Sharir, and Li Zhang for many useful discussions. This work was funded by ARO–MURI grant DAAH04-96-1-007, and NSF grants CCR-9623851 and IRI-9619625.

References

[1] P. Agarwal and M. Sharir. The complexity of the

union of unit cylinders in space. Manuscript. To appear (1997).

[2] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[3] P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, page to appear, 1998.

[4] P. K. Agarwal and L. J. Guibas. Kinetic problems on arrangements. Manuscript. To appear (1998).

[5] P. K. Agarwal, L. J. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving set of points. In *5-th workshop on algorithms & data structures (WADS)*, 1997.

[6] P. K. Agarwal, L. J. Guibas, T. Murali, and J. Vitter. Cylindrical static and kinetic binary space partitions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 39–48, 1997.

[7] P. K. Agarwal, L. J. Guibas, and M. Rauch. Kinetic minimum spanning trees. Manuscript. To appear (1998).

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

[9] B. Aronov and M. Teichmann. Online maintenance of visibility and shortest-path information. Manuscript. To appear (1997).

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

[11] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, May 1995.

[12] J. Basch, L. J. Guibas, and J. Hershberger. Data structure for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, Jan. 1997.

[13] J. Basch, L. J. Guibas, and G. Ramkumar. Sweeping lines and line segments with a heap. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 469–471, 1997.

[14] J. Basch, L. J. Guibas, C. D. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[15] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[16] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest-point problems. *ACM Trans. Math. Softw.*, 6:563–580, 1980.

[17] J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, Paris, 1995.

[18] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric searching. *Discrete Comput. Geom.*, 10:183–196, 1993.

[19] D. Z. Chen and O. Daescu. Maintaining visibility of a polygon with a moving point of view. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 240–245, 1996.

[20] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation*. Ph.D. thesis, Queen Mary and Westfield College, University of London, 1996.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[22] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[23] T. Dey. Improved bounds for k-sets and k-th levels. Manuscript. To appear (1997).

[24] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[25] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity of many cells in arrangements of planes and related problems. *Discrete Comput. Geom.*, 5:197–216, 1990.

[26] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989. Corrigendum in 42 (1991), 249–251.

[27] J. Erickson. New lower bounds for Hopcroft's problem. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 127–137, 1995.

[28] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[29] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[30] 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 Comput. Sci.*, pages 113–125. Springer-Verlag, 1991.

[31] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to vis-

ibility in terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.

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

[33] S. Kahan. A model for data in motion. In *Proc. 23th Annu. ACM Sympos. Theory Comput.*, pages 267–277, 1991.

[34] N. Katoh, T. Tokuyama, and K. Iwano. On minimum and maximum spanning trees of linearly moving points. *Discrete Comput. Geom.*, 13:161–176, 1995.

[35] D. J. Kim, L. J. Guibas, and S. Y. Shin. Fast collision detection among multiple moving spheres. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 373–375, 1997.

[36] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[37] B. F. Naylor. Interactive solid geometry via partitioning trees. In *Proceedings of Graphics Interface '92*, pages 11–18, May 1992.

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

[39] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gerwählten Punkten I. *Z. Wahrsch. Verw. Gebiete*, 2:75–84, 1963.

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

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

[42] H. Tamaki and T. Tokuyama. How to cut pseudo-parabolas into segments. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 230–237, 1995.

[43] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.

[44] S. Toledo. Extremal polygon containment problems and other issues in parametric searching. M.s. thesis, Dept. Comput. Sci., Tel Aviv Univ., Tel Aviv, 1991.

[45] E. Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, Sept. 1990.

[46] L. Zhang, H. Devarajan, and J. B. P. Indyk. Probabilistic analysis for combinatorial functions of moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 442–444, 1997.