A Practical Evaluation of Kinetic Data Structures

Julien Basch Leonidas J. Guibas * Craig D. Silverstein Li Zhang

Computer Science Department
Stanford University
Stanford, CA 94305

1 Introduction

In many applications of computational geometry to modeling objects and processes in the physical world, the participating objects are in a state of continuous change. Motion is the most ubiquitous kind of continuous transformation but others, such as shape deformation, are also possible. In a recent paper [BGH97], Basch, Guibas, and Hershberger proposed the framework of *kinetic data structures* (KDSs) as a way to maintain desirable information about the state of a geometric system in continuous motion or change. They gave examples of kinetic data structures for the maximum of a set of (changing) numbers, and for the convex hull and closest pair of a set of (moving) points in the plane. There are two notable and novel aspects of the kinetic data structure framework. First, while extensive work has been done on dynamic data structures in computational geometry [CT92], this is all focussed on handling insertions/deletions of objects rather than continuous change. Kinetic data structures by contrast gain their efficiency by exploiting the continuity or coherence in the way the system state changes. Second, unlike Atallah's dynamic computational geometry framework [Ata85], which was introduced to estimate the maximum number of combinatorial changes in a geometric configuration under predetermined motions in a certain class, the KDS framework is fully *on-line* and allows each object to change its motion at will, due to interactions with other moving objects, the environment, etc.

The purpose of this paper is to study and validate the use of the kinetic data structures proposed in [BGH97] in practice. The implementation and experimental evaluation of such structures brings to light several important issues which were not addressed in the original paper. For instance, KDSs implement the continuous maintenance of the function of interest through a discrete-event simulation. The calculation of the event times poses numerical difficulties which raise questions about the integrity of the structure if nearby events should happen out of order, or if the time of an event changes when recalculated. Also, the algorithmic measures for evaluating the quality of KDSs in [BGH97] are all worst-case measures, which may not reflect behavior under ordinary motions which occur in practice. It may well be that simple data structures do better in such situations.

In order to study these issues, we have compared the KDSs of [BGH97] with simpler variants under various common data/motion distributions, each meant to generate a different amount of

* Support was provided in part by National Science Foundation grant CCR–9623851 and by US Army MURI grant 5–23542–A.

change in the combinatorial state of the various data structures. These variants include a KDS for Delaunay triangulation [GMR91]. While these alternative data structures have inferior worst-case performance, there are no results stating how they compare when using less artificial point distributions. From analyzing the data obtained, we conclude that the KDSs of [BGH97], with some simple modifications proposed in this paper, perform remarkably well. Compared to the alternatives, they perform their task using less complicated root-finding primitives and process fewer events in almost all cases studied. This advantage is particularly pronounced when frequent motion changes are involved.

The paper is organized as follows. In Section 2 we review the kinetic setting and the kinetic data structures described in [BGH97]. In Section 3, we describe a number of alternative data structures for moving points. In section 4, we describe the specific way in which floating point errors affect kinetic data structures and propose some remedies. We present and discuss experimental results in Section 5. Finally, in Section 6 we propose further experiments to elucidate the nature and hidden beauty of kinetic data structures.

2 Kinetic Data Structures

2.1 Terms and Definitions

Given a set of continuously moving points in the plane, we wish to maintain some *configuration function* of interest, dependent on the positions of the points. This function could be, for example, the minimum separation of a pair of the points. Even though this separation changes continuously, it can be derived from a combinatorial function F—describing the identity of the pair of points of minimum separation—that changes at discrete times only. A kinetic data structure maintains the combinatorial function F through a list of *certificates* whose validity at a particular instant in time implies the correctness of F. These certificates are typically low degree algebraic inequalities involving a few of the moving points, such as "s is inside the circumcircle of Δpqr," or "Δpqr has positive orientation."

In the KDS framework each point is assumed to follow a particular *flight plan.* Typically we will assume that these motions are algebraic. Under this constraint, for a given certificate c we can, by solving for the root of some equation, calculate the next future time $t(c)$ at which c becomes false. All these "certificate failure times," or *events* for short, are kept in an event queue associated with the combinatorial function F. Since F cannot change between events, we can perform a dynamic simulation of the system by jumping from event to event. Processing an event involves recomputing a new certificate list—since at least one certificate is now false—that re-validates (the same or a modified) F at the current time. If F changes at the time of an event, the event is called *external.* Otherwise the event is called *internal.* For a KDS to be of good quality, the following criteria should be met:

- the certificate list does not change too much when an event occurs;
- the overhead of internal events with respect to external events is reasonable;
- the KDS itself is of reasonable size, typically linear or slightly superlinear; and
- each point is involved in only polylogarithmically many certificates.

To elaborate on the second condition, we say a KDS is *efficient* if, over a given class of allowed motions, the ratio of the worst case number of total (internal plus external) events to the worst case number of external events is small (as a function of the number of points involved). More precisely, let us denote by $C_F(S)$ the number of combinatorial changes over time of a combinatorial function F defined over a geometric configuration S. We define $C_F(k,n) = \max_S C_F(S)$, where S describes all possible motions of n points in the plane describing trajectories which are algebraic of degree at most k. For a kinetic data structure K, let us similarly define $C_K(S)$ to be the number of events processed by K, and let $C_K(k,n) = \max_S C_K(S)$ over the same set of configurations S. Efficiency, as defined above, means that the the worst case ratio $C_K(k,n)/C_F(k,n)$ is polylogarithmic in n, or at most $O(n^\epsilon)$. Note that in this definition of efficiency, as originally proposed in [BGH97], we are comparing two worst-case scenarios, even though these may arise for very different configurations of moving points.

The last of the four conditions above, called the *locality* condition, is important when points change their motion law, something we call a *flight plan update*. This is common in typical simulations, where for instance points may bounce off walls or each other. When a point changes motion, all certificates dependent on that point much be updated. The locality condition ensures the efficiency of this operation.

2.2 Three Kinetic Data Structures

In this section we present the kinetic data structures developed in [BGH97] and implemented in this paper.

Maximum: Given n points moving on a vertical line, we wish to maintain the identity of the maximum point. This can be done with a *kinetic tournament* as follows: Divide the set of points into two (approximately) equal subsets. Compute the maximum of each subset recursively, and set the overall maximum to be the larger of the two maxima. The certificate list for the maximum can be directly taken to be the list of $\Theta(n)$ tests performed during this computation. This structure is local, as any single point is involved in $O(\log n)$ certificates. The processing of an event requires recomputing the maximum involved in that certificate and possibly all of its parents in the recursion, yielding $O(\log n)$ certificates changes in total. The overall number of events corresponds to the amount of work done by a standard off-line, divide-and-conquer algorithm to compute the upper envelope of the trajectories of the points. This quantity is, in the worst case, only logarithmically more than the number of changes to the maximum. Hence this structure is efficient.

Convex Hull: For the 2-dimensional convex hull, a similar divide-and-conquer technique is effective. Divide the points arbitrarily into two sets and determine the convex hull of each set. We can merge these two convex hulls using a linear number of certificates as follows: Let the two sub-hulls be denoted as "blue" and "red." Each certificate is a triangle composed of one red vertex (called the "tip") and one blue edge (the "base"), or vice versa. Furthermore, each certificate triangle has the property that the slopes of the three edges are ordered v_{in}, e, v_{out}, where v_{in} and v_{out} are the edges going into and out of the tip, respectively (edges in the sub-hulls are oriented counter-clockwise).

There are two classes of events: collinearity events and parallel-edge events. Events in the first class happen when the three points on a certificate become collinear, indicating the certificate triangle is changing orientation. This causes at most two edges of a convex hull to change; the exact change depends on whether the tip is entering the hull or leaving it, and on whether it is going through the base or to one side of it. The parallel-edge events happen when a certificate ceases to be valid because its edge slopes are no longer ordered properly. In this case, one edge of the certificate will be parallel to an edge of an adjacent certificate, forming a rhombus with one diagonal. A *flip* event takes place in this case: we flip to the other diagonal of the rhombus. Two certificates are destroyed and two new ones created, but no hull changes. Note that these events can happen at any (or each) level of the recursive hull construction, but only collinearity events at the top level are external.

The KDS described above is not local, since a single point can be a member of many certificates, as described in [BGH97]. It is possible to make the KDS local by using a slightly modified list of certificates, but we have chosen to implement only the simpler version described above.

Closest Pair: The closest pair KDS is based on the construction of a *cone structure* for the point set, as follows: denote by x, u, d the horizontal, $30°$ and $-30°$ directions. For each point p, in turn from right to left, erect two walls (called the u-wall and d-wall) extending to the right in the d and u directions, until they hit already existing walls (Figure 3.4). The points visible from p between these two walls are called the *candidates* of p. The leftmost of these candidates is called the *left candidate* of p. In order to maintain this structure, we need to maintain for each cone three sets with efficient union and split capabilities:

- $Cands(p)$ is a balanced binary tree of all candidates of p, sorted by vertical coordinate. Each node ν of this tree is augmented with $\min(\nu)$, the identity of the leftmost point of the subtree rooted at that node,
- $Parents_d(p)$ is also a balanced binary tree which consists of all the points whose u-wall is stopped by the d-wall of p, sorted by increasing projection along d.
- $Parents_u(p)$ is the same for points whose d-wall is stopped by the u-wall of p, sorted by increasing projection along u.

If this construction is repeated twice with rotations of $60°$ and $-60°$, the closest pair is amongst the (at most) $3n$ candidate pairs selected in this process, and the winning candidate can be found with a tournament.

It can be shown that the cone structure depends only on the order between the projections of the points along our three directions. Hence, to kinetize this structure, it is enough to maintain these orderings (with $O(n)$ certificates), as well as a kinetic tournament on all candidate pairs. There are about quadratically many internal events in the worst case, and each event can be processed by rescheduling $O(\log n)$ events.

3 Alternative Data Structures for Moving Points

The data structures described in [BGH97] are not the only one possible to handle the stated problems for moving points. Below we describe some alternative data structures that, though they lack theoretical guarantees, may prove to be practical for a wide variety of realistic situations.

3.1 Judging Kinetic Data Structures in Practice

As we mentioned in Section 1, the *efficiency* of a KDS is the ratio of the worst case number of internal events to the worst case of the number of external events. A better way to measure the quality of a KDS would be to do it in the spirit of competitive analysis [ST85], i.e., to attempt to bound the ratio $C_K(S)/C_F(S)$, as S ranges over all allowed motions of the n points. Under this improved definition of efficiency, only a few extremely simple problems have known good algorithms [Gui]. No kinetic data structures for convex hull or closest pair has been proved efficient with respect to this measure—this worst-case competitive ratio can be extremely high. Yet, we might hope that with ordinary motions the situation is much better. A major motivation of this paper is the desire to (empirically) judge kinetic data structures using this new measure for ordinary motions. We measure and compare the actual efficiency of several KDSs by measuring the number of internal and external events for each of them when run on points moving with constant speed (possibly different for each point). The initial positions and speed are taken at random from a number of common but diverse distributions (see Section 5).

Along with efficiency, *locality* is an important measure of the quality of a KDS. However, worst-case guarantees of locality may not matter if even the most obvious data structures are local in practice. In order to judge locality under a number of different distributions, our test suite includes simulations where the points bounce elastically against walls. Since each bounce requires rescheduling every event associated with the bouncing point, non-local structures will suffer in this framework.

A final measure of the quality of a KDS is the amount of CPU time it takes to run a given test. This can be modeled by the number of equations the KDS solves in the course of computing event times. The overall running time is dominated by these operations. A further distinction, between solving equations of different degrees, allows for a more accurate model of running time.

3.2 Delaunay Triangulation

For points moving in the plane, the simplest combinatorial function to kinetize is probably the Delaunay triangulation, as it is *self-certifying*. That is, its set of edges, together with the cocircularity test that certifies their "local Delaunayhood", is a correct certificate structure [GMR91]. Because of this property, there is a simple kinetic data structure for the Delaunay triangulation which has no internal events. Furthermore, a kinetic Delaunay triangulation is particularly useful since it is possible to obtain both the convex hull and the closest pair from it (in one case directly; in the other via a linear size tournament tree on the Delaunay edges). The exact worst-case number of changes to the Delaunay triangulation is currently not known even for linear motions[1], so this structure cannot be proved efficient for the maintenance of the convex hull or closest pair. Moreover, it is not local.

If the motion of each point can be described by a polynomial spline whose segments have degree k, the event times for the for the Delaunay certificates will involve finding roots of polynomials of degree $4k$. Besides taking longer to solve than the linear or quadratic tests of the structures of [BGH97], these Delaunay polynomials suffer more from floating point errors (see Section 4).

[1]Though it is somewhere between quadratic and cubic.

(Iu) $x(p) < x(Rightmost(Parents_u(p)))$
(Id) $x(p) < x(Rightmost(Parents_d(p)))$
(IId) $d(p) < d(Leftmost(Cands(p)))$
(IIu) $u(p) < u(Rightmost(Cands(p)))$
(IIId) $d(p) < d(Leftmost(Parents_d(p)))$
(IIIu) $u(p) < u(Leftmost(Parents_u(p)))$
(IV) For each node $\nu \in Cands(p)$, certificates that $\min(\nu)$ is the leftmost amongst $\min(leftchild(\nu))$, $\min(rightchild(\nu))$, and the point at ν.

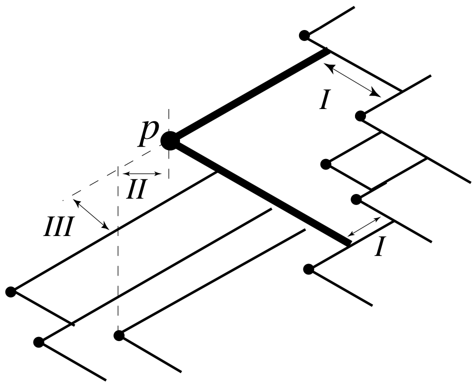


Table 1: The list of certificates associated with the cone of a point p. Certificates of type IV are not shown in the figure. With this alternate list of certificates, the change in d ordering between some points hidden by the cone of p need not be processed.

3.3 Brute Force Algorithm for Convex Hull

There is a simple, brute force data structure for maintaining a convex hull. Assume the flight plans of all points are fixed, and under this assumption calculate when each point will hit (or leave) the convex hull. When a point on the hull has a flight plan update, we must recalculate the event time of every point. (When a non-hull point updates its flight plan, we need only recalculate a single event.) Furthermore, when a point enters or leaves the convex hull, all events must be rescheduled.[2] Because a change in a point's motion may falsify a certificate, this data structure is not a KDS in the traditional sense. Furthermore, because one flight plan update causes us to reschedule n events, this structure is not at all local. Nevertheless, its simplicity makes it an appealing data structure, particularly if the flight plans are fixed or the number of points is small.

3.4 Modified Closest Pair KDS

The closest pair KDS proposed in [BGH97] maintains more structure than is necessary. In particular, it keeps an exact linear ordering of the x-projections of the points in order to certify the validity of the cone structure, and events generated by such certificates do not always modify the cone structure (Figure 3.4). If we replace the exact ordering by an appropriate partial ordering, we obtain a data structure that has the same number of events in the worst case but many fewer events for reasonable point distributions. The crucial observation is that the points of the *Cands* tree are not only ordered by vertical coordinate, but also by increasing projection along d and decreasing projection along u. Likewise, the $Parents_d$ and $Parents_u$ trees are also ordered by increasing horizontal coordinate.

With these observations, we design a set of certificates as shown in Table 3.4. It is routine to check that these, taken over all points, define a certificate list for the cone structure. The closest pair is certified by three of these certificate lists (one for each direction of cone structure), plus those from the kinetic tournament on the distances between points and their left candidate.

It can also be shown that only a few certificates need to change when an event happens, and the

[2]Obvious-seeming optimizations for this case, such as updating only those events that involve a changed hull edge, do not work.

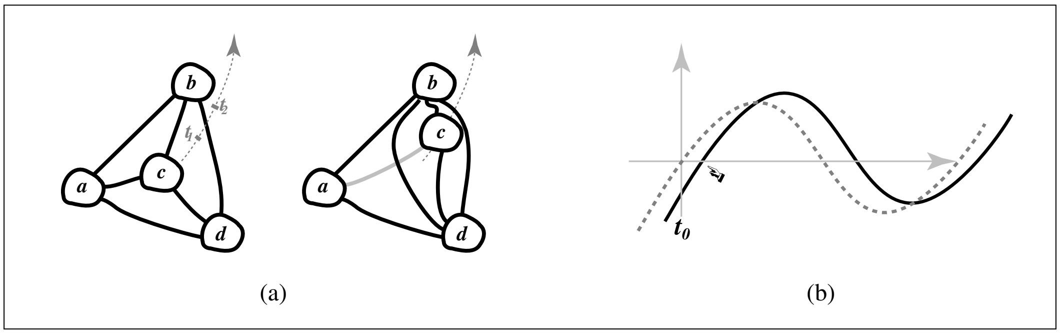


Figure 1: Problems caused by floating point errors. (a) In this Delaunay triangulation, precision errors cause us to flip edge (ab) flip at time t_1, while (bcd) are scheduled to become collinear at a later time t_2. The result of this out-of-order scheduling is a corrupted structure. (b)The polynomial corresponding to a cocircularity event has a root now, and we need to reschedule the next cocircularity event of these four points. Because of precision errors, we might think now is later.

locality remains $O(\log n)$. In the worst case, the number of events is still $\Omega(n^2)$, as the closest pair can change that many times, but chances are the event count will be better in many situations.

4 Numerical issues

Along with degeneracy, arithmetic precision errors are the bane of the geometric algorithms implementor. We discuss here the precision issues specifically related to the kinetic setting: the effect of floating point errors on scheduling, and the techniques available to minimize the potential problems. For definiteness, we limit our attention to the Delaunay KDS which, since it involves polynomials of degree 4 (for linear motions of the points), is most susceptible to precision difficulties. The same issues arise in the KDS for the convex hull, which involves equations of degree 2 and, to a lesser degree, in the KDS for the closest pair (which also involves some equations of degree 2, though mostly equations of degree 1).

To focus on the specific problems of a KDS, we make the reasonable assumption that the value of a polynomial at a given point (time) can be computed exactly, but the roots of a polynomial are computed with a small error. In this setting there are two problems we must address: on the one hand, events can get out of order (Figure 1(a)) and, on the other hand, an event that really happens in the future can appear to have already happened or vice versa (Figure 1(b)). In our implementations, we address the second problem directly and design tests to detect the first problem.

It happens that the second problem is extremely common. Indeed, assume that at time $t = t_0$ we are processing the event "$(abcd)$ are cocircular". Once we have processed this event, we need to schedule another event for the same four points, because they may become cocircular again. The associated polynomial $P(t)$ has a root at $t = t_0$ (for the just-handled event). Because of numerical error, however, this root might be believed to be at some $t_\epsilon > t_0$, and the event that was just processed will be improperly rescheduled. A first fix to this problem is to consider a root only if it is more than some predefined parameter ϵ into the future, but this leads to possibly missing other important events involving other points. Instead, we make use of the fact that we know t_0 is a root: when rescheduling the points *abcd* of a just-handled event, we ignore the root closest to t_0. This is superior to dividing the polynomial by $(t - t_0)$ because it avoids additional precision errors that would be introduced by the synthetic division and we are not interested in multiple roots.

When an event is processed at t, it is necessary to change a few certificates and hence deschedule and reschedule a few events. In such circumstances, both descheduled and scheduled events must have a positive value at t. If such an event has a negative value at t, this indicates that it has been scheduled out of order with the currently processed event. We introduced tests to stop the simulation if this was to happen. We did not observe any out of order event detectable in this way in all the simulations that were performed. However, it should be noted that some out of order scheduling could go undetected. We believe that out of order events can happen in certain cases, and work needs to be done to handle them properly.

A final problem with machine precision occurs only during extremely long simulations with bouncing. As time increases, less precision is available to distinguish between two close-in-time events, since the large times involved require more and more bits to represent. To avoid this problem, we can imagine storing only the time difference between neighbor elements in the event queue heap, rather than their absolute times.

5 Experiments and Results

5.1 Experimental Setup

We report experimental results for a wide range of data structures and point behaviors. The various criteria we considered are summarized in Table 2.

The alternative algorithms were chosen to contrast with the kinetic data structures in [BGH97]: DELAUNAY is general purpose, while BRUTE-CH is highly non-local. The point distributions were chosen because the size of the static convex hull for these distributions ranges from $O(\log n)$ to $O(n)$. Finally, we chose to implement bouncing to differentiate data structures that can deal gracefully with flight plan updates and those (like BRUTE-CH) that cannot.

We ran each test 5 times, on inputs ranging from 10 points to 10,000 points. We report only the average values. The values are the number of external events, the total number of events, and the number of equations solved (classified by degree). In addition, we plot the total *weighted* cost of solving equations for each test. This statistic is motivated by the fact that most of the time spent by the data structures we consider (above 80 percent, for DELAUNAY) is spent solving equations used in scheduling events. We weigh each equation by the time it takes to solve: degree 1 equations (which are dominated by division) take 1 time unit, degree 2 equations (which are dominated by square root) take 4 time units, and degree 4 equations (which require either many square and cube roots or an iterative algorithm) take 68 time units. These ratios were obtained empirically by repeatedly solving several random equations of each type (on both a 90 MHz. Pentium and a 167 Mhz. UltraSparc-I) and averaging the CPU times.

For BOUNCING, we ran each test three times, once where we stopped the simulation after $n/10$ bounces, once after n bounces, and once after $10n$ bounces, where n is the number of points. For every test, all statistics scaled linearly with the number of bounces, so we report only data for n bounces. The boundary wall for each distribution is the boundary of the distribution: a square for SQUARE and a circle for CIRCLE and DISK.

Category	Name	Description	Comments
Data Structure	BGH97-CH	For convex hull	$\Theta(n)$ internal events
	BRUTE-CH	For convex hull	Highly non-local
	DELAUNAY	For convex hull or closest pair	Degree 4 equations
	BGH97-CP	For closest pair	Mostly degree 1 equations
	BGH97-CP'	For closest pair	Harder to implement than BGH97-CP
Point Distribution	SQUARE	Uniform in a square from $(-1,-1)$ to $(1,1)$	Hull changes $O(\log^2 n)$ times
	DISK	Uniform on a unit disk	Static hull is large
	GAUSSIAN	Circularly symmetric normal distribution	Convex hull changes rarely
	CIRCLE	On the edge of a circle	Convex hull changes often
Point Behavior	NO BOUNCING	Constant velocity	
	BOUNCING	Points bounce when they reach the edge of the distribution (not appropriate for Gaussian)	Locality is important

Table 2: The criteria used for selecting the experiments.

All DELAUNAY tests were conducted on an SGI using an 8 processor MIPS R4400 at 250MHz., running IRIX 6.2. The code was written in C++ and compiled with the SGI `CC` compiler. The rest of the CLOSEST PAIR tests were run on an SGI using a MIPS R4600 processor at 133 Mhz., running IRIX 5.3. The programs were written in C++ and compiled using the SGI `CC` compiler with the `-O2 -mips2` optimization options. The CONVEX HULL and MAXIMUM tests were run on a 167 Mhz. UltraSparc-I processor running Solaris 2.5.1. The programs were written in C and compiled using the SUN `gcc` compiler version 2.7.2.1, using the `-O6` optimization option.

5.2 External vs. Total Events

As Figure 3 shows, the number of external events for CONVEX HULL depends very much on the distribution. This stands to reason, since the distributions were chosen to differ on this property. Figure 4, however, shows that the total number of events processed by BGH97-CH , internal plus external, is remarkably independent of the distribution for CONVEX HULL: it is linear for all distributions except CIRCLE, for which it is slightly superlinear. This near-independence of the number of internal events from the distribution is observed also in DELAUNAY , which solves about $O(n^{3/2})$ equations during a simulation, and in BGH97-CP' , which has the same asymptotic behavior as DELAUNAY , while the closest pair itself changes only linearly many times. Thus the ratio C_K/C_F of Section 1 is not good for any of the methods examined here. It is more interesting to compare the relative performance of the different methods we benchmarked for each configuration function, as we now proceed to do.

5.3 Results for CONVEX HULL

Figure 5 shows the performance of various data structures on CONVEX HULL under the GAUSSIAN distribution. Because GAUSSIAN has fewer events than other distributions, BRUTE FORCE does rather well; recall that BRUTE-CH is severely non-local and takes linear time to handle events. Even for GAUSSIAN, however, BRUTE-CH performs no better than BGH97-CH for more than 200 points, and its rate of growth is greater. The non-locality of BRUTE-CH is more of an issue for BOUNCING, since for BRUTE-CH each bounce requires a linear amount of computation. The high rate of growth of BRUTE-CH is evident in Figure 6. For the number of points we consider, the poor performance of DELAUNAY for CONVEX HULL comes from two factors: the higher asymptotic growth of the number of events, and the huge cost involved with solving degree 4 equations.

Since BGH97-CH is a divide-and-conquer data structure, it is possible to obtain a constant factor speedup by avoiding recursing all the way to the bottom. For instance, we could stop the recursion at 20 points and use BRUTE-CH at that point. Another option would be to maintain an arbitrary triangulation instead of the Delaunay triangulation, since this would only involve degree 2 equations. We will present results for this hybrid algorithm in the full paper.

5.4 Results for CLOSEST PAIR

Three data structures for maintaining the closest pair are compared in Figure 7. Delaunay is the slowest, even when we don't consider the extra machinery needed to recover the closest pair from the Delaunay structure, but this is now due only to the high cost of solving degree 4 equations. Indeed, as expected, BGH97-CP grows at quadratic rate, although with a small constant, as the quadratic behavior affects only the number of degree 1 equations. The new BGH97-CP' is strictly superior to BGH97-CP since it is similar in construction but has fewer events. Despite the fact BGH97-CP' has only 1/20 as many events as BGH97-CP for 10,000 points, its cost of solving equations is only marginally better. This is, once again, because the savings translate entirely to a reduced number of degree 1 equations, which are much less expensive than degree 2 equations.

5.5 Discussion

The results above indicate that the kinetic data structures from [BGH97] are very competitive with alternative data structures for moving points, but the advantages are not exactly the theoretical advantages discussed in that paper. Locality, though an important theoretical concern, does not seem to be an issue for the distributions we studied, except for BRUTE-CH which is flagrantly non-local. For instance, DELAUNAY does not degrade in performance when points bounce, as we would expect if its non-locality were a pronounced issue. On the other hand, the non-theoretical issue of equation complexity *is* important, making DELAUNAY a slower data structure than the kinetic structures. Nevertheless, DELAUNAY is attractive due to its ease of implementation.

It should be possible to prove the results we obtained in this paper with a probabilistic analysis. A companion paper [BDIZ] gives a number of results in this direction.

6 Conclusion

In this paper, we reported the implementation of and experimentation with several kinetic data structures for two archetypical configuration functions: the closest pair and the convex hull for points moving in the plane. Our primary finding is that kinetic data structures do not pose major implementation problems and perform well on several natural distributions. For a large number of points, alternative data structures suffer from non-locality and expensive root-finding operations they perform. However, all data structures stand to gain from more careful implementation, particularly from using simpler data structures when the number of points is small.

There is ample opportunity for further experimentation. The data structures presented here can handle arbitrary changes in velocity, but we have studied only two well-behaved possibilities for adjusting the motions. Furthermore, the distributions we studied change the configuration function at most $O(n)$ times, though some relatively artificial distributions can change the function up to $O(n^2)$ times and may prove illuminating for this reason.

Several other kinetic data structures can be imagined for each of the problems we looked at. For instance, for the convex hull, it is enough to maintain any triangulation, not just the Delaunay triangulation. This is desirable since arbitrary triangulations require only degree 2 equations to maintain. For the closest pair, we can maintain the L_1 Delaunay triangulation [CD85] with a kinetic tournament on its edges (a provably efficient solution [Che93], although non-local). These alternative data structures can be evaluated both in the theoretical framework of [BGH97], in a probabilistic setting [BDIZ], and in the experimental framework of this paper. The most fundamental kinetic structure of all—the kinetic tournament—also has two competitors that might be good in practice due to improved locality: a heap [BGR], and a heater [BGR96]; experimentation is needed to compare these alternatives.

Our handling of precision problems is not ideal, and needs theoretical justification. Many open questions remain. Is it possible to order events without explicitly solving the relevant equations (e.g., by deciding if the i-th root of one equation is less than the j-th root of another equation)? Is it possible to develop a "resilient" KDS that can gracefully handle events that are out of order?

Acknowledgments

We wish to thank Aris Gionis and Piotr Indyk for useful discussions.

References

[Ata85] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11:1171–1181, 1985.

[BDIZ] J. Basch, H. Devarajan, P. Indyk, and L. Zhang. Combinatorial geometry of randomly moving particles. Submitted to SoCG 97.

[BGH97] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *SODA*, page to appear, 1997.

[BGR] J. Basch, L. J. Guibas, and G.D. Ramkumar. Sweeping over lines and line segments with a heap. Submitted to SoCG 97.

[BGR96] J. Basch, L.J. Guibas, and G.D. Ramkumar. Reporting red-blue intersections between connected sets of lines segments. In J. Diaz and M. Serna, editors, *Algorithms — ESA '96*, LNCS 1136, pages 302–319, sep 1996.

[CD85] L. P. Chew and R. L. Drysdale, III. Voronoi diagrams based on convex distance functions. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 235–244, 1985.

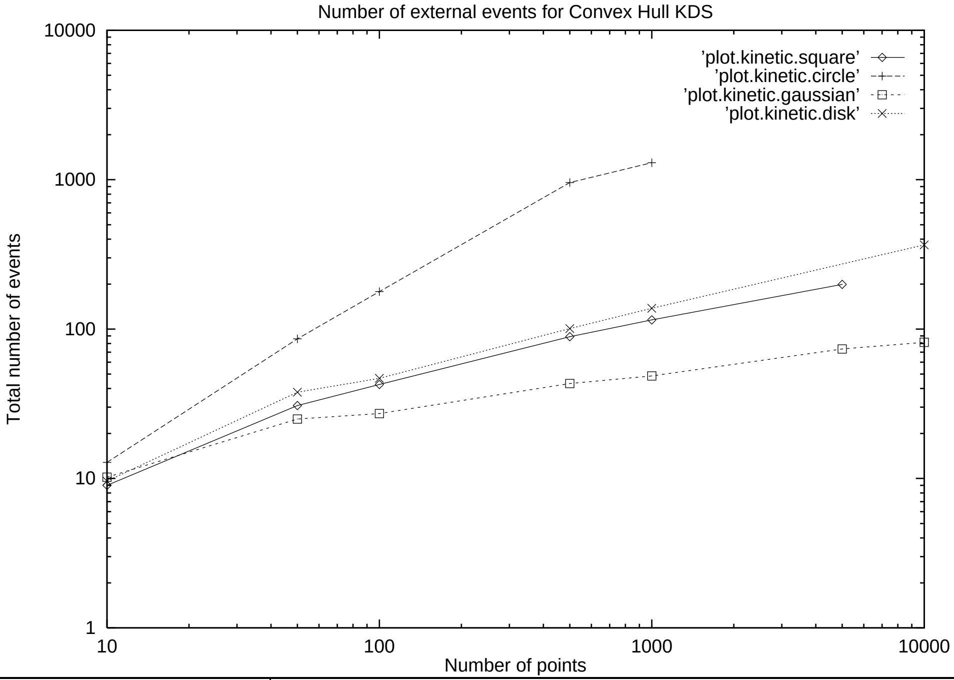
[Che93] L. P. Chew. Near-quadratic bounds for the L_1 Voronoi diagram of moving points. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 364–369, Waterloo, Canada, 1993.

[CT92] Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proc. IEEE*, 80(9):1412–1434, September 1992.

[GMR91] 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.

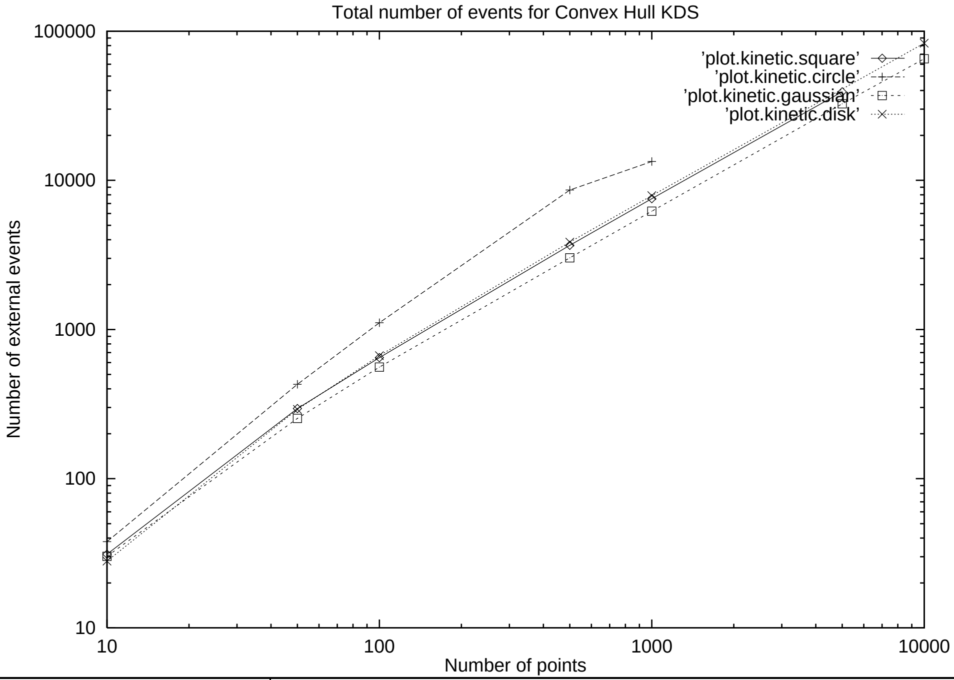
[Gui] L. Guibas. Unpublished.

[ST85] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.



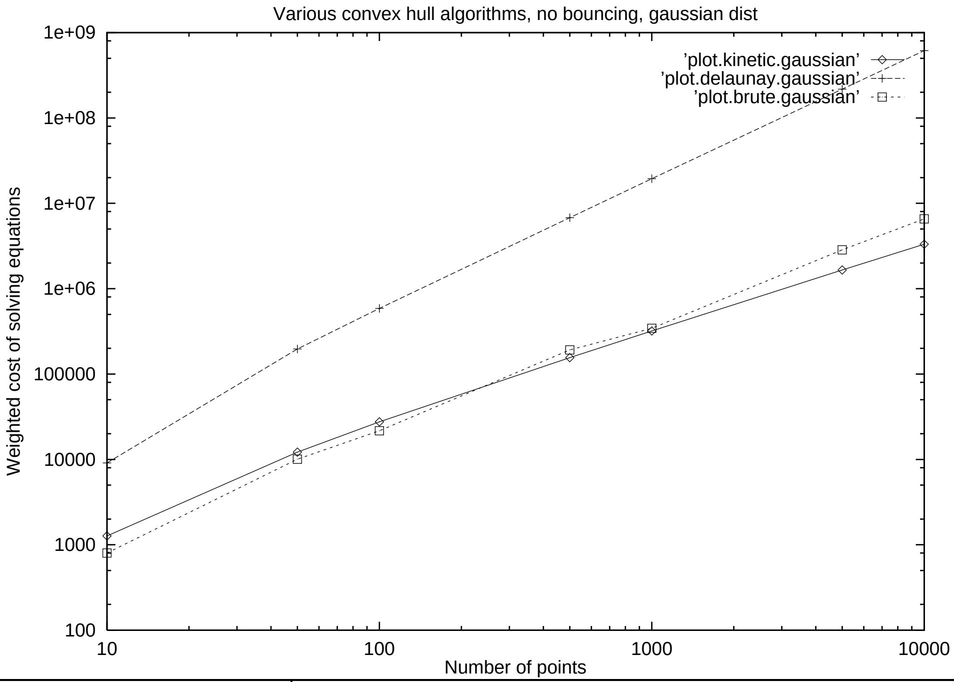
dist.	statistic	10	50	100	500	1000	5000	10000
kinetic /	ext. events	9	31	43	89	115	199	0
square	total events	31	295	647	3666	7540	39231	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	341	3466	7811	46269	95685	493503	0
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	13	86	178	956	1299	0	0
circle	total events	38	430	1112	8609	13381	0	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	393	5003	13274	106329	165497	0	0
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	10	25	27	43	49	74	82
gaussian	total events	30	253	559	3025	6210	32602	65261
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	317	3044	6889	38927	79885	413533	827930
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	10	38	47	101	138	0	366
disk	total events	28	292	668	3858	7907	0	83065
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	305	3506	8223	49012	100594	0	1048299
	degree-4 eqns.	0	0	0	0	0	0	0

Table 3: The number of external events on CONVEX HULL under various distributions. Though we use BGH97-CH , the number of external events is independent of the data structure used.



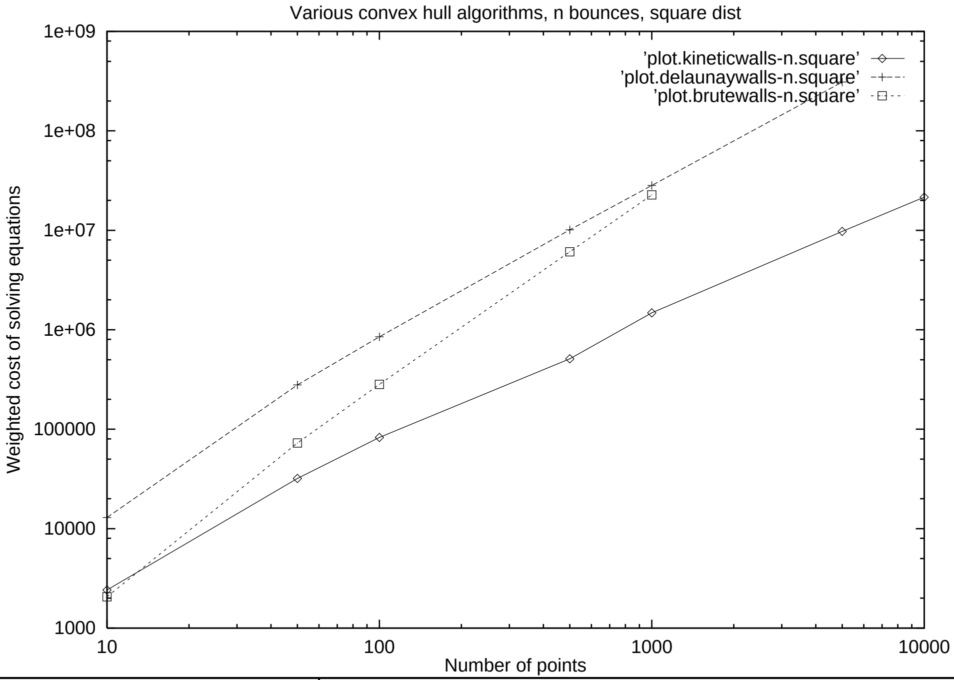
dist.	statistic	10	50	100	500	1000	5000	10000
kinetic /	ext. events	9	31	43	89	115	199	0
square	total events	31	295	647	3666	7540	39231	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	341	3466	7811	46269	95685	493503	0
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	13	86	178	956	1299	0	0
circle	total events	38	430	1112	8609	13381	0	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	393	5003	13274	106329	165497	0	0
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	10	25	27	43	49	74	82
gaussian	total events	30	253	559	3025	6210	32602	65261
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	317	3044	6889	38927	79885	413533	827930
	degree-4 eqns.	0	0	0	0	0	0	0
kinetic /	ext. events	10	38	47	101	138	0	366
disk	total events	28	292	668	3858	7907	0	83065
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	305	3506	8223	49012	100594	0	1048299
	degree-4 eqns.	0	0	0	0	0	0	0

Table 4: The total number of events for CONVEX HULL, using KINETIC, under various distributions.



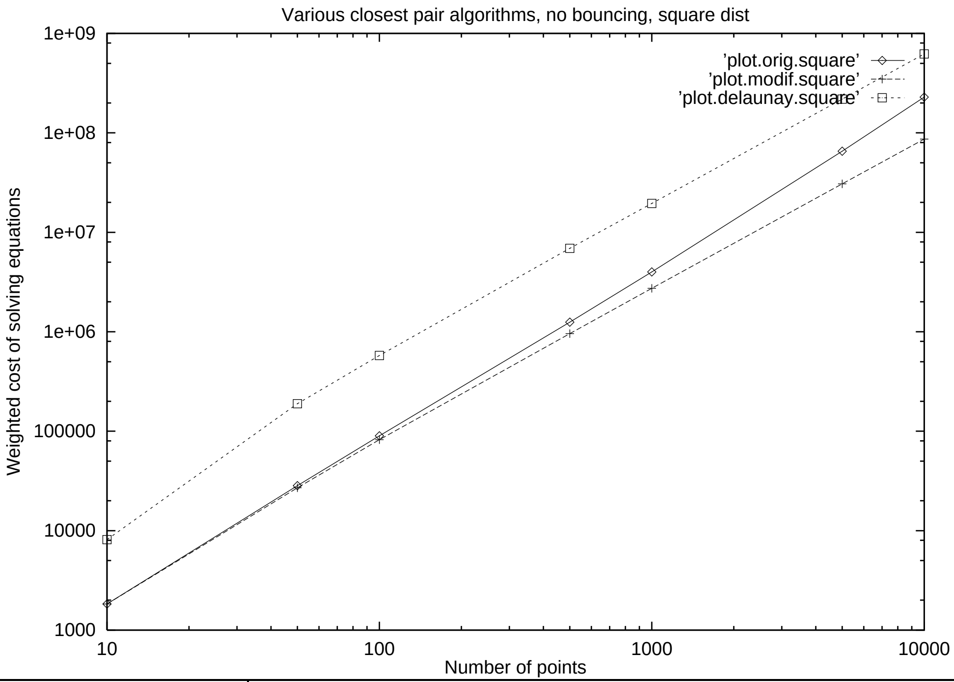
dist.	statistic	10	50	100	500	1000	5000	10000
kinetic /	ext. events	10	25	27	43	49	74	82
gaussian	total events	30	253	559	3025	6210	32602	65261
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	317	3044	6889	38927	79885	413533	827930
	degree-4 eqns.	0	0	0	0	0	0	0
delaunay /	ext. events	9	23	35	47	51	72	79
gaussian	total events	46	624	1795	20092	57158	639544	1807663
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	53	126	188	257	273	389	422
	degree-4 eqns.	131	2880	8615	99968	285265	3196968	9037496
brute /	ext. events	10	25	27	43	49	74	82
gaussian	total events	10	25	27	43	49	74	82
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	200	2518	5415	47960	85933	714070	1640452
	degree-4 eqns.	0	0	0	0	0	0	0

Table 5: Cost of solving equations for CONVEX HULL under GAUSSIAN. BRUTE-CH does fairly well due to the small number of events.



dist.	statistic	10	50	100	500	1000	5000	10000
kineticwalls-n /	ext. events	12	101	213	916	2353	11981	24089
square	total events	40	559	1447	8691	25354	168829	372349
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	604	7980	20640	127466	370168	2438562	5377456
	degree-4 eqns.	0	0	0	0	0	0	0
delaunaywalls-n /	ext. events	12	100	219	1156	2372	11981	0
square	total events	52	934	2722	30620	84839	915620	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	112	752	1573	8001	16147	80988	0
	degree-4 eqns.	184	4051	12383	148035	415126	4550667	0
brutewalls-n /	ext. events	12	101	213	1167	2353	0	0
square	total events	12	101	213	1167	2353	0	0
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	516	18182	70536	1518235	5665472	0	0
	degree-4 eqns.	0	0	0	0	0	0	0

Table 6: Cost of solving equations for CONVEX HULL under SQUARE when the points can bounce n times (n is the number of points). BRUTE-CH suffers from the higher number of events, due both to the different distribution and the bouncing of points.



dist.	statistic	10	50	100	500	1000	5000	10000
orig /	ext. events	7	38	79	388	782	3800	7540
square	total events	94	2200	8620	200981	788441	2399062	76257468
	degree-1 eqns.	148	3699	15037	374536	1497214	37428913	149906375
	degree-2 eqns.	421	6159	18713	218327	623959	7007647	19760109
	degree-4 eqns.	0	0	0	0	0	0	0
modif /	ext. events	7	38	79	388	782	3800	7540
square	total events	81	1226	3683	42502	120988	1361651	3832860
	degree-1 eqns.	148	2376	7208	84228	240412	2715702	7650640
	degree-2 eqns.	421	6159	18713	218327	623959	7007647	19760109
	degree-4 eqns.	0	0	0	0	0	0	0
delaunay /	ext. events	9	33	46	88	111	168	190
square	total events	43	623	1789	20493	57614	643490	1827536
	degree-1 eqns.	0	0	0	0	0	0	0
	degree-2 eqns.	53	179	246	478	605	918	1051
	degree-4 eqns.	116	2772	8470	101548	286907	3215693	9135677

Table 7: Cost of solving equations for CLOSEST PAIR under SQUARE. BGH97-CP' has many fewer events than BGH97-CP , but since these events take only degree 1 equations to schedule, the time savings are not as significant as the reduction in events would indicate.