A Practical Evaluation of Kinetic Data Structures

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

Computer Science Department
Stanford University
Stanford, CA94305
{jbasch,guibas,csilvers,lizhang}@cs.stanford.edu

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, Basch, Guibas, and Hershberger [BGH97] proposed the framework of *kinetic data structures* (KDSs) as a way to maintain, in a completely on-line fashion, 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. The KDS framework allows each object to change its motion at will according to interactions with other moving objects, the environment, etc.

We implemented the KDSs described in [BGH97], as well as some alternative methods serving the same purpose, as a way to validate the kinetic data structures framework in practice. In this note, we report some preliminary results on the maintenance of the convex hull, describe the experimental setup, compare three alternative methods, discuss the value of the measures of quality for KDSs proposed by [BGH97], and highlight some important numerical issues.

2 Kinetic Data Structures

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 distance between all pairs of the points. Even though this distance changes continuously, it can be derived from a combinatorial function F — describing the identity of the pair of points that realize this distance — 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 at that instant. These certificates are typically low degree algebraic inequalities involving a few of the moving points, such as "point p is to the left of point q," or "the distance $d(p,q)$ is greater than the distance $d(p,r)$."

In the KDS framework each point is assumed to follow a particular *flight plan*. Typically we will assume that these motions are piecewise algebraic of low degree. Under this constraint, for a given certificate c, we can calculate the next future time $t(c)$ at which c becomes false by solving for "the next larger" root of an algebraic equation,. 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, and appropriately updating the event queue. 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 (*responsiveness*);
- the overhead of internal events with respect to external events is reasonable (*efficiency*);
- the KDS itself is of small size, typically linear or slightly superlinear (*compactness*); and
- each point is involved in only a small number of certificates (*locality*).

Precise ways to make these criteria quantitative are discussed in [BGH97]. 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). 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 different configurations of moving points.

The *locality* condition is important when points change their motion law, something we call a *flight plan update*.

*Supported in part by National Science Foundation grant CCR–9623851 and by US Army MURI grant DAAH04–96–1–0007.

†Supported by the Department of Defense, with partial support from NSF Award CCR–9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

To appear in the 13th Symposium of Computational Geometry, 1997.

This is common in typical simulations, where for instance points may bounce off walls or off each other. When a point changes motion, the failure times of all certificates dependent on that point must be updated. The locality condition ensures that this operation can be performed within reasonable time bounds.

3 Three ways to maintain a convex hull

Along with the method proposed in [BGH97], we implemented two other methods for comparison: the Delaunay triangulation and a less sophisticated method, dubbed "brute force." All these methods share the traits that they are event driven and exact.

BGH97-CH : The principle of the kinetic data structure proposed in [BGH97] for the maintenance of the convex hull is the following: the set of points is divided arbitrarily into a red half and a blue half, and the convex hull of each half is computed and maintained recursively. The convex hull of the whole is then certified by the slope ordering between all red and blue convex hull edges and by the orientation of a set of bichromatic triangles. If the motion is algebraic of degree k, it is possible to compute the time at which two edges become parallel, and the time at which a triangle becomes degenerate, by solving in each case an equation of degree $2k$. In [BGH97], it is shown that, with some small adaptation, this kinetic data structure is both local and efficient.

DELAUNAY : An alternate way to maintain the convex hull of moving points is to maintain their Delaunay triangulation, whose edges form a superset of the convex hull. This configuration is especially straightforward to kinetize, as its set of edges, together with "InCircle" tests that certify "local Delaunayhood," is a correct certificate structure [GMR91]. As a way to maintain the convex hull, this KDS is not local (a given point can have an arbitrarily high degree in the underlying graph) and is not known to be efficient (the best known upper bounds for the number of changes to the DT when points move along algebraic trajectories is roughly cubic). If the motion of each point can be described by algebraic pieces of degree k, the event times for the Delaunay certificates will be roots of polynomials of degree $4k$.

BRUTE-CH : There is a simple, brute force data structure for maintaining a convex hull. 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. (When a non-hull point updates its flight plan, we need only recalculate its own event time.) Furthermore, whenever a point enters or leaves the convex hull, all events must be rescheduled. Hence, as one flight plan update causes the rescheduling of up to $\Omega(n)$ events, this structure is not local.

4 Numerical issues

The major problem in implementing robust kinetic data structures is the approximate numerical computation of roots of polynomials. As the same issues arise for any implementation of a line sweep algorithm on curves (for which we are not aware of any general implementation), the solution we propose is of independent interest.

To focus on the specific problems posed by a KDS, we make the 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: (1) events can get out of order and (2) an event that really happens in the future can appear to have already happened, or vice versa. We discuss the second problem, which occurs frequently; we never observed an occurrence of the first problem.

To be concrete, 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 $t = t_0$. Because of numerical error, however, this root might appear to be at some $t_\epsilon > t_0$, and the event that was just processed will be improperly rescheduled. To avoid this problem, we discard the root closest to t_0 before finding the next event time.

Using this method, we were able to run reliable simulations involving 50,000 points, or about 400,000 events for the BGH97-CH KDS.

5 Experimental setup

In order to remain independent of specific implementation details, the cost of a KDS was taken to be a weighted sum of the number of polynomial equations solved. This statistic is motivated by the fact that most of the time spent by the data structures we consider (over 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 80 time units. These ratios were obtained empirically by repeatedly solving several random equations of each type (on a 90 MHz. Pentium; a 167 Mhz. UltraSparc-I; and an 8 processor, 250 MHz. MIPS R4400).

We ran the three methods described in section 3 with n ranging from 10 to 10000 points with initial positions and speeds chosen independently at random in the unit square.[1] The first set of experiments was intended to measure the efficiency on average (as opposed to worst case) of the different kinetic data structures. For this purpose, we let the simulation run until the convex hull stabilized and there was no more events in the event queue (Figure 1). The second set of tests was designed to judge of the importance of the locality requirement for a KDS: we introduced walls against which the points bounced, and let the simulation run until n bounces had occurred (Figure 2). Since each bounce requires rescheduling every event associated with the bouncing point, non-local structures are expected to perform poorly. For each test, the result was averaged over five runs.

All 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 with the `-O2 -mips2` optimization options.

6 Results

This study allowed us a close examination of the practical value of the quality measures for KDSs proposed by [BGH97] and highlighted some important issues that did not arise in

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

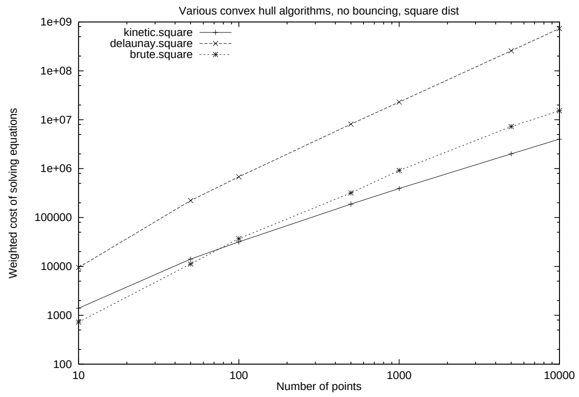

Table 1: Cost (weighted sum of the number of equations solved) of the three different methods that maintain the CONVEX HULL, when the points move along an unchanging, linear flight plan.

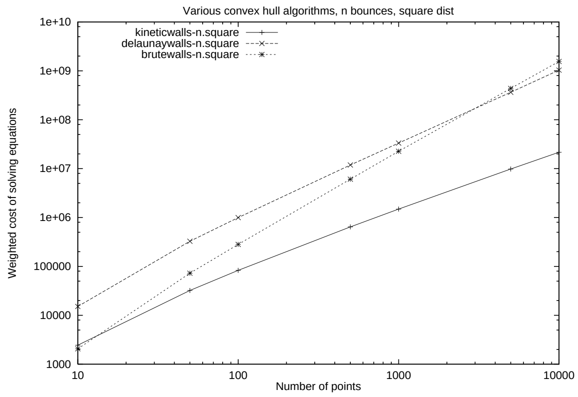

Table 2: Cost (weighted sum of the number of equations solved) of the three methods for maintaining the convex hull, when the points bounce inside a square. The simulation was run until there were n bounces (n is the number of points).

the theoretical study. The first important conclusion, however, is that the implementation of kinetic data structures did not pose any major problem and the framework is readily practical. It provides a solid paradigm for dynamic simulations that avoids approximation due to arbitrary time discretization.

On the side of efficiency, the number of equations solved can be seen to be roughly linear for BGH97-CH , roughly $\Theta(n^{3/2})$ for both DELAUNAY and BRUTE-CH . The worst case is therefore far from being attained, but, as the number of external events itself is extremely low (about $\log^2 n$), the efficiency ratio is in all cases rather high. It therefore makes more sense to compare the relative performance of the different methods. In this sense, BGH97-CH is vastly superior to Delaunay but only marginally better than BRUTE-CH for the numbers considered (Figure 1).

Simulations with bouncing were performed to test the value of the locality requirement. We mentioned earlier that DELAUNAY was not a local KDS in the worst case. However, in this context, as the expected degree of a vertex is constant, the performance of DELAUNAY doesn't degrade with bouncing. On the other hand, BRUTE-CH , which is non-local at all points, has a quadratic growth due to the intense rescheduling necessary upon each bouncing. Finally, BGH97-CH handles bouncing gracefully as expected (Figure 2).

There is one particular aspect of a KDS that is not taken into account in the framework of [BGH97]: it is the cost of solving a single equation. The huge cost of solving degree 4 equations for DELAUNAY is the only reason why this method performs so poorly for n up to a hundred. In this range, the number of equations solved is comparable with that of BGH97-CH . This suggests that it would be worthwhile to maintain some arbitrary triangulation instead of the Delaunay. An arbitrary triangulation can be maintained using only collinearity tests and might well perform better than BGH97-CH for small n.

7 Conclusion

In this paper, we reported the implementation of and experimentation with several data structures for maintaining the convex hull of points continuously 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 or the expensive root-finding operations they perform. However, all methods stand to gain from more careful implementation. Particular advantage can be gained by using simpler data structures when the number of points is small.

Several other kinetic data structures can be imagined for maintaining the convex hull. As mentioned above, it is enough to maintain any triangulation, not just the Delaunay one. 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. We will report in a forthcoming paper on experiments with kinetic data structures for maintaining other configuration functions, such as the closest pair and the maximum.

A program demonstrating kinetic data structures is available at `http://graphics.stanford.edu/~jbasch/demokin`.

Acknowledgments

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

References

[BDIZ] J. Basch, H. Devarajan, P. Indyk, and L. Zhang. Probabilistic analysis for combinatorial functions of moving points. This volume.

[BGH97] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *8th Symposium on Discrete Algorithms*, pages 747–756, 1997.

[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.