Computer Modeling of Motion

Mobile Objects in Graphics, Robotics, Vision, and Geometry

◇◇

Leonidas J. Guibas
Stanford University

◇◇

Why Study Motion?

- motion is ubiquitous in the physical world
- motion is of central importance in graphics, robotics, and vision
- we do not yet possess unified theories of motion: vast gap between high-level and low-level motion descriptions
- motion happens over time; this means that we need on-line and real-time techniques.

Motion in Computer Vision

Areas	Issues
motion estimation shape from motion active vision	robustness error analysis

Motion in Robotics

Areas	Issues
kinematics	inverse problem
dynamics and control	uncertainty
	unpredictability
motion planning	many degrees of freedom

Motion in Graphics

Areas	Issues
viewpoint motion	visibility set, view interpolation
environmental motion	
non-rigid objects	complex dynamics
animation: specification, control	constraints, levels of control

Motion in Geometry

Areas	Issues
collision detection	object hierarchies
sweep methods	event scheduling
event counts	combinatorial complexity

Problem: Keeping track of the interactions of a large number of moving bodies.

Solution: Kinetic Data Structures

Running example: A set of rigid unit disks moving around in the plane. We want to maintain the closest pair of disks.

Configuration Functions

A configuration function (CF) is an attribute of interest of the evolving system. In the moving disks example, it will be the separation of the closest pair of disks.

Even though the CF changes continuously, its combinatorial description changes discretely.

We will be interested in methods for maintaining this combinatorial description of the CF through an event-driven simulation.

Kinetic Data Structures

Classical data structures support a set of queries and updates on the objects in a static set $\mathcal{S}$: queries return the CF of interest, while updates change object attributes (which may in turn change the CF).

Dynamic data structures perform updates which are insertions and deletions of objects to/from the static $\mathcal{S}$.

Kinetic data structures (KDS's) are optimized to maintain the CF of interest under continuous motion of the objects in $\mathcal{S}$. Discrete modifications are also supported.

The Kinetic Model

- Objects move according to posted flight-plans; these can be updated and need not be perfectly known.
- A KDS maintains the CF by scheduling events. Events are typically the failure times of certificates. The currently valid certificates prove the correctness of the CF.
- A certificate is typically an elementary condition involving a small number of the moving objects (disks A and B are closer than C and D). The failure time of a certificate can be calculated from the flight plans of the participating objects.

Key Features of KDS's

A KDS maintains through time a set of certificates whose conjunction proves the validity of the current combinatorial description of the CF. The set of certificates is updated as necessary.

- KDS's gain efficiency be exploiting the continuity or coherence of the motion.
- KDS's are completely on-line: objects can change their flight plans at any moment.
- KDS's can support discontinuous changes as well.
- KDS-based simulations are based on discrete events and do not require a discretization of time.

Criteria for Evaluating KDS's

Let $\mathcal{S}$ be a set of n moving objects. We will typically assume that the object motions are pseudo-algebraic w.r.t. the KDS. For a given class of flight plans, a desirable KDS is one which is:

responsive: the cost to update the certificate structure after a certificate failure is small

efficient: the ratio of total (internal plus external) events to external events is small

local: each object partakes in a small number of certificates

Small here means polylog, or $O(n^\epsilon)$. Local KDS's are also compact.

Comparisons with Earlier Work

- dynamic data structures
- dynamic computational geometry [Atallah, '85]
- 'data in motion' [Kahan, '91]
- discrete-event simulation

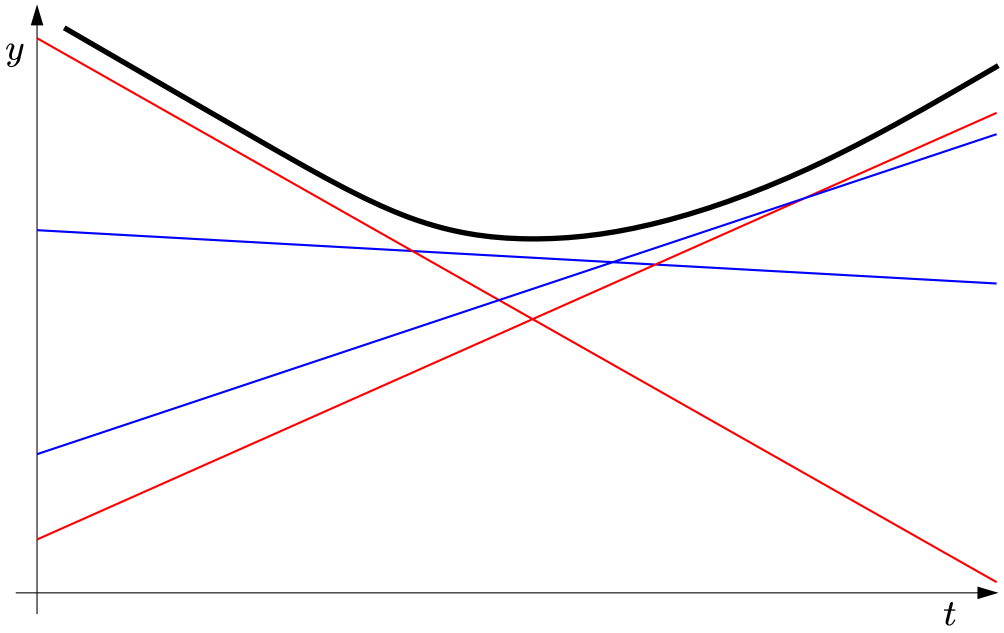
An Illustrative Example: the Maximum Problem

Given n points moving continuously along the y–axis, we want to maintain the point of largest ordinate.

We further assume:

- linear motions
- no collisions

A View in Space-Time

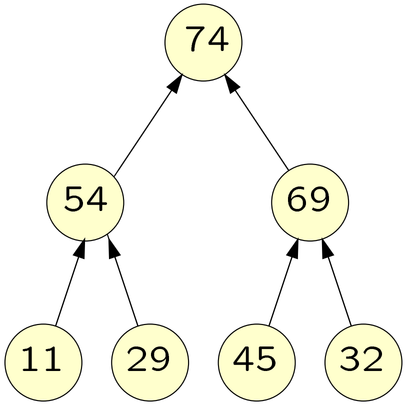


An upper-envelope computation: can be done in $O(n \log n)$ time by D & C — but not on-line.

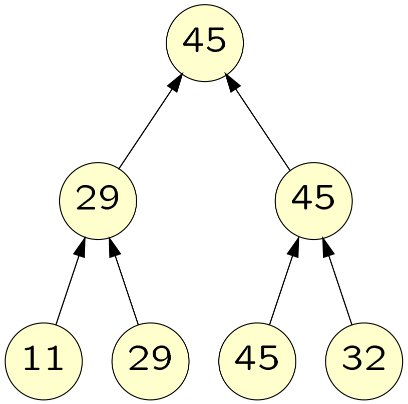
Two more failed attempts

Maintaining a sorted list: not efficient.

Maintaining a kinetic heap: hard to analyze.



The Kinetic Tournament

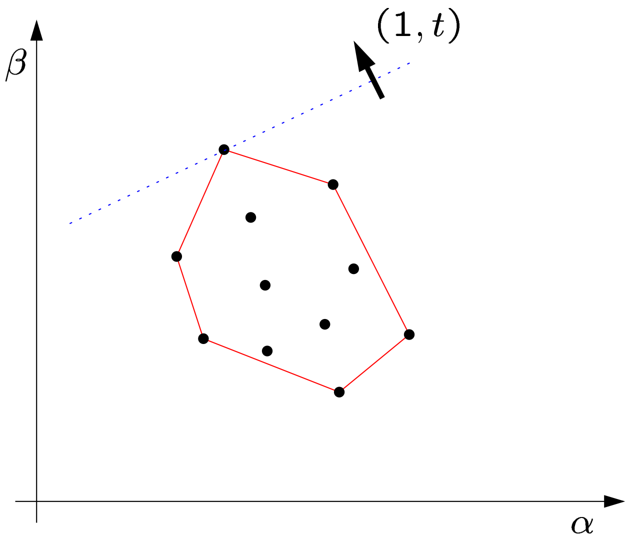


The kinetic tournament (κ-T) is responsive, local, and compact.

The κ-T is also efficient, because events in the κ-T correspond to the comparisons peformed by a D & C algorithm for the upper envelope in space-time.

A ‘Competitive’ Algorithm

Suppose the i-th point moves as $y_i(t) = \alpha_i t + \beta_i$. Then we can maintain the maximum at a cost proportional to the number of changes in the maximum.

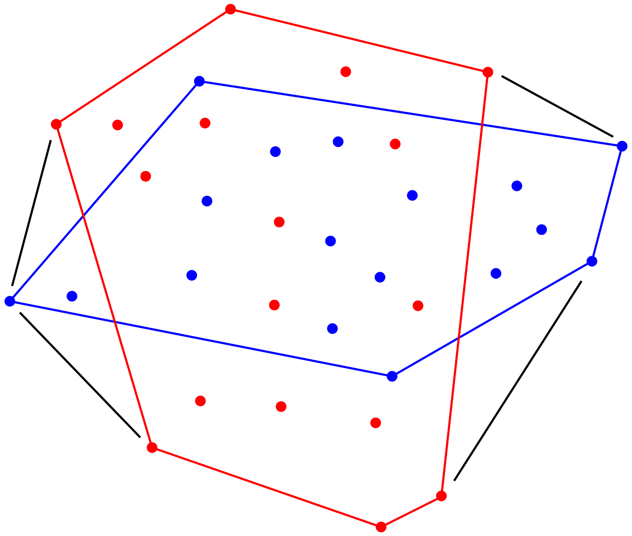


A General Kinetization Strategy

1. Solve static problem for the CF of interest.
2. From the algorithm, extract proof of correctness of the CF as a collection of elementary certificates.
3. Maintain such a certificate collection under object motion.

Kinetic Convex Hull

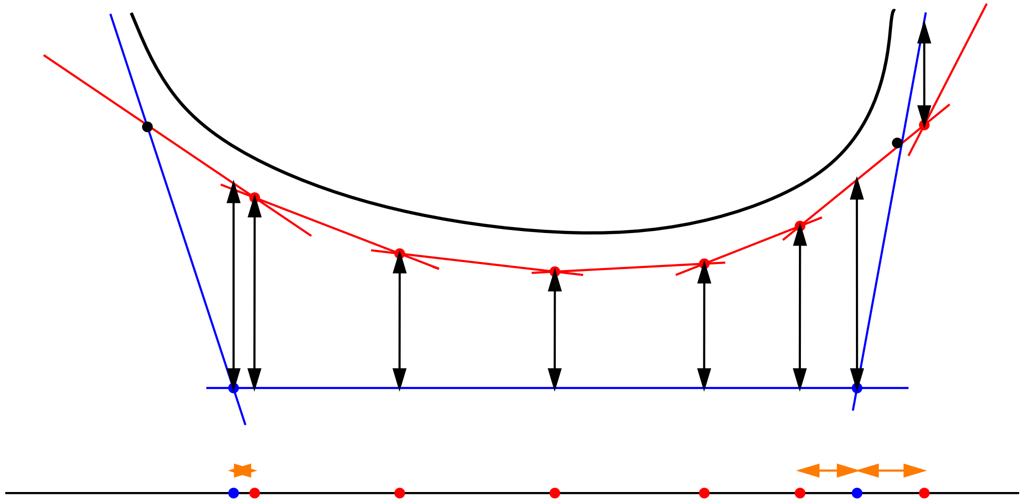
Suppose we are given a set $\mathcal{S}$ of n moving points in the plane. Our kinetic convex hull (κ-CH) algorithm uses a D & C strategy.



We dualize and maintain the upper envelope of moving lines.

Certifying the Red-Blue Merge

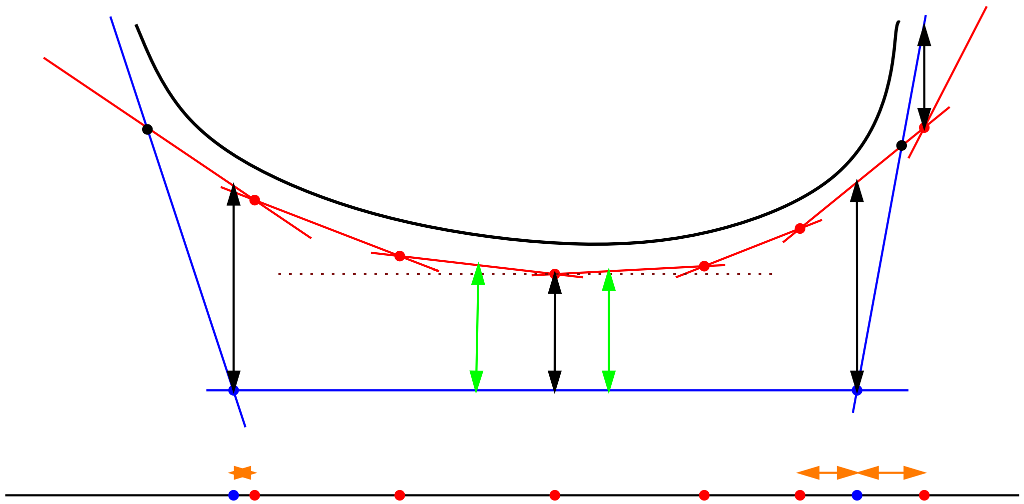
We need to merge two upper envelopes of lines.



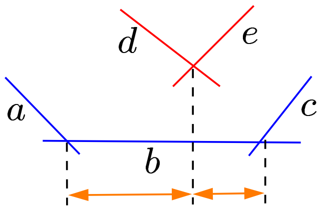
We use x-certificates and y-certificates.

Obtaining Locality

We use slope s-certificates as well, to make the KDS local.

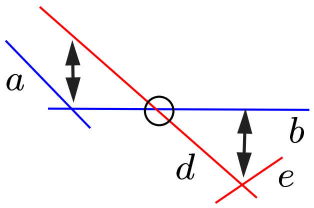


Example Certificate Sets



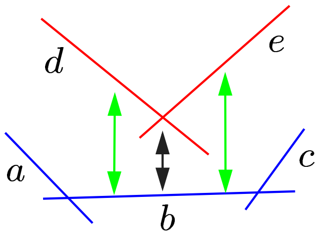
$ab <_x de$
$de <_x bc$

x-ordering certificates



$ab <_y d$
$de <_y b$

intersection certificates

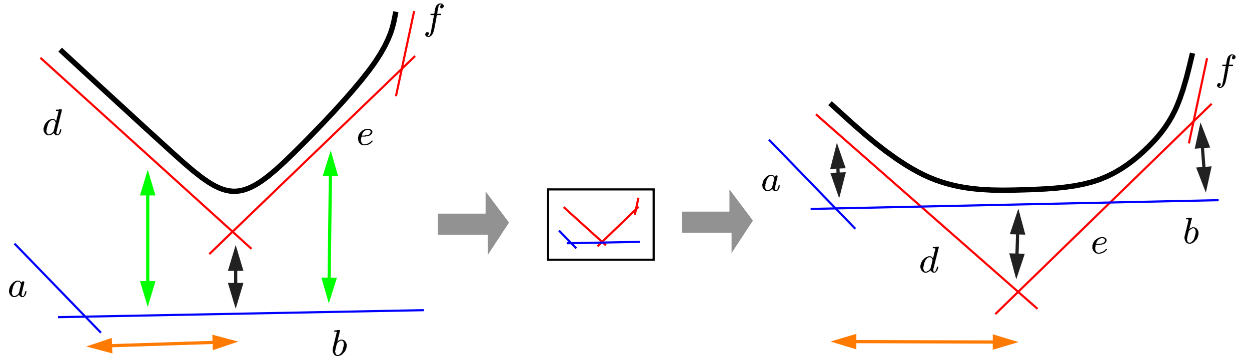


$d <_s b$
$b <_s e$
$b <_y de$

non-intersection certificates

Processing an Event

Below the central y-certificate fails (an y-event). Other events can be processed similarly.



Analysis of the κ-CH Algorithm

In a 3-d space-time, the moving lines define surfaces. The 2-d upper envelopes sweep the surface upper envelope.

The y-events are vertices on the surface upper envelope: their number is $O(n\lambda_s(n)\log n) = O(n^{2+\epsilon})$.

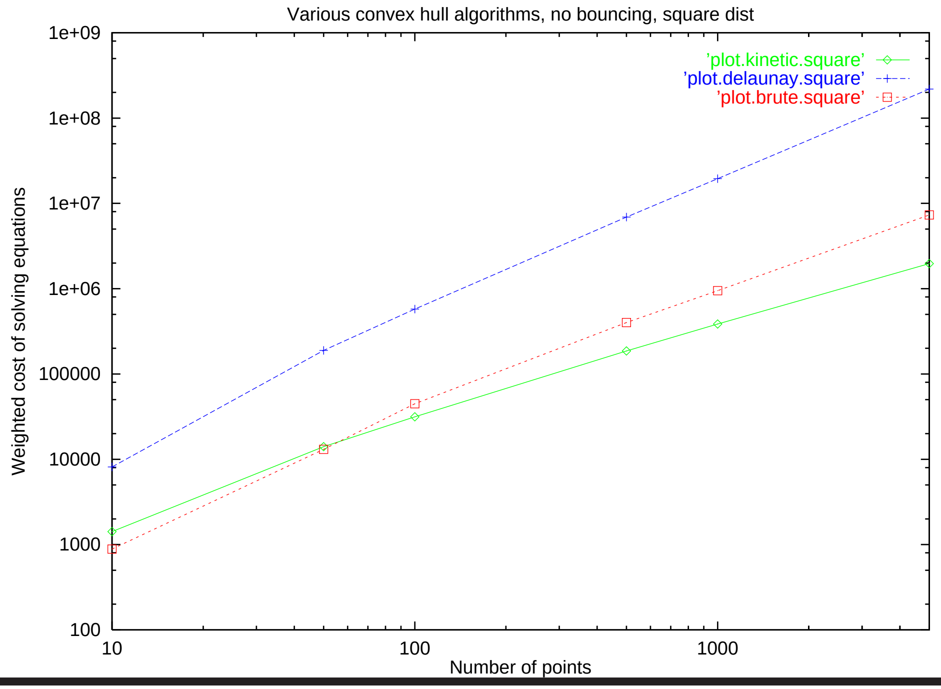
The s-events are at most $O(n^2)$.

The x-events correspond to vertices in the overlay of the red and blue surfaces, which are themselves upper envelopes; their number is also $O(n^{2+\epsilon})$.

Thus κ-CH is efficient (and responsive, local, and compact).

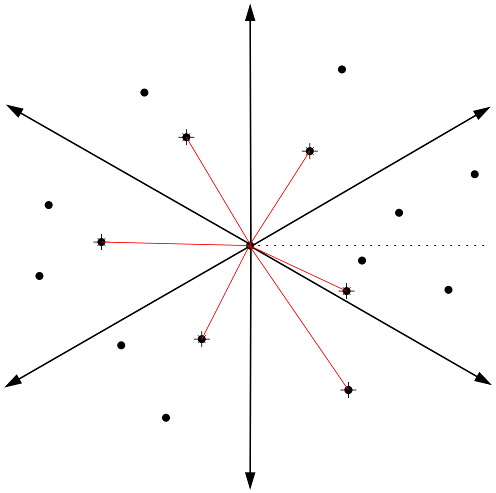
Data from an Implementation

Here we compare κ-CH to Delaunay and a brute force algorithm. Points follow random linear motions in a unit square.



The Closest Pair Problem

Now suppose that we want to maintain the closest pair among n moving points in the plane. Our kinetic closest-pair (κ-CP) algorithm is based on a new static CP algorithm.

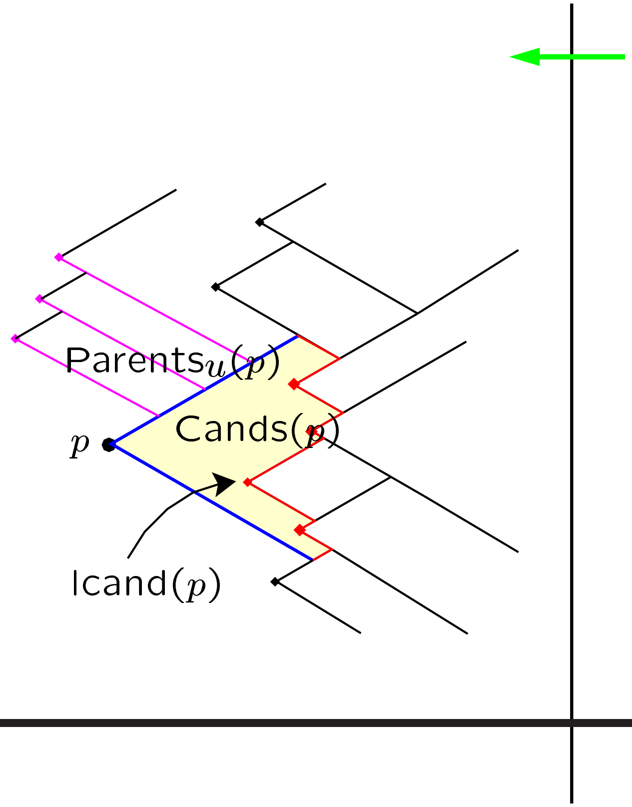


We maintain, for each point, directional nearest neighbors in each of six cones.

The CP must be one of the $6n$ pairs thus defined.

Sweeping to Obtain Directional Neighbors

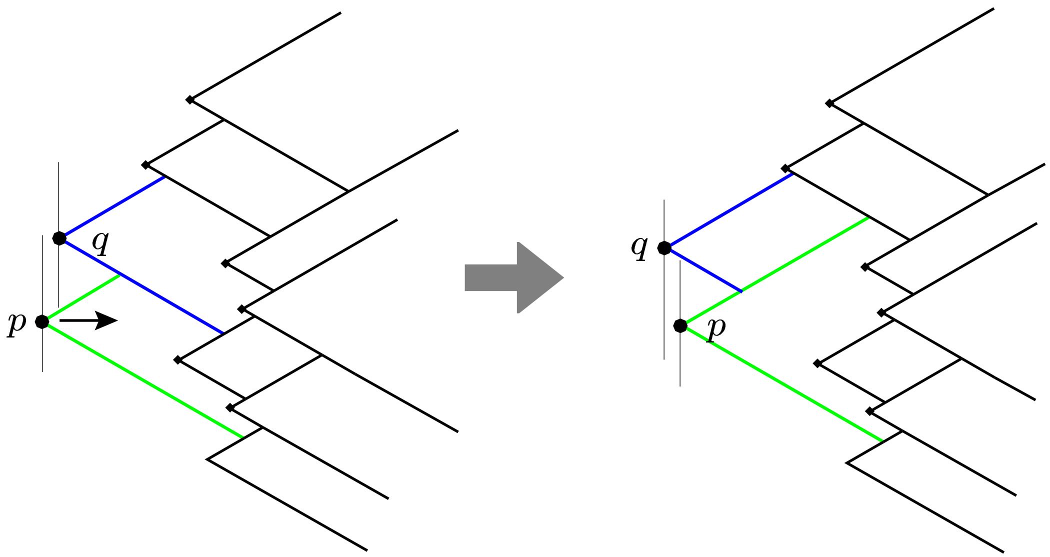
The horizontal sweep computes three data structures for each point p, Cands(p), Parents$_u$(p) and Parents$_d$(p).



Each of the three sweeps can be done in $O(n \log n)$ time and will detect all directional neighbors in its direction.

Kinetizing the CP Algorithm

For κ-CP, each of Cands(p), Parents$_u$(p) and Parents$_d$(p) is maintained as a balanced tree supporting efficient merges and splits. The only relevant events are changes in the order of the points along the 0°, 60°, and 120° directions.



Completing the CP Maintenance

There are $O(n^2)$ order exchange events to process, each of which can be processed in polylog time.

To determine the CP, we neeed to also maintain a κ-T on the distances of all the points to their directional neighbors. This tournament must handle flight plan updates or discrete changes, whenever these directional neighbors change.

It can be shown that that the cost of processing the tournament events is also roughly quadratic.

Thus κ-CP is efficient, responsive, local, and compact.

Other KDS results

- Closest pair in E^d, MSTs [B., G., Zhang]
- Extent measures (width, dimater, etc.) in E^2 [Agarwal, G., H. Veach]
- BSPs in E^2 [Agarwal, G., Murali, Vitter]
- Practical implementation and validation [B., G., Silverstein, Zhang]
- Probabilistic analysis of CFs for moving points [B., Devarajan, Indyk, Zhang]

Conclusions

KDS's provide a new theoretical and practical framework for dealing with continuously variable data. Many open questions remain:

- Which certificate structures evolve gracefully?
- How do we account for the cost of acquiring information about the motions?
- How do we implement KDS's robustly (out-of-order events, global vs. local synchronization)
- How can we make KDS's 'real-time'?

Many problems are waiting to be kinetized.