Kinetic 3D Convex Hulls via Self-Adjusting Computation (An Illustration)

Umut A. Acar
Toyota Technological Institute
Chicago, IL.
umut@tti-c.org

Guy E. Blelloch
Carnegie Mellon University
Pittsburgh, PA.
blelloch@cs.cmu.edu

Kanat Tangwongsan
Carnegie Mellon University
Pittsburgh, PA.
ktangwon@cs.cmu.edu

Categories and Subject Descriptors: E.1 [**Data Structures**]: Kinetic Data Structures—*geometrical problems and simulations*.

General Terms: Algorithms, Design, Experimentation, Performance, Theory.

Keywords: Kinetic data structures, self-adjusting computation, convex hulls.

1. INTRODUCTION

This note and the accompanying video illustrate our solution to kinetic 3D convex hulls using self-adjusting computation. First introduced by Basch, Guibas, and Hershberger [5], the kinetic approach to motion simulations requires maintaining a data structure along with a set of *certificates*: each certificate is a comparison and its failure time (the time at which the outcome of the comparison changes). To simulate motion, an event scheduler updates the certificates in chronological order of their failure times and invokes an *update procedure* that keeps the data structure consistent with the certificates. Even though kinetic data structures for many problems have been proposed and some have already been implemented [12, 11, 10, 6], the problem of kinetic maintenance of 3D convex hulls has remained essentially open [9] (for results on the dynamic version, see Chan's paper [7] and references thereof).

Traditional approaches to kinetic motion simulation require the users to design and implement the update procedure by hand. Recent work proposed an alternative approach based on self-adjusting computation [4]. The approach relies on a generic *change-propagation algorithm* to update the data structure. Self-adjusting computation [2, 1] is a (general-purpose) technique for making static algorithms dynamic. While a static algorithm assumes that its input does not change, a dynamic algorithm can respond to changes to its data, including changes to the outcomes of comparisons, by running the change-propagation algorithm. When paired with an event scheduler, the approach enables kinetizing a program that computes properties of static, non-moving objects. The advantages of the approach include the ability to compose kinetized algorithms and the ability to handle integrated dynamic and kinetic changes automatically. Furthermore, the user needs to code, maintain, and verify correctness of only the static algorithm, as the kinetic version is guaranteed to produce the same output as the static algorithm if executed at that moment.

In the self-adjusting computation model, as a static algorithm executes, we construct a dynamic data structure, called a *dynamic dependence graph (DDG)*, that represents the operations performed during the execution. The nodes of a DDG represent blocks of executed operations, and the edges represent dependence information between nodes. Bodies of function calls constitute a natural notion of blocks in practice. Given a DDG, and any change to computation data, we update the computation by running the change-propagation algorithm. The algorithm identifies the *affected* blocks that use the changed data and re-executes the earliest affected block that does not depend on other affected blocks. Depending on the operations in a block, re-execution can affect other blocks by changing their data, create new blocks, or delete existing blocks due to conditional branches that take a different branch. The change-propagation algorithm recovers previously executed blocks via memoization.

The asymptotic complexity of change propagation for a particular class of changes (e.g., an insertion/deletion) can be analyzed by representing the execution by their traces and measuring the edit distance between them. For a large class of computations, traces can be defined as sets of executed operations, and trace distance can be measured by the symmetric set difference of the sets of executed operations. This analysis technique is called *trace stability* [1, 3].

Previous work evaluated the effectiveness of self-adjusting-computation approach to kinetic motion simulation on a broad number of 1- and 2-dimensional algorithms. This note and the accompanying video illustrate our solution to kinetic 3D convex hulls using self-adjusting computation. We kinetize the randomized incremental convex-hull algorithm [8]. Starting with a single tetrahedron of four non-planar randomly chosen points, the algorithm constructs the hull by inserting the rest of the points one by one and updating the hull after each insertion. To ensure stability, we make small changes to the representation of the data structures used in the standard algorithm. We do not give a stability bound for our algorithm in this paper. To evaluate the effectiveness of our approach experimentally, we implemented the static incremental convex-hull algorithm in the Standard ML language and kinetized it using our library for applying self-adjusting computation techniques [4].

2. EXPERIMENTS

This section reports preliminary experimental results. The experiments were run on a 2.0GHz Power Mac G5 with 2 GB

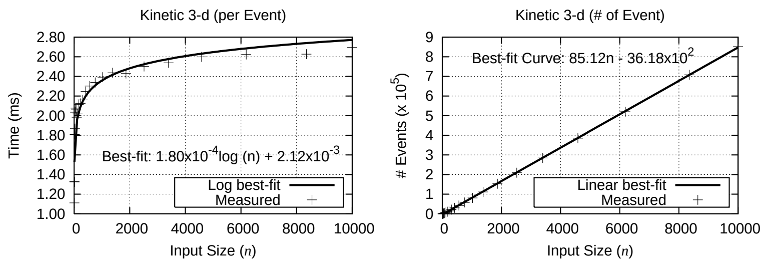

Figure 1: Time per kinetic event and number of events.

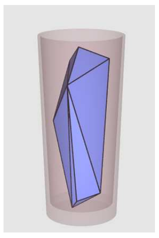

Figure 2: The hull for gas molecules.

of memory. We used the MLton compiler for Standard ML. Since MLton uses garbage collection, the measurements depend on the specifics of its garbage-collection system. We therefore report the *application time*, measured as the total time minus garbage-collection time. Our results rely on a standard floating-point root solver described in earlier work [4]. The inputs for our experiments were generated randomly: each point admits the linear-motion model $\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{v} \cdot t$, where $\boldsymbol{x}_0$ and $\boldsymbol{v}$ are chosen uniformly at random from $[0, 1]^3$ and $[-0.5, 0.5]^3$, respectively.

Figure 1 shows the average time for a kinetic event (left) and the total number of events (right). We also show the least-square-fit curves to the expressions $a \cdot \log n + b$ and $a \cdot n + b$ respectively. Our experiments for integrated dynamic and kinetic changes yield similar results. These experiments indicate that the algorithm is responsive (i.e., responds to kinetic event quickly) and efficient (i.e., processes linear number of events). The experiments indicate that the constant-factors involved in the approach are reasonably small: we observe a linear speedup between kinetic events and re-computing the hull from scratch (the speedup factor reaches 1,500 at 10,000 points).

3. THE MOVIE

The movie starts with an example of computing the convex hull of a set of gas molecules inside a glass. Since molecules can bounce off the walls of the glass and leave the glass, the algorithm for computing the hull should respond to dynamic changes and kinetic changes (due to motion). We describe our kinetization technique based on self-adjusting computation and show experimental results. We then illustrate the two properties of the kinetized algorithms—the ability to respond to integrated dynamic, and kinetic changes and composability—with two examples. First we show a movie of the convex hull being maintained inside of a box as we randomly insert and delete points. Second, we show a movie for computing the points furthest away from each other by composing the convex-hull algorithm with an algorithm that finds points of a list that are furthest away from each other. This algorithm requires $O(m)$ time per kinetic event (m is the number of points on the hull) and is therefore practical when m is small. The movie ends by giving a simulation of the convex hull of gas molecules inside of a glass (a solution to the example). The solution is obtained by composing an algorithm that selects only the points inside a glass with our convex-hull algorithm.

4. REFERENCES

[1] Umut A. Acar. *Self-Adjusting Computation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, May 2005.

[2] Umut A. Acar, Guy E. Blelloch, Matthias Blume, and Kanat Tangwongsan. An experimental analysis of self-adjusting computation. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 96–107, 2006.

[3] Umut A. Acar, Guy E. Blelloch, Robert Harper, Jorge L. Vittes, and Shan Leung Maverick Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 531–540, 2004.

[4] Umut A. Acar, Guy E. Blelloch, Kanat Tangwongsan, and Jorge L. Vittes. Kinetic algorithms via self-adjusting computation. In *ESA 2006: Proceedings of the Fourteenth Annual European Symposium on Algorithms*, pages 636–647, 2006.

[5] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[6] Julien Basch, Leonidas J. Guibas, Craig D. Silverstein, and Li Zhang. A practical evaluation of kinetic data structures. In *SCG '97: Proceedings of the thirteenth annual ACM symposium on Computational geometry*, pages 388–390, 1997.

[7] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1196–1202, 2006.

[8] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*, chapter 11. Springer-Verlag, 2000.

[9] Leonidas Guibas. Kinetic data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. CRC Press, 2004.

[10] Leonidas Guibas, Menelaos Karaveles, and Daniel Russel. A computational framework for handling motion. In *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 129–141, 2004.

[11] Leonidas Guibas and Daniel Russel. An empirical comparison of techniques for updating delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual ACM symposium on Computational geometry*, pages 170–179, 2004.

[12] Daniel Russel. Kinetic data structures. In CGAL Editorial Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.