A Kinetic Data Structures Framework for CGAL

Leonidas J. Guibas* Menelaos I. Karavelas† Daniel Russel‡

1 Introduction

We have developed a CGAL-based framework for implementing kinetic data structure based algorithms along with a package for performing exact operations on the roots of polynomials. A discussion of the kinetic data structures framework was presented at ALENEX 2004 [5]. The kinetic data structures framework provides

- A kinetic kernel which provides moving primitives and predicates acting on them
- A static kernel which allows existing CGAL-based algorithms to be used on snapshots of kinetic data structures
- Support for run time notification of motion changes
- Helper classes to eliminate most broiler-plate code
- Support for exact processing of kinetic data structures
- Implementations of Delaunay and regular triangulations in 2 and 3 dimensions
- GUIs for visualizing and manipulating kinetic data structures

The support for exact kinetic data structures is provided by our polynomial package. This package provides

- Fast comparison of roots of polynomials
- Transparent filtered operations
- Support for many common operations on polynomials
- Support for non-square-free polynomials

Together these packages extend CGAL to provide support for geometric data structures built on top of moving primitives, specifically, data structures which fit in the kinetic data structures paradigm. Kinetic data structures were introduce by Basch et. al. in '97 [1, 4]. They exploit the combinatorial nature of most geometric data structures—the combinatorial structure remains invariant under some motions of the underlying geometric primitives and, when the structure does need to change, it does so at discrete times and in a limited manner.

In the framework, primitives move along smooth trajectories (we provide support for motion along polynomial parametric curves, although other motions can be added if needed). The combinatorial validity of a structure is proved by some set of predicates having the correct values. These predicates are called certificates. For many predicates of interest, such as orientation tests, the value of the predicate is a polynomial in the trajectories of the primitives, so it is a polynomial itself, called the certificate polynomial. The certificate changes value at the roots of its polynomial, at which point the combinatorial structure must be updated. This is called an event. The polynomial package provides support for the necessary operations on polynomials and their roots.

*Stanford University, guibas@graphics.stanford.edu
†University of Notre Dame, mkaravel@cse.nd.edu
‡Stanford University, drussel@graphics.stanford.edu

2 The framework design

The framework is divided into five main concepts as shown in Figure 1. They are:

- KINETICKERNEL: a class which defines kinetic geometric primitives and predicates acting on them. This kernel is analogous to the CGAL kernel and will not be discussed.
- MOVINGOBJECTTABLE: a container which stores kinetic geometric primitives and provides notifications when their trajectories change. In the kinetic data structures paradigm each primitive must move along a piecewise algebraic curve, each algebraic piece of which is known as a trajectory. A primitive's trajectory can change at any time as long as the motion is C^0 continuous. When a trajectory changes, the time of any event whose certificate involves the primitive must be updated, since the certificate polynomial will have changed.
- INSTANTANEOUSKERNEL: a model of the CGAL kernel concept which allows static algorithms to act on a snapshot of the kinetic data structure. This means that any existing CGAL data structure can be used without modifications to initialize or verify a corresponding kinetic data structure. We further discuss the INSTANTANEOUSKERNEL in Section 3.
- SIMULATOR: a class that ensures that events get handled at the correct times. It also helps audit kinetic data structures to make sure that they are valid.
- POLYNOMIALKERNEL: a computational kernel for representing and manipulating polynomials and their roots. This kernel will be discussed in more detail in Section 4.

In addition we provide numerous helper classes and graphical user interfaces to aid in development and debugging.

In a typical scenario using the framework, a SIMULATOR and MOVINGOBJECTTABLE are created and a number of geometric primitives (e.g. points) are added to the MOVINGOBJECTTABLE. Then a kinetic data structure, for example a two dimensional kinetic Delaunay triangulation, is initial-

ized and passed pointers to the SIMULATOR and MOVINGOBJECTTABLE. The kinetic Delaunay triangulation extracts the trajectories of the points from the MOVINGOBJECTTABLE and the current time from the SIMULATOR. It then uses an instance of an INSTANTANEOUSKERNEL and CGAL's Delaunay_triangulation_2 component to initialize the kinetic data structure with the Delaunay triangulation of the points at the current time. An instance of a KINETICKERNEL is used to compute the in_circle certificate function for each edge of the initial Delaunay triangulation. The kinetic data structure requests that the SIMULATOR solve each certificate function and schedule an appropriate event. The SIMULATOR uses the POLYNOMIALKERNEL to compute and compare the roots of the certificate functions.

Initialization is now complete and the kinetic data structure can be run. Running consists of the SIMULATOR repeatedly finding the next event and processing it. Here, processing an event involves flipping an edge of the Delaunay triangulation and computing five new event times. The processing occurs via a callback from an object representing the event to the kinetic Delaunay data structure.

If the trajectory of a moving point changes, for example it bounces off a wall, then the MOVINGOBJECTTABLE notifies the kinetic Delaunay data structure. The kinetic Delaunay data structure then updates all the certificates of edges adjacent to faces containing the updated point and reschedules those events with the SIMULATOR.

3 Use of the CGAL kernel

CGAL's kernel and traits based design to allows us to apply existing implementations of static data structures and algorithms to snapshots of kinetic data. To do this, we provide an model of the CGAL kernel, called the INSTANTANEOUSKERNEL in which the primitives (points, spheres, etc.) are replaced by kinetic primitives. This kernel stores the time of the snapshot it is acting on. When a predicate functor is evaluated, it requests that the kernel compute the static representation of the arguments as they would appear at the time of the snapshot and then evaluates the appropriate static predicate. This architecture allows all the CGAL predicates, and thus all CGAL algorithms, to be used without changes.

Although the CGAL FILTERED_KERNEL can be used, filtering support is currently not optimal as an exact representation of the static snapshot of the kinetic primitive will be computed and then passed to the filtered predicate. We can remove this overhead by handling the filtering ourselves, i.e. use our own implementation of FILTERED_PREDICATE which first creates the snapshot using an interval number type and only creates the exact snapshot when needed.

The SIMULATOR can ask the kinetic data structure to audit itself when there is a gap between events, greatly easing debugging of kinetic data structures. For example the 2-dimensional kinetic Delaunay triangulation can do this by using the CGAL Delaunay_triangulation_2 class data structure to compute the current Delaunay triangulation (using the INSTANTANEOUSKERNEL, and then compare the resulting triangulation to the current kinetic triangulation.

4 Polynomial Kernel

A key operation is exact kinetic data structures is comparison of two roots of polynomials to determine which occurs first. This cannot be done directly using existing exact computation techniques, although the CORE [7] library has recently added the necessary support. Our polynomial kernels provide this functionality. The package is structured around several main concepts. They are

- FUNCTION: a univariate polynomial
- ROOTENUMERATOR: an object which can enumerate the roots of a polynomial between lower and upper bounds (possibly both infinite). We provide a number of models based on Descartes' rule of sign, Bézier curves and Sturm sequences, all of which support exact operations. In addition we provide a couple of numeric models and a wrapper for the CORE EXPR type.
- POLYNOMIALKERNEL: a kernel which ties together the ROOTENUMERATOR and useful predicates acting on polynomials and their roots. We provide filtered and unfiltered models.

Our basic representation of roots of polynomials for exact computation is a polynomial coupled with a an interval which isolates a unique root of the polynomial. This allows for fast comparisons since

- most comparisons simply require comparing the isolating intervals
- when difficult comparisons are performed (nearby roots), the size of the isolating intervals are reduced, making future comparisons easier
- root isolation is dealt with in a lazy manner. The focus is on roots nearer to the current time. Roots far ahead in the future are not isolated, which could save computation time if the corresponding events are not within our simulation time window or if the events get descheduled by the simulator before they reach the head of the event queue.

In addition to the above, we have implemented a technique based on Sturm sequences which allows us to test two isolated roots for equality.

We are currently extending our root representation to support field operations on algebraic numbers. Our technique uses resultants to generate a new polynomial having as a root the outcome of the field operation on the original algebraic numbers. To complete our representation, we also generate an appropriate isolating interval for this root. Finally, we plan to incorporate existing specialized techniques for han-

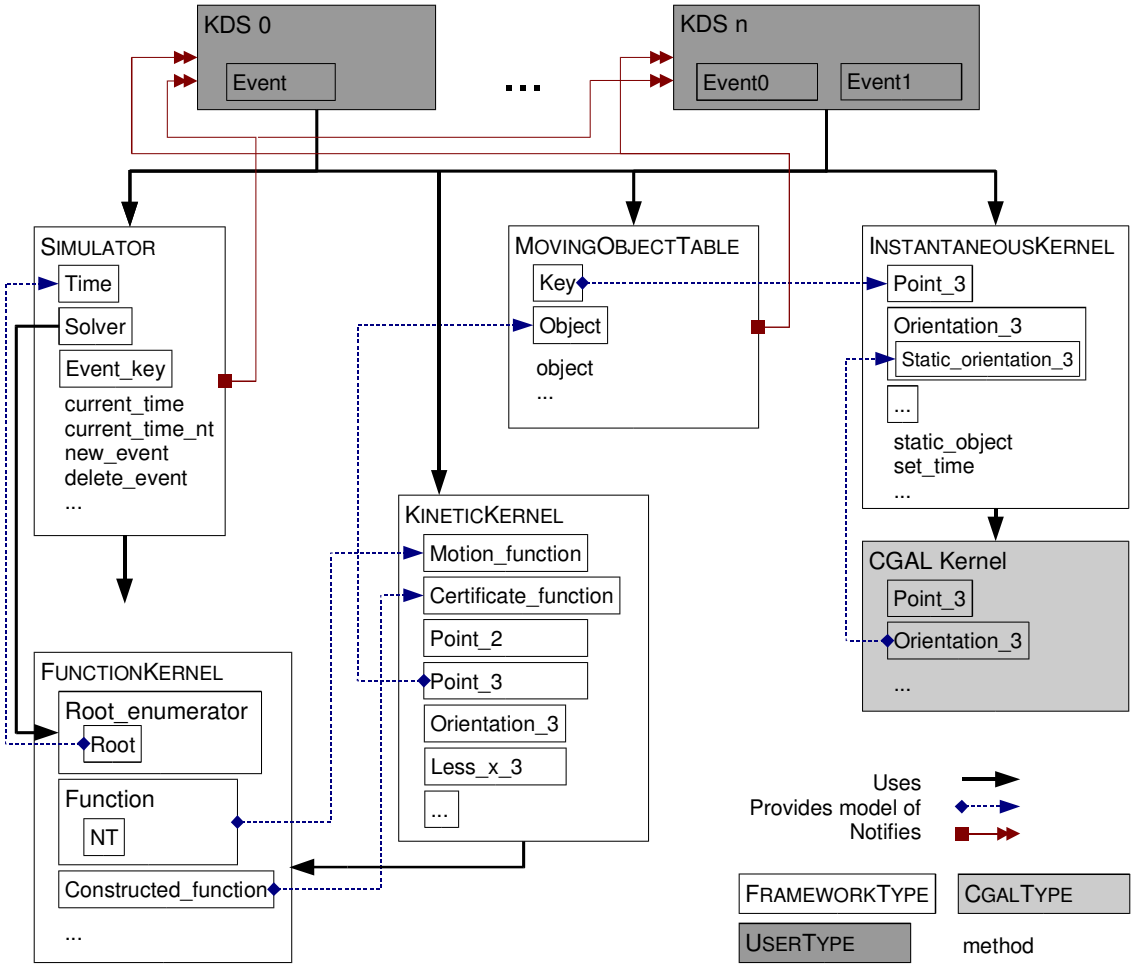

Figure 1: *Framework architecture:* Each large white box represents a main concept, the sub boxes their contained concepts, and regular text their methods. A "Uses" arrow means that a model of concept will generally use methods from (and therefore should take as a template parameter) the target of the arrow. A "Provides model of" arrow means that the source model provides an implementation of the destination concept through a typedef. Finally, a "Notifies" arrow means that the class notifies the other class of events using a standardized notification interface.

dling low degree polynomials [2, 3, 6].

References

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

[2] I.Z. Emiris and E.P. Tsigaridas. Comparison of fourth-degree algebraic numbers and applications to geometric predicates. Technical Report ECG-TR-302206-03, INRIA Sophia-Antipolis, 2003.

[3] I.Z. Emiris and E.P. Tsigaridas. Methods to compare real roots of polynomials of small degree. Technical Report ECG-TR-242200-01, INRIA Sophia-Antipolis, 2003.

[4] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.

[5] L. Guibas, M. Karaveles, and D. Russel. A computational framework for handling motion. In *ALENEX*, 2004.

[6] M.I. Karavelas and I.Z. Emiris. Root comparison techniques applied to the planar additively weighted Voronoi diagram. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 320–329, January 2003.

[7] C. Yap. A new number core for robust numerical and geometric libraries. In *Proc. 3rd CGC Workshop on Geometric Computing*, 1998. URL `http://www.cs.nyu.edu/exact/core/`.