A CGAL-based Univariate Algebraic Kernel and Application to Arrangements

Sylvain Lazard * Luis Peñaranda* Elias Tsigaridas †

Abstract

Solving univariate polynomials and multivariate polynomial systems is critical in geometric computing with curved objects. Moreover, the real roots need to be computed in a certified way in order to avoid possible inconsistency in geometric algorithms. We present a CGAL-based univariate algebraic kernel, which follows the CGAL specifications for univariate kernels. It provides certified real-root isolation of univariate polynomials with integer coefficients (based on the library RS) and standard functionalities such as basic arithmetic operations, gcd and square-free factorization.

We compare our implementation with that of other univariate algebraic kernels that follow the same CGAL specifications. In particular, we compare it to the one developed at MPII. We also apply this kernel to the computation of arrangements of univariate polynomial functions.

1 Introduction

Implementing geometric algorithms robustly is known to be a difficult task for two main reasons. First, all degenerate situations have to be handled and second, algorithms often assume a real-RAM model which is not realistic in practice. In recent years, the paradigm of exact geometric computing arose as a standard for robust implementations. In this paradigm, geometric decisions, such as "is a point inside, outside or on a circle?", are made exactly, usually using exact arithmetic combined with interval arithmetic for efficiency; on the other hand, geometric constructions, such as the coordinates of a point of intersection, may be approximated.

We address here one recurrent difficulty arising when implementing algorithms dealing with curved objects. Such algorithms usually require evaluating, manipulating and solving systems of polynomials equations and comparing their roots. One of the most critical parts of dealing with polynomials or polynomial systems is the isolation of the real roots and their comparison.

We restrict here our attention to the case of univariate polynomials and address this problem in the context of CGAL, a C++ Computational Geometry Algorithms Library, which is an open source project and became a standard for the implementation of geometric algorithms [3].

CGAL is designed in a modular fashion. Algorithms are typically parameterized by a *traits* class which encapsulates the geometric objects, predicates and constructions used by the algorithm. Typically, this allows implementing algorithms independently of the type of input objects. For instance, a sweep-line algorithm for computing arrangements can be implemented generically for segments or curves. Similarly, the model of computation, such as exact arbitrary-length integer arithmetic or approximate fixed-precision floating-point arithmetic are encapsulated in the concept of *kernel*. An implementation is thus typically separated in three layers, the geometric algorithm which relies on a traits class, which itself relies on a kernel for elementary operations. A choice of traits class and kernel gives freedom to the users and allows comparison.

We present here a kernel for solving and manipulating the real roots of univariate polynomials with integer coefficients which follows CGAL specifications [2]. In particular, this kernel performs the isolation and comparison of the real roots of such polynomials. The kernel also provides various operations on polynomials, such as gcd, which are central for manipulating algebraic numbers. We also present experimental results and compare our kernel with the one developed by Hemmer and Limbach [10].

2 Univariate algebraic kernel

We describe here our implementation of our univariate algebraic kernel. The two main requirements of the CGAL specifications, which we describe here, are the isolation of real roots and their comparison. We also describe our implementation of two important specific operations, greatest common divisor (gcd) computation and refinement of isolating intervals, that are needed, in particular, for comparing algebraic numbers.

Preliminaries. The kernel handles univariate polynomials and algebraic numbers. The polynomials have

*LORIA (INRIA, CNRS, Nancy Université) and INRIA Grand Est, Nancy, France. Firstname.Name(AT)loria.fr

†INRIA Méditerranée, Sophia-Antipolis, France. Elias.Tsigaridas(AT)inria.fr

integer coefficients and are represented by arrays of GMP arbitrary-length integers [9]. We implemented in the kernel the basic functions on polynomials, including basic arithmetics, evaluation, and input/output. An algebraic number that is a root of a polynomial F is represented by F and an isolating interval, that is an interval containing this root but no other. We implemented intervals using the MPFI library, which represents intervals with two MPFR arbitrary fixed-precision floating point numbers; note that MPFR is developed on top of the GMP library for multi-precision arithmetic [11] [12].

Root isolation. For isolating the real roots of univariate polynomials with integer coefficients, we developed an interface with the library RS [14]. This library is written in C and is based on Descartes' rule for isolating the real roots of univariate polynomials with integer coefficients.

We briefly detail here the general design of the RS library; see [13] for details. RS is based on an algorithm known as *interval Descartes* [4]; namely, the coefficients of the polynomials obtained by changes of variable, sending intervals $[a, b]$ onto $[0, +\infty]$, are only approximated using interval arithmetic when this is sufficient for determining their signs. Note that the order in which these transformations are performed in RS is important for memory consumption. The intervals and operations on them are handled by the MPFI library.

Another characteristic of RS is its memory management: it implements a *mark-and-sweep* garbage collector, which is well suited to RS needs.

Algebraic number comparison. As mentioned above, one of the main requirements of the CGAL algebraic kernel specifications is to compare two algebraic numbers r_1 and r_2. If we are lucky, their isolating intervals do not overlap and the comparison is straightforward. This is, of course, not always the case. If we knew that they were not equal, we could refine both isolating intervals until they do not overlap. See below for details on how we perform the refinements. Hence, the problem reduces to determining whether the algebraic numbers are equal or not.

To do so, we compute the gcd of the polynomials P_1 and P_2 associated to the algebraic numbers; see below for details on this operation. The roots of this gcd are the common roots of both polynomials. After calculating the gcd, we isolate its roots and refine the isolation intervals until each one of them overlap with exactly one root of P_1 and of P_2. If the isolating interval of r_1 and r_2 both overlap with the isolating interval of a root of the gcd, then $r_1 = r_2$. Otherwise they are not equal.

Gcd computations. Computing greatest common divisors between two polynomials is not a difficult task, however, it is not trivial to do so efficiently. Indeed, a naive implementation of the Euclidean algorithm works fine for small polynomials but the intermediate coefficients suffer an exponential grow in size, which is not manageable for medium to large size polynomials.

We thus implemented a *modular* gcd function, which calculates the gcd of polynomials modulo some prime numbers and reconstructs later the result with the help of the *Chinese remainder theorem*. Details on these algorithms can be found in [8]. Note modular gcd is always more efficient than regular gcd, especially when the two polynomials have no common roots.

Refining isolating intervals. As we mentioned before, refining the interval representing an algebraic number is critical for comparing such numbers. We have implemented two approaches for refinement.

Both approaches require that the polynomial associated to the algebraic number is square free. The first step thus consists of computing the square-free part of the polynomial. This is easily done by computing the gcd of the polynomial and its derivative.

Our first approach is a simple bisection algorithm. It consists in calculating the sign of the polynomial associated to the algebraic number at the endpoints and midpoint of the interval. Depending on those three signs we can take as isolating interval the left of right half of the previous one.[1]

The second approach we implemented is the *quadratic interval refinement* [1]. Roughly speaking, this method splits the interval in many parts and guesses in which one the root lies. If the guess is correct, the algorithm will divide, in the next refinement step the (chosen) interval in more parts and, if not, in less. Unfortunately, we can not always guarantee that we guess the correct interval at each step, so on average the algorithm turns out to be just a bit faster than the bisection one. Moreover, we have to implement its data structures very carefully in order to be efficient. In particular, this required the development of functions to handle dyadic numbers efficiently. Note that these functions are also useful in the bisection method when increasing the precision (because working directly with MPFR is rather tricky).

Currently, refinement function based on both approaches are present in our kernel and the user can choose the one best suited to her/his needs.

3 Benchmarks

In this section, we compare the running time for root isolation of our algebraic kernel with the one devel-

[1] Note that since the polynomial is square free the signs at the two endpoints of any isolating interval always differ. We thus do not need to compute the sign at both endpoints.

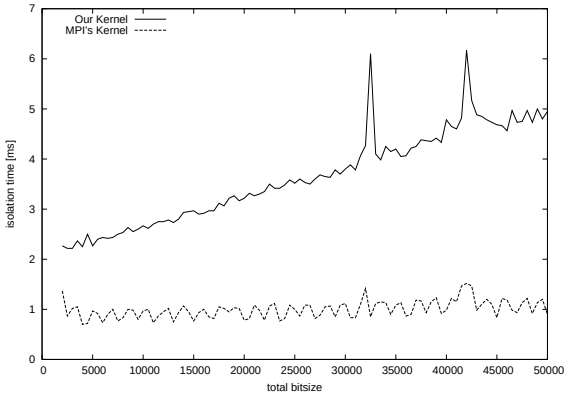

Figure 1: Degree 12 polynomials.

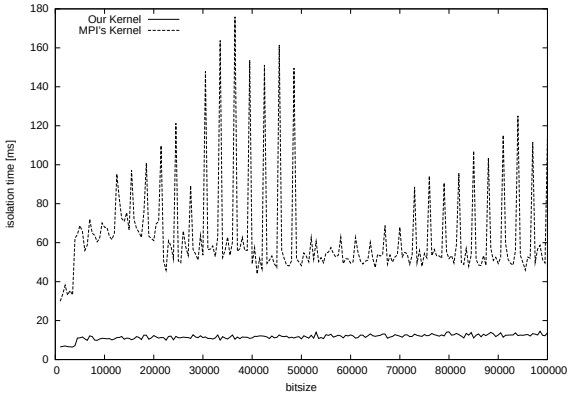

Figure 2: Polynomials of constant degree 100

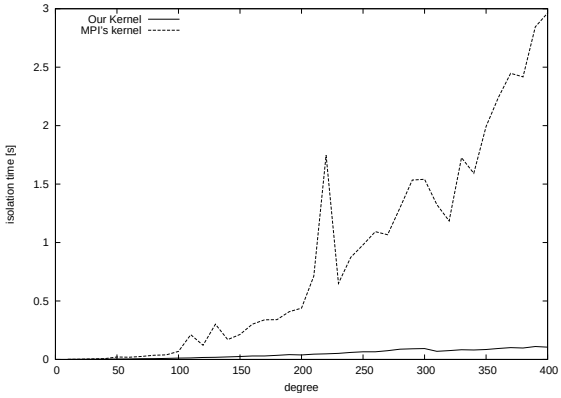

Figure 3: Polynomials of constant bitsize 20000

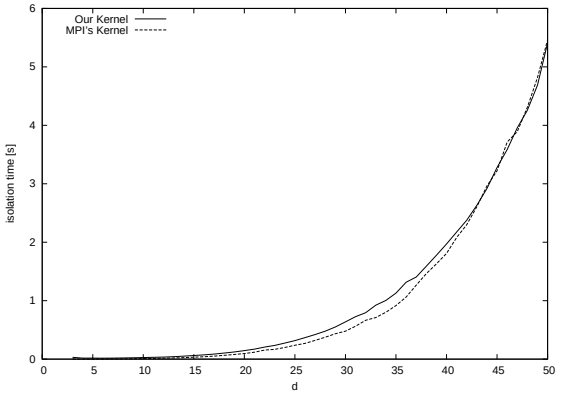

Figure 4: Mignotte polynomials

oped by Hemmer and Limbach [10]. Running times for root isolation is a very representative test because, as explained before, it involves other non-trivial operations on polynomials. For technical reasons, we could not compare our running time for comparing algebraic numbers with those of [10]. All tests were ran on a single-core 3.2 GHz Intel Pentium 4 with 2 Gb of RAM, using 64-bit Linux.

The first test sets comes from [10]. See Figure 1. It consists of polynomials of degree 12, each one being the product of six degree-two polynomials that have at least a root in the interval $[0, 1]$. The tests are averaged over 50 trials. The value represented on the x-axis is the total bitsize of all the coefficients of the input polynomial.

Secondly, we consider random polynomials with constant degree 100 and varying bitsize. See Figure 2. As before, the x-axis is the total bitsize of all the coefficients. The tests are averaged over 100 trials. We also consider random polynomials with constant bitsize 20000 but varying degrees. The results, shown in Figure 3 are averaged over 100 trials.

Finally, we tested Mignotte polynomials, that is nearly degenerate polynomials of the form $x^d - 2(kx - 1)^2$. The difficulty with solving these polynomials lies in the fact that two of their roots are very close to each other (the isolating intervals for these two roots are thus very small). For these tests, we used Mignotte polynomials with coefficients of bitsize 50, with varying degree d. Running time are shown in Figure 4: the tests are averaged over 5 trials.

4 Discussion

Figure 1 shows that our kernel's performance is worse than MPII's one for small degree polynomials. This difference comes from the fact that Rs, the most consuming part of our process, is conceived for handling high degree polynomials. This fact is confirmed by Figures 2 and 3, which show that for polynomials of larger degrees, our kernel runs faster.

We can also see in these two figures that the isolation time does not depend much on the bitsize of the input polynomials but mainly on the degree. This makes sense because of the considered algorithms for root isolation: *bitstream Descartes* [6] and *interval Descartes* [13] do not use, in most cases, all the bits of the coefficients. This should theoretically imply that, on random polynomials, the running time does not depend at all on the input bitsize. We however observe in Figure 1 that this is not quite the case for our kernel. This is presumably caused by the cost of copying the input polynomials to Rs memory space.

Despite this fact, we observe that the running time

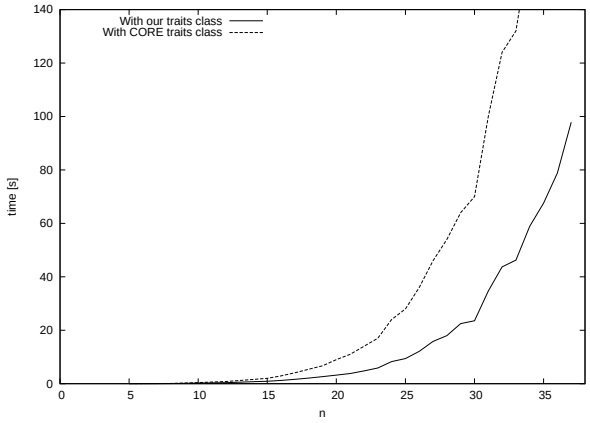

Figure 5: Arrangement calculation

of our implementation is very stable and does not depend too much on the bitsize or on the degree of the input polynomials. On the other hand, MPII's kernel does not either depend on the input bitsize. However it is much more dependent on the degree and is less stable.

Finally, we can mention that Figure 4 shows no significant difference between both kernels for isolating roots of Mignotte polynomials. MPII's kernel is barely faster for the smaller values of d, but that difference disappears when this value grows. However, this figure depicts the expected difficulty of solving Mignotte polynomials compared to random polynomials.

5 Arrangements

As an example of possible benefit of having efficient algebraic kernels in CGAL, we used our implementation to construct arrangements of polynomial functions. Wein and Fogel provided a CGAL package for calculating arrangements [15]. This package calculates the arrangements of general curves [7]. It is the user who must implement the data structures to store the curves and the primitive operations; requiring for example comparing positions of points, comparing the vertical order of curves at infinity and intersecting and splitting curves. All these functions must be grouped in a *traits class*, which is a transparent and convenient way to work with a package in CGAL. We implemented a traits class which uses the functions of our algebraic kernel and compared its performance with another traits classes which comes with CGAL's arrangement package and uses the CORE library [5].

To test the arrangement calculation, we generated n polynomials of degree $n-1$ with (n) coefficients of bitsize n. The running time for the construction of this type of arrangements is shown in Figure 5. We observe that we gain a factor of roughly two when using our kernel.

Acknowledgments

We would like to thanks M. Hemmer, E. Berberich, M. Kerber, and S. Limbach for fruitful discussion on the kernel developed at MPII and on the experiments.

References

[1] J. Abbott. *Quadratic interval refinement for real roots.* Poster in ISAAC, 2006.

[2] E. Berberich, M. Hemmer, M.I. Karavelas, and M. Teillaud. *Revision of the interface specification of algebraic kernel.* Technical Report, ACS-TR-243301-01, 2007.

[3] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org/`

[4] G.E. Collins, J.R. Johnson, and W. Krandick. *Interval arithmetic in Cylindrical Algebraic Decomposition.* Journal of Symbolic Computation, (2002) 34:145–157.

[5] CORE. `http://cs.nyu.edu/exact/`

[6] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt and N. Wolpert. *A Descartes algorithm for polynomials with bit-stream coefficients.* In Proc. 8th Int. Workshop on Computer Algebra in Scient. Comput. (CASC), LNCS. Springer, 2005.

[7] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein and N. Wolpert. *Effective Computational Geometry for Curves and Surfaces. Chapter 1, Arrangements.* J.-D. Boissonnat. and M. Teillaud (editors), Springer, 2006.

[8] K.O. Geddes, S.R. Czapor and G. Labahn. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.

[9] GMP. GNU multiple precision arithmetic library. `http://gmplib.org/`

[10] M. Hemmer and S. Limbach *Benchmarks on a generic univariate algebraic kernel* ACS Technical Report No. ACS-TR-243306-03, 2006.

[11] MPFI. Multiple precision interval arithmetic library. `http://perso.ens-lyon.fr/nathalie.revol/software.html`

[12] MPFR. Library for multiple-precision floating-point computations. `http://mpfr.org/`

[13] F. Rouillier and P. Zimmermann. *Efficient Isolation of Polynomial Real Roots.* Journal of Computational and Applied Mathematics, vol. 162 n. 1, p. 33-50, 2003.

[14] RS. Real roots of systems with a finite number of complex solutions. `http://fgbrs.lip6.fr/`

[15] R. Wein and E. Fogel *The new design of CGAL's arrangement package.* Technical report, Tel-Aviv University, 2005. `http://www.cs.tau.ac.il/~wein/publications/pdfs/Arr_new_design.pdf`