A Generic Algebraic Kernel for Non-linear Geometric Applications

Eric Berberich, Michael Hemmer, Michael Kerber

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Generic Algebraic Kernel for Non-linear Geometric Applications

Eric Berberich — Michael Hemmer — Michael Kerber

N° 7274

May 2010

_ Algorithms, Certification, and Cryptography _

ISSN 0249-6399 ISRN INRIA/RR--7274--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

A Generic Algebraic Kernel for Non-linear Geometric Applications

Eric Berberich * , Michael Hemmer † , Michael Kerber ‡

Theme : Algorithms, Certification, and Cryptography
Algorithmics, Programming, Software and Architecture
Équipes-Projets Geometrica

Rapport de recherche n° 7274 — May 2010 — 17 pages

Abstract: We report on a generic (uni- and bivariate) algebraic kernel that becomes available to the public with CGAL 3.7. It comprises complete, correct, though efficient state-of-the-art implementations on polynomials, roots of polynomial systems, and the support to analyze algebraic curves defined by bivariate polynomials. The kernel is accompanied with a ready-to-use interface to enable arrangements induced by algebraic curves, that have already been used as basis for various geometric applications, as arrangements on Dupin cyclides or the triangulation of algebraic surfaces. We present two novel applications: arrangements of rotated algebraic curves and Boolean set operations on polygons bounded by segments of algebraic curves. We also provide exhaustive experiments showing that our implementation is competitive and often outperforms existing implementation on non-linear curves available in CGAL, which demonstrates the general usefulness of the presented software.

Key-words: Algebraic Kernel, Algebraic Curves, Arrangements, Boolean Set Operations, Robust Geometric Computing, Experiments, CGAL, Generic Programming

* Max-Planck-Institut für Informatik, Saarbrücken, Germany; eric@mpi-inf.mpg.de
† Geometrica, INRIA Sophia Antipolis, France; Michael.Hemmer@sophia.inria.fr
‡ Institute of Science and Technology, Klosterneuburg, Austria; mkerber@ist.ac.at

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Un noyau algébrique générique pour des applications géométriques non linéaires

Résumé : Nous décrivons un noyau algébrique générique univarié et bivarié, qui est distribué publiquement dans CGAL 3.7. Il contient des implantations complètes correctes et efficaces sur les polynômes et les racines de systèmes polynomiaux, ainsi que des fonctionnalités pour analyser des courbes algébriques définies par des polynônes bivariés. Le noyau est accompagné d'une interface pour calculer des arrangements induits par des courbes algébriques, qui ont déjà été utilisés comme base pour des calculs géométriques variées, telles que les arrangements de cyclides de Dupin ou les triangulations de surfaces algébriques. Nous présentons deux applications nouvelles : les arrangements de courbes transformés de courbes algébriques par certaines rotations, et les opérations booléennes sur des 'polygones' définis par des arcs de courbes algébriques. Nous fournissons également des résultats expérimentaux exhaustifs montrant que notre implantation est compétitive et surpasse souvent les implantations existantes dans CGAL sur des courbes. Ceci montre l'utilité du code présenté.

Mots-clés : noyau algébrique, courbe algébrique, arrangement, opérations booléennes, calcul géométrique robuste, expériences, CGAL, programmation générique

1 Introduction

A considerable amount of work in Computational Geometry is motivated by the fact that many geometric algorithm, for instance, used in Computer Aided Design systems, are actually not robust. Often, this is caused by the use of fast but inexact floating point arithmetic, which can lead to wrong (and inconsistent) decisions within algorithms [28]. On the other hand, Computer Algebra has developed very general and exact tools that could solve such problems in principle, but a naive application of these tools is by far too slow, especially when applied in geometric settings. Thus, our cardinal research interest is to incorporate methods from all these areas in order to design and implement geometric algorithms that are exact, complete, and efficient [30].

In this context CGAL [19], the Computational Geometry Algorithms Library, was started in 1996 with its first beta release in June 1997. Since that time, CGAL can be considered as the state-of-the-art in implementing geometric algorithms. While the major focus of the library had been on linear geometry, the project also raised its attention towards non-linear geometry, see, for instance, [20, 21, 26, 18]. Evidently, all these examples tackle problems whose solution require solving polynomial systems, efficient comparison and approximation of algebraic numbers, etc. This led to the demand [17],[29, §13] for an *Algebraic Kernel* providing an interchangeable black-box implementation of state-of-the-art algorithms for the above mentioned functionalities [29, §8],[5].

We contribute the first open-source implementation of an algebraic kernel for polynomials in one and two variables, becoming publicly available with CGAL 3.7; see Section 2. Our kernel is parameterized in the coefficient type of the algebraic curves, and thus generically supports various number types, even beyond integers and rationals. It correctly computes and handles solutions of univariate and bivariate polynomial systems of any degree including all sorts of degenerate cases. Internally, it comprises several recent algorithmic results in real root isolation and topology computations of algebraic curves[1] [15] and pairs of such [14] to achieve efficiency. Its prototypical version has already been an essential building block for numerous geometric applications. While its univariate part was used to exactly handle parameterized curves defined over extension fields of degree 2 [12], the bivariate part enabled computing arrangements of algebraic curves in the plane (see next paragraph), computing arrangements on quadrics [4] and on ring Dupin cyclides [6], triangulating algebraic surfaces of arbitrary degree [7], and, most recently, computing Voronoi diagrams for lines in space [23].

Our second main contribution is a mature and ready-to-use support for arbitrary algebraic curves, or segments of such, in the arrangement[2] package of CGAL 3.7; see Section 3. This is the most general class of objects for which arrangement computation is currently available. The underlying algorithm has been exposed in [14], and a preliminary implementation has been described therein. Since then, the software has undergone a continuous maturation process: the implementation properly distinguishes between an algebraic layer (provided by the algebraic kernel) and a geometric layer that provides the geometric

[1] An algebraic curve is defined by $C := \{(x, y) \in \mathbb{R}^2 \mid f(x, y) = 0\}$ for a polynomial f in two variables, called the *defining polynomial* of C.

[2] The *arrangement* $\mathcal{A}(\mathcal{C})$ is the subdivision of the plane into 0-, 1-, and 2-dimensional cells induced by a set of curves $\mathcal{C}$.

primitives. Furthermore, it is now possible to define curves over algebraic extension fields; we demonstrate how this allows to compute arrangements of rotated algebraic curves with our software. Finally, we newly enabled Boolean set operations for shapes bounded by segments of arbitrary algebraic curves.

Our new software subsumes all previously available CGAL implementations on curved objects, which are arrangements of circles, conics, rational functions, and Bézier curves [29, §30], [20]. As a final contribution, we experimentally compare our general implementation to all these dedicated solutions; see Section 4: Our software generally outperforms implementations for conics and rational functions, is only a constant factor worse than the dedicated classes for circles and is comparably fast for the case of cubic Bézier curves. We take these results as a proof for the maturation of our traits class as well as of the underlying algebraic kernel, and for the general usefulness of the provided software.

2 The Algebraic Kernel Package

CGAL follows the *generic programming paradigm*, that is, algorithms are formulated and implemented such that they abstract from the actual types, constructions, and predicates. Using the C++ programming language this is realized by means of class and function templates, respectively. The paradigm is applied to all layers: Lower layers allow to employ different number types; one example is the algebraic kernel that we present in this section. Higher levels are written such that every algorithm and data structure is parameterized by a so-called *traits* class, which provides the bundle of types, constructions and predicates that is required by a particular algorithm or data structure. Section 3 discusses the arrangement package of CGAL, which supports many types of planar curves using this technique – the user just needs to provide an appropriate traits class.

The interface of the algebraic kernel [29, §8] is subdivided into two parts that are concerned with univariate polynomials and bivariate polynomial systems, respectively. According to this structure, we provide two classes,[3] a *univariate* and *bivariate* kernel. Following the generic programming paradigm, both kernels are class templates that allow the user to select his preferred coefficient type. We consider it as a major achievement of our generic design that a considerable collection of number types is supported, namely several exact types for integers or rationals as they are provided by the libraries GMP, CORE, or LEDA, and also types that represent algebraic extensions fields (as demonstrated in Section 3.2).

Both kernels basically consist of three major blocks: (a) a support for polynomials covering fundamental methods such as gcd computation or square free factorization; (b) a solver for real solutions of polynomial systems; (c) a proper handling of these solutions such as their certified approximation, exact comparison, and sign evaluation of polynomials at these solutions.

Polynomial Support: This is available as a separate package [29, §7]. The computation of the gcd and the square free factorization heavily utilizes modular arithmetic. Though generic in the actual coefficient type, we achieve competitive running times to other implementations in dedicated libraries such as the NTL [22]. The computation of subresultants and Sturm-Habicht sequences [2]

[3] `CGAL::Algebraic_kernel_1< Coeff >` and `CGAL::Algebraic_kernel_2< Coeff >`

is also provided; these operations constitute a major ingredient in our bivariate kernel. We implemented the algorithm by Ducos [11] for computing subresultants. We next describe the other two building blocks separately for the uni- and bivariate kernel:

Algebraic_kernel_1: Our univariate kernel is flexible in its choice of the employed root solver. Currently, there are two available solvers which are both based on Descartes' rule of signs. The first is a generic implementation of the algorithm by Collins and Akritas [9], the second is a variant that considers each coefficient as a bitstream [13]. The later, which is the default, is particularly useful in case of algebraic coefficients but has also proven to be competitive [24] to other state-of-the-art solvers that are dedicated to integral polynomial.

A solution, a *real algebraic number*, is represented as real root of a square free polynomial where an isolating interval uniquely defines the root. An approximation of a solution, with respect to any requested absolute or relative error, is provided by refining the interval. The implemented refinement method is a slight modification of the one presented in [1], which has quadratic convergence.

Algebraic_kernel_2: The bivariate kernel is based on an algorithm computing a geometry-enhanced topological analysis of a single curve [15] and of a pair of curves [14] (both analyses essentially are special cases of a cylindrical algebraic decomposition [8]). The main idea behind both analyses is to compute the critical x-coordinates of curves and curve pairs by projection (resultants), and compute additional information about the critical fibers using subresultants and Sturm-Habicht sequences. With that information, the fiber at critical x-coordinates is computed by a variant of the bitstream Descartes method; see [27] for a comprehensive description of these techniques.

The two described analysis methods are the underpinnings of most of the kernel methods: For instance, to find the solutions of a system of two bivariate equations, a curve pair analysis is triggered, and the critical fibers are traversed subsequently, collecting all intersection points on the way. Such a solution point is stored to be the i-th fiber point of one of the two curves at the corresponding x-coordinate (for some i). This representation allows to approximate the coordinates of solutions efficiently (by increasing the precision to the corresponding fiber in the curve analysis), but it also permits exact operations.

As an example, we discuss the computation of the sign of a bivariate polynomial f at an algebraic point p, where p is represented as the fiber point of some other curve with defining equation g. We first check whether $f(p) = 0$. For that, we consider the curve pair analysis of the curves defined by f and g and check whether p is an intersection. If not, we can simply evaluate f at more and more precise approximations of p using interval arithmetic, until we can certify the (non-zero) sign of $f(p)$. We remark that a lot of special cases are left out in this discussion for brevity (e.g., overlapping curves, curves with vertical components), but they are completely handled in our software.

The implementation of our bivariate kernel was guided by the idea of implementing geometric applications, as we present in Section 3. In particular, we assume that several function calls arise for a certain curve or curve pairs. Because of that, our bivariate kernel stores all computed analyses in a cache to avoid costly recomputations. Indeed, for such an analysis, a worst-case com-

plexity of $O(n^{10}(n+\tau)^2)$ has been shown, where n is the degree of the curve, and τ is the maximal bitsize of its coefficients [27].

3 Geometric Applications

In this section we focus on new geometric applications that are build on top of our bivariate algebraic kernel. Further applications (that used experimental versions of the kernel) have been discussed in previous work [12, 4, 6, 7, 23].

3.1 Arrangements of Algebraic Curves

Arrangements are fundamental structures ubiquitous in computational geometry. A modular and efficient realization of two-dimensional arrangements is available in CGAL [29, §30]. Arrangements with arbitrary degeneracies can be constructed with the help of a sweep-line algorithm or via incremental insertion. The main class is parameterized by a *geometric-traits class*[4] that provides the low-level support for the intended family of curves. This includes definitions of suitable data types for points and (possibly curved) segments, as well as geometric primitives to manipulate and query them, for instance, comparing two points lexicographically, or aligning two segments on the right of a common intersection point; see [29, §30] for a complete list of requirements.

Through our implementation of this machinery, CGAL 3.7 will contain an *algebraic traits*[5] that serves the needs for computing arrangements of algebraic curves of arbitrary degree, and segments[6] of it. All required geometric types, predicates, and constructions are established with the help of a mediating layer [3] that translates the geometric primitives into the language of the algebraic kernel, as previously exposed in [20, 14]. Algebraic curves must be specified in implicit form, that is, by their defining polynomial. The coefficient type of the polynomial is given as a template argument to our new traits class. Profiting from the generic design, we are able to support the same collection of number types as for the algebraic kernel (see Section 2); a result of this capability is exemplified in Section 3.2. Moreover, our new traits defines a user-friendly interface to construct points and segments. We refer to the manual of CGAL 3.7 for details.

3.2 Rotations of Algebraic Curves

For a curve C with defining polynomial $f \in \mathbb{Z}[x,y]$ and some angle α, let C' be the curve arising from rotating C by α counterclockwise around the origin. The defining polynomial of C' is given by

$$f'(x,y) \quad = \quad f(\cos(\alpha)x + \sin(\alpha)y, -\sin(\alpha)x + \cos(\alpha)y).$$

For certain choices of α, $\sin(\alpha)$ and $\cos(\alpha)$ can be represented by square root expressions. This is possible if and only if the angle is constructible with compass and straightedge. The following well-known result characterizes all possibilities.

[4] `CGAL::Arrangement_2< GeoTraits, ... >`

[5] `CGAL::Arr_algebraic_segment_traits_2< Coeff >`

[6] a *segment* of an algebraic curves C is an x-monotone path of on C not passing a singular point of C

Theorem 1 (Gauss) *An angle α is constructible with compass and straightedge if and only if $\alpha = c \cdot \frac{360}{2^k p_1 \cdots p_s}$ where $c, k, s \in \mathbb{N}$ and $p_1, \ldots, p_s$ are distinct Fermat primes, that is, primes of the form $2^{2^m} + 1, m \geq 0$.*

In the following, we discuss the example of $\alpha = 30°$. In this case, $\sin(\alpha) = \frac{1}{2}$ and $\cos(\alpha) = \frac{\sqrt{3}}{2}$. That means, when rotating a curve with integer coefficients by α, the rotated curve has coefficients from the domain $\mathbb{Z}_{\sqrt{3}} := \{a + b\sqrt{3} \mid a, b \in \mathbb{Z}\}$ (after clearing denominators); the same holds for every integral multiple of α. The CGAL number type for square-root extensions[7] is used to model $\mathbb{Z}_{\sqrt{3}}$. Our algebraic traits can be instantiated using this type. This enables the computations of arrangements of algebraic curves that are rotated by multiples of α, as shown on the right.

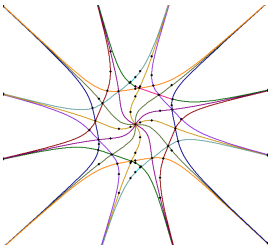


Figure 1: An arrangement of a degree 8 curve rotated by multiples of 30°.

We have also considered all other integer angles which are constructible with compass and straightedge, namely $45°, 18°, 15°, 9°, 6°$, and $3°$, and multiples of those. Choosing α to be a small angle allows finer rotations of the curve, but leads to a performance penalty due to the more complicated coefficient domain one has to deal with. We refer to [27, §5.2] for a more extensive treatment, that also compares the approach to an alternative that rotates by approximate angles.

3.3 Boolean Set Operations

We present a straight-forward implementation of Boolean set operations on shapes bounded by algebraic segments, which can be used to represent *semi-algebraic sets*. Such a set is defined by a finite number of algebraic inequalities. For example, a convex polygon with k edges is given by the point set that satisfies $a_i x + b_i x + c_i \geq 0$, for $1 \leq i \leq k$, $a_i, b_i, c_i \in \mathbb{R}$, $\neg(a_i = 0 \wedge b_i = 0)$.

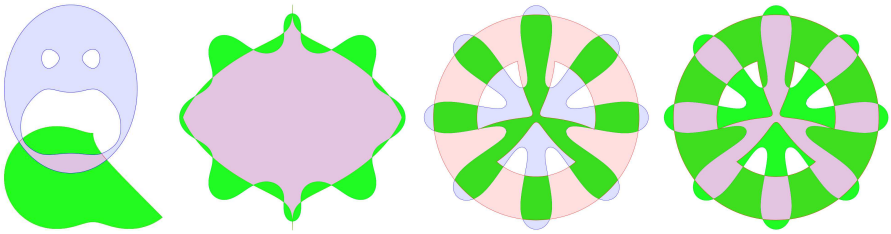
Their implementation is immediate using the possibility to extend the cells of an arrangement with data and to overlay two such arrangements in CGAL [29, §30]: Each vertex, each edge, and each face of an arrangement is enhanced by a Boolean flag that indicates whether that cell should be contained in a set. The complement of a set is simply computed by inverting all flags, while a binary operation – as union, intersection, or (symmetric) difference – is obtained by overlaying two such enhanced arrangements and updating the resulting flag according to the operation. A final removal of redundant vertices and edges simplifies the overall structure; see [10, §2] for details. Thus, we achieve arbitrary semi-algebraic sets, for instance, sets with (partially) open boundaries or with low-dimensional features as antennas and isolated vertices.

However, practical settings often desire shapes to be closed and free of low-dimensional features. A *regularized* Boolean set operation, given by $P \text{ op}^\star Q = \text{closure}(\text{interior}(P \text{ op } Q))$, ensures this property. CGAL's corresponding package [29, §19] expects as input general polygons with holes,[8] which are finite shapes (not necessarily contractible), bounded by finitely many x-monotone segments which do not intersect each other in their interior. All internal representations and computations are based on planar arrangements. Thus, to enable

[7] `CGAL::Sqrt_extension< Integer, Integer >`, where `Integer` must model $\mathbb{Z}$

[8] `CGAL::General_polygon_with_holes_2`

algebraic polygons we use the algebraic traits presented in Section 3.1. To illustrate Boolean set operations on such polygons we extended CGAL's corresponding demo (that previously only supported polygons bounded by line segments, circular arcs, and Bézier curves); see Figure 2. Visualization is established by a certified renderer for algebraic segments [16].



(a) Difference (b) Sym. Difference (c) Intersection (d) Sym. Difference

Figure 2: Boolean set operations on algebraic polygons (red and blue) each defined by a bounded face of a curve's arrangement. Degree of curves: (a) blue: 10, red: 7 (b) blue: 16, red: 14, (c+d) blue: 16, red: union of two degree 4 curves

4 Experimental Comparisons

Among the traits classes that are currently existing for CGAL's arrangement package, our algebraic traits is clearly the most general one. In fact, all previously available implementation constitute special cases and can be handled by our traits, too. This section concentrates on the performance of the specialized and dedicated traits classes to our general version; we refer to [27] for further comparisons with alternative (non-CGAL) approaches.

All experiments have been executed on a 2.40GHz 32-bit Intel(R) Core(TM)2 Duo CPU P9400 with 6144 KB cache and 4GB RAM memory, running Fedora 10 (Cambridge). All programs where compiled using `g++` version 4.3.2 optimized with `-O3` and `-DNDEBUG`. For brevity, we only present running time for a few instances. Appendix **??** shows the complete tables listing every tested instance.

Circles: The simplest class of curved objects are circles and circular arcs. The circle with center (a, b) and radius r is given as the vanishing set of $(x-a)^2+(y-b)^2-r^2$. For rational a, b, and r, CGAL's *circular traits*[9] enables arrangement computation of circular arcs; it only needs to handle algebraic numbers of degree up to 2 (i.e., square-root expressions) because the coordinates of an intersection point of two circles can not be of higher algebraic degree.

To compare with our approach, we created instances by creating m random point triples $(a_1, b_1, r_1), \ldots, (a_m, b_m, r_m) \in \{1, \ldots, m\}^3$ for fixed m. Each triple represents a circle with center (a_i, b_i) and radius r_i. We observe from Table 1 that our implementation is roughly a factor of 4 slower than the specialized CGAL class, and the factor decreases for increasing m. We emphasize that our traits class is free of any special treatment for circles (and the same is true

[9] We used `CGAL::Arr_circle_segments_traits_2`, as it appeared faster than the feature-identical class `CGAL::Arr_circular_arc_traits_2` on the tested instances

#Circles	#(V,E,F)	circular	**algebraic**	speedup	μs
200	(24268, 48142, 23877)	4.14	20.89	0.19	861
400	(85081, 169374, 84295)	15.69	71.96	0.21	845
600	(192640, 384084, 191447)	36.86	162.00	0.22	840
800	(338786, 675986, 337202)	67.45	284.72	0.23	840
1000	(548041, 1094088, 546050)	110.64	460.84	0.24	840

Table 1: Results for m circles with center on an $m \times m$ grid and integer radius up to m. The last column states the time per vertex in μs.

#Ellipses	#(V,E,F)	conic	**algebraic**	speedup	μs
30	(538, 1016, 480)	13.37	0.72	18.52	1341
60	(2200, 4280, 2082)	58.13	2.71	21.43	1232
90	(4656, 9132, 4478)	124.64	5.52	22.57	1186
200	(23744, 47088, 23346)	648.71	27.01	24.01	1137
400	(96502, 192204, 95704)	2660.99	108.94	24.42	1128
600	(214358, 427516, 213160)	5899.76	244.41	24.13	1140

Table 2: Results for m random ellipses. The last column states the time per vertex in μs.

for any other special case tested in this section) and handles all input with the same full "algebraic machinery" as for curves of higher degree. Regarding this, we consider a factor of 4 to be an appreciably small overhead. Also, we observe that the running time is roughly linear in the complexity of the returned arrangement.

Conics: A *conic* is an algebraic curve of degree 2. Their general equation is $ax^2 + by^2 = cxy + dx + ey + f = 0$, thus a conic can be presented by a six-tuple $(a, \ldots, f) \in \mathbb{Z}^6$. Although the class of conics contains several non-trivial curves like ellipses, parabolas and hyperbolas, one can still exploit many simplifying properties compared to the curves of arbitrary degree. For instance, no conic has a singular point, except for the special case of two intersecting lines. CGAL's *conic traits*[10] supports bounded segments on conic curves; for all internal algebraic computations, it relies on an exact type for real algebraic numbers. As recommended, we have tested the traits class instantiated with the types provided by CORE [25].

We compared both implementations for the case of ellipses. For that, we generated random ellipses with 30-bit coefficients.[11] The results presented in Table 2 show that our implementation is faster by a magnitude, and the ratio even increases when computing with more ellipses (although the ratio seems to stabilize at a factor of about 24). We conclude that our implementation gives faster results for all realistic inputs. This shows that our algebraic kernel handles the involved operations with algebraic numbers more efficiently than CORE.

[10] `CGAL::Arr_conic_traits_2`

[11] More precisely, we generated random conics with 30-bit coefficients and repeated the construction if the result was not an ellipse

n, m	#(V,E,F)	rational	**algebraic**	speedup	μs
6,10	(114, 254, 141)	2.05	0.25	8.15	2210
6,40	(2074, 4266, 2193)	35.86	3.64	9.85	1755
6,70	(7020, 14238, 7219)	124.78	11.71	10.65	1668
6,100	(13124, 26534, 13411)	223.14	22.82	9.77	1739
3,20	(422, 890, 469)	1.67	0.46	3.61	1096
7,20	(594, 1242, 649)	16.93	1.13	14.90	1912
11,20	(616, 1292, 677)	47.23	1.95	24.16	3173
15,20	(676, 1418, 743)	131.54	3.09	42.45	4583

Table 3: Results for m random rational functions generated by degree n polynomials with 16 bit coefficients. The last column states the time per vertex in μs.

Rational Functions: There is also specialized *traits for rational functions.*[12] For univariate polynomials P and Q, a rational function is defined by $y = P(x)/Q(x)$, or equivalently $Q(x)y = P(x)$. The degrees of P and Q can be chosen arbitrary large; however, the restriction of the y-degree of the defining equation to 1 of course drastically simplifies the realization of the involved primitives. Similar to the conic case, the specialized traits class uses an external number type for algebraic computations (taken from CORE by default).

For comparison, we generated m pairs of polynomials (P, Q), each representing a rational function. The degree of both P and Q was chosen as n, and each coefficient was chosen randomly with 16 bits. We observe in the first part of Table 3 that for fixed n our approach is faster by a roughly constant factor for increasing m. As the second part shows, the gap becomes more significant if we increase n. Again, the reason for this behavior lies in the more efficient handling of the underlying algebraic computations by our algebraic kernel.

Bézier Curves: Finally, CGAL provides support for arrangements of Bézier curves via a *Bézier traits.*[13] A Bézier curve is defined by a sequence of *control points* $p_0, \ldots, p_n$ as the image of the function

$$B : [0,1] \mapsto \mathbb{R}^2, (x,y) \to \sum_{k=0}^{n} p_k \binom{n}{k} t^k (1-t)^{n-k}.$$

Bézier curves are a widely used class of parameterized curves, with applications in graphics and computer-aided design. As for conics and rational functions, CGAL's traits class relies on an external algebraic number type; however, several filter techniques are implemented in order to avoid such computations as much as possible [21].

We compared this Bézier traits class with our class on the most important case of Bézier curves, namely cubic ones with 4 control points. We tested both implementations on instances with random control points, and on degenerate instances where we forced all Bézier curves to pass certain points. The results were roughly similar for both cases and so, Table 4 only lists the random case. In general, both traits classes behave equally efficiently in practice, with minimal

[12] `CGAL::Arr_rational_arc_traits_2`
[13] `CGAL::Arr_Bezier_curve_traits_2`

m	#(V,E,F)	Bézier	**algebraic**	speedup	μs
20	(259, 417, 160)	0.99	1.38	0.71	5346
40	(732, 1269, 539)	3.84	4.14	0.92	5661
60	(1863, 3439, 1578)	10.10	10.38	0.97	5571
80	(2849, 5293, 2446)	19.00	16.23	1.17	5697
100	(4513, 8581, 4070)	29.02	26.87	1.07	5955

Table 4: Results for m Bézier curves. with 4 randomly chosen control points. The last column states the time per vertex in μs.

advantages for our software for an increasing number of curves. We remark that we observed several robustness issues concerning CGAL's Bézier traits class: It is not able to handle overlapping curves and crashes frequently for Bézier curves with self-intersections.[14] Also, we observed significant slow-downs for some instances (possibly due to filter failures), and even a crash on one tested example. On the other hand, the Bézier traits shows its strength for higher degree curves, where it outperforms our approach in many examples thanks to its filter techniques.

5 Conclusions

Our new software introduces full support for algebraic curves in CGAL, achieving the goals of robustness, usability, and efficiency at the same time. Our algebraic kernel provides a fundamental layer for certified computation with curved objects. The design also shows flexibility in order to ease the integration of improved solutions for sub-algorithms. On top of our kernel, various geometric applications in $2D$ and $3D$ have been developed – this work has presented two novel ones. We take the considerable amount of derived applications as a proof of concept of our software, and as a sufficient argument to make it publicly available. Finally, our experiments clearly show the maturation of our implementation regarding performance. Some existing applications in CGAL can directly profit from using our new algebraic traits class; as an example, we mention the computation of exact offset of polygons [29, §24], which currently use the conic traits class to represent the offset's boundary. In the long term, our contribution might also lead to a re-design of the specialized traits classes, so that they make use of the algebraic kernel.

Acknowledgments: The authors thank Pavel Emeliyanenko for supporting work on the implementation and Monique Teillaud for her translation of the abstract to French.

References

[1] J. Abbott. Quadratic interval refinement for real roots. Poster presented at the 2006 Int. Symp. on Symb. and Alg. Comp. (ISSAC 2006).

[2] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2nd edition, 2006.

[14] All of out tested instances were free of self-intersections.

[3] E. Berberich and P. Emeliyanenko. CGAL's Curved Kernel via Analysis. Technical Report ACS-TR-123203-04, Algorithms for Complex Shapes, 2008.

[4] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. of the 15th Ann. Europ, Symp. on Algorithms (ESA 2007)*, volume 4698 of *LNCS*, pages 645–656, Springer, 2007.

[5] E. Berberich, M. Hemmer, M. I. Karavelas, and M. Teillaud. Revision of the interface specification of algebraic kernel. Technical Report ACS-TR-243301-01, Algorithms for Complex Shapes, 2007.

[6] E. Berberich and M. Kerber. Exact arrangements on tori and Dupin cyclides. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, pages 59–66, ACM, 2008.

[7] E. Berberich, M. Kerber, and M. Sagraloff. An efficient algorithm for the stratification and triangulation of algebraic surfaces. *Computational Geometry: Theory and Applications*, 43:257–278, 2010.

[8] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. of the 2nd GI Conference on Automata Theory and Formal Languages*, volume 6, pages 134–183, LNCS, Springer, Berlin, 1975.

[9] G. E. Collins and A. G. Akritas. Polynomial real root isolation using descarte's rule of signs. In *Proc. 3^{rd} ACM Symp. on Symbolic and Algebraic Comp.*, pages 272–275, 1976.

[10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Germany, 2nd edition, 2000.

[11] L. Ducos. Optimizations of the subresultant algorithm. *Journal of Pure and Applied Algebra*, 145:149–163, 2000.

[12] L. Dupont, M. Hemmer, S. Petitjean, and E. Schömer. Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In *Proc. of the 15th Ann. Europ. Symp. on Algorithms (ESA 2007)*, volume 4698 of *LNCS*, pages 633–644, Springer, 2007.

[13] A. Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs*. PhD thesis, Saarland University, Germany, 2008.

[14] A. Eigenwillig and M. Kerber. Exact and efficient 2d-arrangements of arbitrary algebraic curves. In *Proc. of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 122–131, 2008.

[15] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and exact geometric analysis of real algebraic plane curves. In *Proc. of the 2007 international symposium on Symbolic and algebraic computation (ISSAC'07)*, pages 151–158, ACM, 2007.

[16] P. Emeliyanenko, E. Berberich, and M. Sagraloff. Visualizing arcs of implicit algebraic curves, exactly and fast. In *Advances in Visual Computing : 5th International Symposium, ISVC 2009*, volume 5875 of *LNCS*, pages 608–619, Springer, 2009.

[17] I. Z. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards an open curved kernel. In *Proc. of the 20th Annual Symposium on Comp. Geom. (SCG'04)*, pages 438–446, 2004.

[18] I. Z. Emiris and G. M. Tzoumas. Exact and efficient evaluation of the incircle predicate for parametric ellipses and smooth convex objects. *Comput. Aided Des.*, 40(6):691–700, 2008.

[19] A. Fabri, G. J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, the computational geometry algorithms library. *Softw. – Pract. and Exp.*, 30(11):1167–1202, 2000.

[20] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chapter 1, pages 1–66. 2006.

[21] I. Hanniel and R. Wein. An exact, complete and efficient computation of arrangements of bézier curves. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling (SPM 2007)*, pages 253–263, ACM, 2007.

[22] M. Hemmer and D. Hülse. Generic implementation of a modular gcd over algebraic extension fields. In *25th European Workshop on Computational Geometry*, page 4, Université Libre de Bruxelles, 2009.

[23] M. Hemmer, O. Setter, and D. Halperin. Constructing the exact voronoi diagram of arbitrary lines in space. Submitted to ESA 2010.

[24] M. Hemmer, E. P. Tsigaridas, Z. Zafeirakopoulos, I. Z. Emiris, M. I. Karavelas, and B. Mourrain. Experimental evaluation and cross-benchmarking of univariate real solvers. In *SNC '09: Proceedings of the 2009 conference on Symbolic numeric computation*, pages

45–54, ACM, 2009.

[25] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proc. of the 15th Annual ACM Symposium on Comp. Geom. (SCG'99)*, pages 351–359, 1999.

[26] M. I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *Proc. Internat. Symp. on Voronoi diagrams in Science and Engineering (VD2004)*, pages 51–62, 2004.

[27] M. Kerber. *Geometric Algorithms for Algebraic Curves and Surfaces.* PhD thesis, Saarland University, Germany, 2009.

[28] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom. Theory Appl.*, 40:61–78, 2008.

[29] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 3.6 edition, 2010. http //www.cgal.org/Manual/3.6/doc_html/cgal_manual/packages.html.

[30] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. CRC Press, 2004.

#Circles	#(V,E,F)	circular	**algebraic**	speedup	μs
100	(5459, 10726, 5269)	0.86	4.94	0.17	906
200	(24268, 48142, 23877)	4.14	20.89	0.19	861
300	(52594, 104597, 52005)	9.33	44.52	0.20	846
400	(85081, 169374, 84295)	15.69	71.96	0.21	845
500	(134984, 268973, 133991)	25.00	113.72	0.21	842
600	(192640, 384084, 191447)	36.86	162.00	0.22	840
700	(260519, 519651, 259134)	50.79	218.88	0.23	840
800	(338786, 675986, 337202)	67.45	284.72	0.23	840
900	(440708, 879619, 438914)	88.52	370.61	0.23	840
1000	(548041, 1094088, 546050)	110.64	460.84	0.24	840

Table 5: Complete results for circles (Table 1 is an extract of this). The last column states the time per vertex in μs.

#Ellipses	#(V,E,F)	conic	**algebraic**	speedup	μs
10	(66, 112, 49)	1.33	0.12	10.49	1923
20	(214, 388, 176)	4.91	0.31	15.35	1495
30	(538, 1016, 480)	13.37	0.72	18.52	1341
40	(1070, 2060, 992)	27.58	1.37	20.09	1282
50	(1556, 3012, 1458)	40.55	1.92	21.08	1236
60	(2200, 4280, 2082)	58.13	2.71	21.43	1232
70	(2900, 5660, 2762)	76.79	3.55	21.59	1226
80	(3896, 7632, 3739)	104.14	4.63	22.45	1190
90	(4656, 9132, 4478)	124.64	5.52	22.57	1186
200	(23744, 47088, 23346)	648.71	27.01	24.01	1137
300	(49898, 99196, 49301)	1371.83	56.63	24.22	1135
400	(96502, 192204, 95704)	2660.99	108.94	24.42	1128
500	(157232, 313464, 156235)	4341.85	177.88	24.40	1131
600	(214358, 427516, 213160)	5899.76	244.41	24.13	1140

Table 6: Complete results ellipses (Table 2 is an extract of this). The last column states the time per vertex in μs.

n, m	#(V,E,F)	rational	**algebraic**	speedup	μs
2,10	(98, 220, 123)	0.23	0.12	1.89	1285
2,20	(360, 766, 407)	0.85	0.36	2.34	1008
2,30	(900, 1862, 963)	1.88	0.75	2.49	840
2,40	(1836, 3766, 1931)	4.16	1.50	2.77	817
2,50	(2306, 4724, 2419)	4.89	1.95	2.50	848
2,60	(3676, 7486, 3811)	7.40	2.92	2.53	794
2,70	(5066, 10298, 5233)	11.07	4.11	2.69	811
2,80	(6110, 12394, 6285)	12.23	4.90	2.49	802
2,90	(7828, 15868, 8041)	16.76	6.42	2.61	820
2,100	(8918, 18056, 9139)	18.00	7.28	2.47	816
6,10	(114, 254, 141)	2.05	0.25	8.15	2210
6,20	(566, 1188, 623)	11.44	0.99	11.54	1750
6,30	(1222, 2526, 1305)	22.54	2.11	10.66	1729
6,40	(2074, 4266, 2193)	35.86	3.64	9.85	1755
6,50	(3268, 6670, 3403)	54.65	5.71	9.56	1749
6,60	(4800, 9764, 4965)	81.60	8.22	9.91	1714
6,70	(7020, 14238, 7219)	124.78	11.71	10.65	1668
6,80	(8636, 17482, 8847)	143.38	14.83	9.66	1718
6,90	(11266, 22790, 11525)	198.75	19.12	10.39	1697
6,100	(13124, 26534, 13411)	223.14	22.82	9.77	1739
3,20	(422, 890, 469)	1.67	0.46	3.61	1096
4,20	(508, 1066, 559)	4.22	0.63	6.64	1251
5,20	(574, 1204, 631)	8.20	0.83	9.87	1447
6,20	(566, 1188, 623)	11.32	0.97	11.60	1724
7,20	(594, 1242, 649)	16.93	1.13	14.90	1912
8,20	(576, 1204, 629)	21.38	1.29	16.51	2247
9,20	(622, 1312, 691)	34.58	1.63	21.16	2626
10,20	(652, 1368, 717)	47.97	1.84	26.01	2827
11,20	(616, 1292, 677)	47.23	1.95	24.16	3173
12,20	(630, 1322, 693)	62.23	2.18	28.51	3464
13,20	(738, 1536, 799)	102.35	2.59	39.43	3517
14,20	(616, 1288, 673)	87.67	2.71	32.24	4414
15,20	(676, 1418, 743)	131.54	3.09	42.45	4583

Table 7: Complete results for rational functions (Table 3 is an extract of this). The last column states the time per vertex in μs.

m, d	#(V,E,F)	Bézier	**algebraic**	speedup	μs
10,-	(100, 155, 57)	0.32	0.48	0.66	4829
20,-	(259, 417, 160)	0.99	1.38	0.71	5346
30,-	(519, 889, 372)	2.44	2.76	0.88	5328
40,-	(732, 1269, 539)	3.84	4.14	0.92	5661
50,-	(1352, 2456, 1106)	7.44	7.13	1.04	5276
60,-	(1863, 3439, 1578)	10.10	10.38	0.97	5571
70,-	(2394, 4441, 2049)	16.03	12.72	1.26	5314
80,-	(2849, 5293, 2446)	19.00	16.23	1.17	5697
90,-	(3544, 6682, 3140)	SegFault	20.54	-	5797
100,-	(4513, 8581, 4070)	29.02	26.87	1.07	5955
10,1	(66, 92, 28)	0.22	0.53	0.42	8180
20,1	(211, 338, 129)	0.75	1.58	0.47	7529
30,1	(466, 805, 341)	2.62	3.17	0.82	6814
40,1	(731, 1291, 562)	4.84	5.15	0.94	7055
50,1	(1073, 1941, 870)	6.97	8.04	0.86	7502
60,1	(1567, 2905, 1340)	10.91	11.23	0.97	7168
70,1	(2103, 3924, 1823)	16.43	15.45	1.06	7347
80,1	(2539, 4772, 2235)	19.44	18.27	1.06	7197
90,1	(3270, 6185, 2917)	26.23	25.59	1.02	7828
100,1	(3674, 6939, 3267)	28.47	29.81	0.95	8116
10,2	(78, 117, 41)	0.48	0.62	0.77	8011
20,2	(197, 321, 126)	1.30	1.42	0.91	7257
30,2	(387, 664, 279)	2.03	3.34	0.60	8652
40,2	(684, 1215, 533)	4.31	5.34	0.80	7816
50,2	(1197, 2213, 1018)	8.37	9.34	0.89	7804
60,2	(1584, 2939, 1357)	12.03	12.18	0.98	7690
70,2	(2207, 4139, 1934)	20.22	17.03	1.18	7716
80,2	(2385, 4472, 2089)	22.69	20.32	1.11	8521
90,2	(3147, 5842, 2697)	27.31	25.57	1.06	8126
100,2	(3486, 6530, 3046)	30.28	29.31	1.03	8408
10,3	(49, 50, 7)	1.16	0.58	1.97	11998
20,3	(146, 215, 73)	1.87	1.55	1.20	10669
30,3	(179, 204, 32)	3.00	2.46	1.21	13796
40,3	(490, 851, 363)	10.47	4.10	2.55	8374
50,3	(780, 1360, 582)	11.28	7.79	1.44	9999
60,3	(486, 746, 265)	17.97	5.76	3.11	11858
70,3	(1571, 2901, 1332)	Error	14.69	-	9355
80,3	(1325, 2332, 1010)	18.40	14.97	1.22	11303
90,3	(2451, 4452, 2003)	22.52	26.01	0.86	10612
100,3	(3614, 6729, 3117)	33.89	29.48	1.14	8158

Table 8: Complete results for Bézier curves (Table 4 is an extract of this). The last column states the time per vertex in μs. In the first block, we considered m Bézier curves with 4 randomly chosen control points In the remaining blocks, we looked at random Bézier curves constraint to share d common points. For the instance with 90 curves, a segmentation fault occurred in the Bézier traits class. The instance $70, 3$ could not be handled by the Bézier traits class because 2 curves overlapped.

Contents

1 Introduction **3**

2 The Algebraic Kernel Package **4**

3 Geometric Applications **6**
3.1 Arrangements of Algebraic Curves 6
3.2 Rotations of Algebraic Curves 6
3.3 Boolean Set Operations 7

4 Experimental Comparisons **8**

5 Conclusions **11**

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399