

John Keyser, J. Maurice Rojas, and Koji Ouchi

ABSTRACT. We describe a method, based on the rational univariate reduction (RUR), for computing roots of systems of multivariate polynomials with rational coefficients. Our method enables exact algebraic computations with the root coordinates and works even if the underlying set of roots is positive dimensional. We describe several applications, with special emphasis on geometric modeling. In particular, we describe a new implementation that has been used successfully on certain degenerate boundary evaluation problems.

1. Introduction

We propose a new development of *exact* arithmetic over the coordinates of the points in the zero set of a system of polynomial equations with rational coefficients.

Fix a *square* system of polynomials, i.e., a system of n polynomials in n variables. Assume temporarily the zero set of the system is of *dimension zero*, i.e., the system has only finitely many common roots. Then, there exist *univariate* polynomials h and h_1, \dots, h_n with rational coefficients s.t. the finite zero set of the system is represented as [BPR03]

$$\{(h_1(\theta), \dots, h_n(\theta)) \mid \theta \in \mathbb{C} \text{ with } h(\theta) = 0\}.$$

This representation for the zero set of the system is called the *Rational Univariate Representation (RUR)*.

We implement an algorithm for computing the *exact* RUR for the zero set of a square system of polynomial equations, meaning that we compute all the rational coefficients of univariate polynomials in the RUR in full digits. Then, we develop *exact* arithmetic over numbers of the form $e(\theta_1, \dots, \theta_m)$ where e is a polynomial with rational coefficients and θ_i is an algebraic number. Putting these together, we can compute the *exact* RUR for the zero set of a (not necessarily square) system and 2) have *exact* arithmetic over the coordinates of the points in the zero set of a system of polynomial equations with rational coefficients. Furthermore, we can make our method succeed even if the zero set of the system has some positive dimensional component. In fact, we can detect whether or not a system has some positive dimensional components.

The first author was supported in part by NSF Grants DMS-0138446 and CCR-0220047.
The second author was supported in part by NSF Grants DMS-0138446 and CCR-0220047.
The third author was supported in part by NSF Grants DMS-0138446 and CCR-0220047.

The *resultant* for a system of $n + 1$ polynomials in n variables is a polynomial in the coefficients of the input polynomials that vanishes iff the system has a common root. There are different types of resultants depending on the place where the common roots of the system lie. We implement the algorithm for computing the exact RUR based on the *toric resultant*. The toric resultant based method computes the common roots of a square system in $(\mathbb{C}^*)^n$.

It is shown that some modifications of the method allow us to compute a super set of the zero set of a not necessarily square system in \mathbb{C}^n . If we could perform *exact* arithmetic over the points expressed in the *exact* RUR then we could extract the common roots of the system from this super set.

A resultant based method fails when the resultant for a system becomes zero. The latter happens, for instance, when the zero set of the system has some positive dimensional components. To handle these cases, we use the resultant perturbation technique [DE03]. Different points on positive dimensional components of the zero set of the system are picked up when different perturbation is applied. Thus, if we can perform *exact* comparison over the points expressed in the *exact* RUR then we could develop a generic algorithm to detect whether or not the zero set of the system has some positive dimensional components.

There exist implementations of both numerical and exact algorithms for computing the RUR for the zero set of a system of polynomial equations. The exact algorithm requires some normal form computation, typically, Gröbner basis computation. The toric resultant based method is presumably more efficient than the method based on Gröbner basis. Both have complexity exponential in n , but, Gröbner basis based method may cost quadratic exponents in the worst case. Furthermore, unlike the toric resultant, Gröbner basis is discontinuous w.r.t. perturbations in the coefficients of the input polynomials.

We develop exact arithmetic over the points in the zero set of a system of polynomial equations expressed in the RUR via a *root-bound approach*. The root-bound approach has been successfully used for the exact sign determination of *real* algebraic numbers of the form $E(\theta_1, \dots, \theta_n)$ where θ_i is a rational number or a *real* root of a univariate polynomial with rational coefficients and E is an algebraic expression involving $\pm, *, /$ and $\sqrt[\cdot]{}$. The sign of such a number can be obtained from the location (on the real line) of some approximation of the number to a certain precision that is controlled by the root-bound for the number. We apply this method to the exact sign determination of the real and imaginary parts of the coordinates of the points expressed in the exact RUR. In fact, our implementation will use LEDA or CORE, the libraries that support the root-bound approach to the exact sign determination of real algebraic numbers. The non-trivial part of the development is computing approximations for the real and imaginary parts of all the roots of a univariate polynomial with rational coefficients to any given precision. There does not exist an algebraic method. We must try some numerical method.

In order to show the ability of our method, we develop an exact geometric solid modeler that handles degenerate configurations.

Solving a system of polynomial equations is a key operation in geometric algorithms. In geometric algorithms, polynomials are used to describe the shape of objects and their relations. Points are described as intersections of curves and/or

surfaces, or the common roots of a system of polynomial equations. Implementation of robust geometric algorithms requires exact computation over coordinates of the common roots of a system of polynomials.

There exist only few exact geometric modelers that handle non-linear surfaces and almost none can remove degeneracies. Thus, designing an exact geometric modeler is one of the most appropriate applications to demonstrate the power of the proposed method.

The rest of this paper is organized as follows:

Section 2 gives a brief summary of related work.

Section 3 gives a detailed description of the RUR and how it is implemented.

Section 4 describes ways in which the RUR approach can be applied to various geometric problems.

Section 5 gives examples and experimental results from our implementation of the RUR method.

Section 6 concludes with a discussion of useful avenues for future work.

2. Related Work

RUR. Representing roots of a system of polynomial equations in the RUR first appeared in [Kro31] but was started to be used in computer algebra only recently [GV97] [Roj99a] [Rou99].

If a system is zero-dimensional, i.e., if a system has only finitely many roots, then the RUR for the zero set of the system can be computed via multiplication table method [Rou99] [GVRR99a] [BPR03]. The method has been extended s.t. it finds all the isolated roots as well as at least one point from each *real* positive dimensional component [ARD02]. A standard implementation of the multiplication table method requires some normal form computation, usually Gröbner basis computation.

A Gröbner-free algorithm to compute the RUR for a zero dimensional system has been proposed [GLS01]. The most recent work even handles some systems with multiple roots [Lec02]. Because of their iterative nature, it is hard to apply these algorithms to exact computation.

Resultants. The *toric resultant* (or the *sparse resultant*) for a system of $n + 1$ polynomial equations with indeterminate coefficients in n variables is a polynomial with integer coefficients in these indeterminates (as variables) that vanishes iff the system has a solution on some toric variety containing $(\mathbb{C}^*)^n$ [CE93] [Stu94] [CLO98] [CE00] [Stu02] [Emi03]. The toric resultant is expressed as some divisor of the determinant of some square matrix, called *Newton matrix* [CE93] [Stu94] [Emi96] [CE00] [Stu02] [Emi03] [Roj03]. Several efforts have been made to construct smaller Newton matrices [EC95] [D'A02] [Emi02] [Khe03]. Considering the solutions of a system of polynomial equations on the associated toric variety gives a sharper bound, known as BKK bound [Ber75] [Kus76] [Kho78] [Roj94] [Roj04], on the number of the isolated roots of the system than traditional ones (e.g. Bezout's bound), and thus, lower degree resultants.

The u -resultant method fails if the zero set of the system has some positive dimensional component since the u -resultant for such a system is identically zero. In this case, some term of the Generalized Characteristic Polynomial (GCP), which is

defined as the resultant of the perturbed system, can be used to read the coordinates of points in some finite subset of the zero set of the system.

The homogeneous Generalized Characteristic Polynomial (GCP) [Can90] works for homogeneous systems that have some positive dimensional components unless there are some multiple roots at the point at infinity.

The toric perturbation [Roj99a] [Roj00] that is defined as some term of the toric GCP [Roj97] [Roj99a] [Roj00] works even if the system has some multiple roots at the point at infinity.

A potentially more efficient version of GCP is proposed in [DE01] [DE03]. The method uses more general lifting of Newton polytopes of the polynomials in the system and find expectedly fewer monomials to be perturbed.

The toric resultant based method can be modified s.t. it finds some set containing all the affine roots [Roj96] [Roj99b] [Roj99a] [Roj00].

Polynomial System Solving. Often, in geometric applications, we are interested only in finding/counting real solutions of a system of polynomial equations. Although the set of real numbers is a subset of the set of complex numbers, in general, real solving (finding real solutions) is *not easier* than complex solving (finding complex solutions) [Roy96] [GVR99b].

The multiplication table method can be modified for real solving of zero-dimensional systems [GVR99b] [BPR03]. We saw that the method has been extended s.t. the method finds all the isolated real roots of a system as well as at least one point from each real positive dimensional component [ARD02].

The toric perturbation technique can be modified for real counting/solving of zero-dimensional systems [Roj98].

Solving a positive dimensional system means constructing an irreducible decomposition of the zero set of a system of polynomial equations [SW96]. For bivariate systems, irreducible decomposition via computing gcd of polynomials is practical. For general cases, numerical irreducible decomposition has been developed [SV00] [SVW01] [SVW03]. The algorithm obtains much geometric information by numerically computing homotopies to generic points lying on positive dimensional components. Thus, the algorithm is hard to implement exactly.

Root Bound. The root-bound approach to exact sign determination for *real* algebraic numbers has been implemented in the libraries LEDA [BMS96] [MN99] and CORE [KLY99]. These libraries support exact arithmetic and comparison over *real* algebraic numbers of the form $e(\xi_1, \dots, \xi_m)$ where e is an expression involving $+$, $-$, $/$ and $\sqrt[m]{}$, and each of ξ_1, \dots, ξ_m is a *real* root of some univariate polynomial with integer coefficients. Several improvements on root-bounds are reported in [BFMS97] [BFMS00] [LY01] [PY03].

We determine whether or not an algebraic number (given in the exact RUR) is 0 by applying a root bound approach to the real and imaginary parts of the number. We must determine the sign of a number of the form $e(\xi_1, \dots, \xi_m)$ where ξ_1, \dots, ξ_m are the real or imaginary parts of some algebraic numbers. When ξ_1, \dots, ξ_m are rational numbers or real algebraic numbers (given as roots of a univariate polynomial with rational coefficients), their signs can be exactly determined algorithmically (e.g. Sturm's method). In our case, there is no such way. Instead, we use some numerical method that computes all the roots of a univariate polynomial with rational coefficients to *any given precision*. Research on numerical methods

for computing all the roots of a univariate polynomial has a long history [Abe73] [Bin96] [BCSS97] [BF00], but none fits our purpose.

Robust Geometric Computation. The need for robustness in geometric algorithms has been addressed for years. See, for instance, [HHK89] [DY95] [Yap97].

Exact computation as a way to eliminate numerical errors appearing in solid modeling has been discussed for years [Hof89]. Much of earlier work deals with polyhedral solids but few with curved solids [SI89] [Yu91] [MOBP94] [For97]. Exact Geometric Computation (EGC) project tries to develop a large set of exact geometric algorithms using LEDA or CORE [BFMS99].

Perturbation Methods. Perturbation methods are general approaches to remove degeneracies. There are two types of perturbations; symbolic and numerical.

Symbolic perturbations move the input by an infinitesimal amount and the result is obtained by taking the limit when the infinitesimal amount goes toward zero. The works on symbolic perturbations are seen in [EM90] [Yap90] [EC91] [EC92] [Sei94] [ECS97] [For97]. Symbolic perturbations usually cost too much.

MAPC and ESOLID. MAPC [KCMK99] [Key00] [KCMK00] is the library that manipulates exact computation for two-dimensional real algebraic points and curves. ESOLID [KKM99a] [KKM99b] [Key00] [KCF+02] [KCF+04] is a robust geometric modeler built on top of MAPC. ESOLID is currently the only system that supports exact boundary evaluation for solids with curved surface. For efficiency, MAPC assumes curves are non-singular and ignores some singular intersections such as tangential intersections. Thus, ESOLID works provided when solids are in general positions.

2.1. Speed Up. Although the main concern of this work is guaranteeing the exactness, the performance is always also taken into account. Specifically, 1) we will try several proposed methods [EC95] [DE02] [Emi02] [CK03] [Khe03] even if they work only for particular type of inputs, 2) take advantages of the sparseness of matrices and 3) use fixed-precision arithmetic to guide results.

3. Exact Rational Univariate Representation

Consider a system of m polynomials f_1, \dots, f_m in n variables with rational coefficients. Let Z be the set of all the common roots of the system. Fix a finite subset Z' of Z . Every coordinate of the points in Z' is represented as a univariate polynomial with rational coefficients, called a *coordinate polynomial*, evaluated at some root of the other univariate polynomial with rational coefficients, called the *minimal polynomial* [Rou99]. That is, there exist $h, h_1, \dots, h_n \in \mathbb{Q}[T]$ s.t.

$$Z' = \{(h_1(\theta), \dots, h_n(\theta)) \in \mathbb{C}^n \mid \theta \in \mathbb{C} \text{ with } h(\theta) = 0\}.$$

This representation for Z' is called the *Rational Univariate Representation (RUR)* for Z' .

We describe a method to find the RUR for some finite subset Z' of Z which contains at least one point from every irreducible component of Z . In particular, if Z is zero-dimensional then $Z' = Z$.

In section 3.1, we discuss how the RUR for $Z' \cap (\mathbb{C}^*)^n$ is computed exactly when a system is square. In section 3.2, we present our method for determining the sign of an algebraic expression over the coordinates of points in Z' exactly. In section 3.3, we combine two results to establish our exact algorithm for computing the affine roots of any system using the RUR.

3.1. The Exact RUR for Square Systems. In this section, we describe a strictly exact implementation of the toric resultant based algorithm for computing the exact RUR for the roots of a square system having non-zero coordinates. The minimal polynomial in the RUR is derived from the toric perturbation [Roj99a] which is a generalization of the “toric” u -resultant.

3.1.1. *Toric Resultants.* Let f be a polynomial in n variables X_1, \dots, X_n with rational coefficients. Define the *support* of f to be the finite set A of exponents of all the monomials appearing in f corresponding to non-zero coefficients. Thus, A is some finite set of integer points in n -dimensional space \mathbb{R}^n . Then

$$f = \sum_{a \in A} c_a X^a, \quad c_a \in \mathbb{Q}^*$$

where $X^a = X_1^{a_1} \dots X_n^{a_n}$ for $a = (a_1, \dots, a_n)$.

A system of $n+1$ polynomials $f_0, f_1, \dots, f_n \in \mathbb{Q}[X_1, \dots, X_n]$ with corresponding supports A_0, A_1, \dots, A_n can be fixed by coefficient vectors $\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_n$ where

$$\mathbf{c}_i = (c_{ia} \in \mathbb{Q}^* \mid a \in A_i) \quad \text{s.t.} \quad f_i = \sum_{a \in A_i} c_{ia} X^a.$$

For such a system, there exists a unique (up to sign) irreducible polynomial

$$\text{TRes}_{A_0, A_1, \dots, A_n}(\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_n) \in \mathbb{Z}[\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_n],$$

called the *toric resultant* or the *sparse resultant*, s.t.

$$\begin{aligned} & \text{the system } (f_0, f_1, \dots, f_n) \text{ has a common root in } (\mathbb{C}^*)^n \\ \implies & \text{TRes}_{A_0, A_1, \dots, A_n}(\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_n) = 0. \end{aligned}$$

We also write $\text{TRes}_{A_0, A_1, \dots, A_n}(f_0, f_1, \dots, f_n)$ for the toric resultant.

We implement the subdivision based algorithm [CE93, CE00, Emi03], summarized here. The algorithm constructs a square matrix N , called the *toric resultant matrix* or *Newton matrix*, whose determinant is some multiple of the toric resultant. The elements of N are the coefficients of the input polynomials f_0, f_1, \dots, f_n . The rows and columns of N are labeled by monomials, or equivalently, integer points belonging to the supports A_0, A_1, \dots, A_n of the input polynomials. The row labeled by $a_r \in A_i$ contains the coefficients of $X^{a_r} f_i$ s.t. the column labeled by a_c contains the coefficient of the monomial term X^{a_c} . Thus, the determinant of N is a homogeneous polynomial in each coefficient vector \mathbf{c}_i . It follows that the total degree of the determinant of N , $\deg_{\mathbf{c}_i} \det N$, with respect to the coefficient vector \mathbf{c}_i is well-defined [CE93, CE00, Emi03].

The lattice points used to label the rows and columns of N are chosen according to the *mixed-subdivision* [Stu94, Stu02] of the *Minkowski sum* of the convex hulls Q_0, Q_1, \dots, Q_n of A_0, A_1, \dots, A_n . The number of integer points (and the degree of $\det N$) is predicted in terms of the *mixed-volume* [CE93, CE00, Emi03] of the Minkowski sum of Q_0, Q_1, \dots, Q_n . More precisely, writing MV_{-i} for the mixed-volume for the Minkowski sum $Q_0 + Q_1 + \dots + Q_{i-1} + Q_{i+1} + \dots + Q_n$, it is shown [CE93, CE00] that

$$(3.1) \quad \begin{cases} \deg_{\mathbf{c}_0} \det N = MV_{-0}, \\ \deg_{\mathbf{c}_i} \det N \geq MV_{-i}, \quad i = 1, \dots, n. \end{cases}$$

The equalities hold if $\det N$ is the sparse resultant [PS93].

We develop a strictly exact implementation of the subdivision based algorithm for computing the toric resultant matrix N . Computation of Q_0, Q_1, \dots, Q_n and

the mixed subdivision of their Minkowski sum both reduce to linear programming problems [CE93, CE00] which are solved by using a standard 2-phase simplex method implemented with multi-precision rational arithmetic.

3.1.2. *Toric Perturbations.* Consider a square system of n polynomials $f_1, \dots, f_n \in \mathbb{Q}[X_1, \dots, X_n]$ with supports A_1, \dots, A_n .

Let $A_0 = \{\mathbf{o}, \mathbf{b}_1, \dots, \mathbf{b}_n\}$ where \mathbf{o} is the origin and \mathbf{b}_i is the i -th standard basis vector in \mathbb{R}^n . Also, let $f_0 = u_0 + u_1X_1 + \dots + u_nX_n$ where $\mathbf{u} = (u_0, u_1, \dots, u_n)$ is a vector of indeterminates. Choose n polynomials $f_1^*, \dots, f_n^* \in \mathbb{Q}[X_1, \dots, X_n]$ with supports contained in A_1, \dots, A_n that have only finitely many common roots in $(\mathbb{C}^*)^n$. The *toric Generalized Characteristic Polynomial* TGCP(s, \mathbf{u}) for the system (f_1, \dots, f_n) is defined to be the toric resultant for the perturbed system $(f_0, f_1 - sf_1^*, \dots, f_n - sf_n^*)$:

$$\text{TGCP}(s, \mathbf{u}) = \text{TRes}_{A_0, A_1, \dots, A_n}(f_0, f_1 - sf_1^*, \dots, f_n - sf_n^*) \in \mathbb{Q}[s][\mathbf{u}].$$

Furthermore, define a *toric perturbation* $\text{TPert}_{A_0, f_1^*, \dots, f_n^*}(\mathbf{u})$ for the system (f_1, \dots, f_n) to be the non-zero coefficient of the lowest degree term in $\text{TGCP}(s, \mathbf{u})$ regarded as a polynomial in variable s . We also write $\text{TPert}_{A_0}(\mathbf{u})$ for $\text{TPert}_{A_0, f_1^*, \dots, f_n^*}(\mathbf{u})$ when f_1^*, \dots, f_n^* are fixed.

The following theorem (the Main Theorem 2.4 in [Roj99a]) guarantees that the RUR for a system is obtained from the toric perturbation.

THEOREM 3.1. [Roj99a]

$\text{TPert}_{A_0}(\mathbf{u})$ is well-defined, i.e. with a suitable choice of f_1^*, \dots, f_n^* , for any system, there exist rational numbers u_0, u_1, \dots, u_n s.t. $\text{TGCP}(s, \mathbf{u})$ always has a non-zero coefficient. $\text{TPert}_{A_0}(\mathbf{u})$ is a homogeneous polynomial in parameters u_0, u_1, \dots, u_n with rational coefficients, and has the following properties:

- (1) If $(\zeta_1, \dots, \zeta_n) \in (\mathbb{C}^*)^n$ is an isolated root of the system (f_1, \dots, f_n) then $u_0 + u_1\zeta_1 + \dots + u_n\zeta_n$ is a linear factor of $\text{TPert}_{A_0}(\mathbf{u})$.
- (2) $\text{TPert}_{A_0}(\mathbf{u})$ completely splits into linear factors over \mathbb{C} . For every irreducible component W of $Z \cap (\mathbb{C}^*)^n$, there is at least one factor of $\text{TPert}_{A_0}(\mathbf{u})$ corresponding to a root $(\zeta_1, \dots, \zeta_n) \in W \subseteq Z$.

Thus, with a suitable choice of f_1^*, \dots, f_n^* , there exists a finite subset Z' of Z s.t. a univariate polynomial $h = h(T)$ in the RUR for $Z' \cap (\mathbb{C}^*)^n$ is obtained from $\text{TPert}_{A_0}(\mathbf{u})$ by setting u_0 to a variable T and specializing u_1, \dots, u_n to some values. Immediately from (3.1)

COROLLARY 3.2.

$$(3.2) \quad \deg_{u_0} \text{TPert}_{A_0}(\mathbf{u}) = \text{MV}_{-0},$$

$$(3.3) \quad \deg_s \text{TGCP}(s, \mathbf{u}) = \sum_{i=1}^n \text{MV}_{-i} \leq \dim N - \text{MV}_{-0}.$$

3.1.3. *The Exact RUR Computation.* We give our algorithm to compute the exact RUR for the roots having non-zero coordinates of a square system. More details will be given below.

Algorithm RUR_square:

Input: $f_1, \dots, f_n \in \mathbb{Q}[X_1, \dots, X_n]$ with supports $A_0, A_1, \dots, A_n \subseteq \mathbb{Z}^n$.

Output: $h, h_1, \dots, h_n \in \mathbb{Q}[T]$, forming the RUR for some finite subset of the roots of the system in $(\mathbb{C}^*)^n$.

- 1: Compute the toric resultant matrix N and MV_{-0} for the system of polynomials with supports A_0, A_1, \dots, A_n .
- 2:
 - 2-1: Choose generic $u_0, u_1, \dots, u_n \in \mathbb{Q}$ and $f_1^*, \dots, f_n^* \in \mathbb{Q}[X_1, \dots, X_n]$.
 - 2-2: Compute some multiple of $\text{TGCP}(s, \mathbf{u})$.
 - 2-3: If $\text{TGCP}(s, \mathbf{u}) \equiv 0$ then go to 2-1.
- 3: Compute $h(T) := \text{TPert}_{A_0}(T, u_1, \dots, u_n)$ where $\text{TPert}_{A_0}(\mathbf{u})$ is the non-zero coefficient of the lowest degree term in (the multiple of) $\text{TGCP}(s, \mathbf{u})$.
- 4: For $i := 1, \dots, n$ do:
 - 4-1: Compute $p_i^\pm(t) := \text{TPert}_{A_0}(t, u_1, \dots, u_{i-1}, u_i \pm 1, u_{i+1}, \dots, u_n)$.
 - 4-2: Compute the square-free parts $q_i^\pm(t)$ of $p_i^\pm(t)$.
 - 4-3: Compute a linear (as a polynomial in t) gcd $g(t)$ of $q_i^-(t)$ and $q_i^+(2T - t)$.
 - 4-4: Set $h_i(T) := -T - \frac{g_0(T)}{g_1(T)} \bmod h(T)$ where $g_0(T)$ and $g_1(T)$ are the constant term and the linear coefficient of $g(t)$.

All the steps can be computed exactly.

Step 1 determines the toric resultant matrix by using the subdivision based algorithm. Some entries still undetermined. Note step 1 needs to be performed once and only once.

Step 2-1 is performed by choosing random values for u_i and coefficients of f_i^* .

From (3.3), we know a bound on the degree of $\text{TGCP}(s, \mathbf{u})$ (as a polynomial in s) at step 2-2, and can therefore compute some multiple of it via interpolation. More precisely, choose $\dim N - MV_{-0}$ many values for s , specialize the entries of N with those values for s , and interpolate $\det N$ to obtain the non-zero coefficient of the lowest degree term s^l in $\text{TGCP}(s, \mathbf{u})$.

Step 2-3 checks to make sure the generic choice made in step 2-1 was acceptable to define. It can be shown that step 2 must be performed only once with probability 1. We could give only some of u_1, \dots, u_n and the coefficients of f_1^*, \dots, f_n^* non-zero values (and the rest zero) but necessarily precomputation becomes costly [Roj99a].

Step 3 determines the univariate polynomial h in the RUR, introducing the variable T . From (3.2), we know the degree of $\text{TPert}_{A_0}(T, u_1, \dots, u_n)$ (as a polynomial in T), and can compute it via interpolation. More precisely, we choose $MV_{-0} - 1$ many values for u_0 along with the one we chose at step 2-2, specialize the elements of N with those values, and interpolate to compute $\text{TPert}_{A_0}(T, u_1, \dots, u_n)$.

Similar to steps 2-2 and 3, $p_i^\pm(t)$ are determined via interpolation at step 4-1.

At step 4-3, we compute the gcd $g(t)$ of $q_i^-(t)$ and $q_i^+(2T - t)$ (as a polynomial in t). We use the fact that $g(t)$ will be linear with probability 1 (dependent on the choices of u_i in step 2-1). In this case, the coefficients of $g(t)$ are the *first subresultants* for $q_i^-(t)$ and $q_i^+(2T - t)$ [Can90] [GV91] which are defined to be Sylvester resultants for some submatrices of the Sylvester matrix for q_i^\pm .

The rest of the above involve only basic operations over the ring of polynomials with rational coefficients.

Hence, it is possible to implement the algorithm **RUR_Square** exactly by the use of multi-precision rational arithmetic in order to obtain the *exact* RUR.

3.2. Root Bound Approach to Exact Computation. In this section, we describe our method to support exact computation for the coordinates of the roots of a system, expressed in the exact RUR. Note that in general these numbers are

complex. We begin by summarizing the method for finding root bounds for real algebraic numbers proposed in [BFMS99] and [KLY99]. We then extend this method to find bounds for complex algebraic numbers. Finally, we describe our method to determine the sign of numbers expressed in the exact RUR.

In this section, we assume for simplicity that all the polynomials have integer coefficients. The results are still valid for polynomials with rational coefficients.

3.2.1. Root Bounds. Let α be an algebraic number. There exists a positive real number ρ , called a *root bound* ρ for α , which has the following property: $\alpha \neq 0 \Leftrightarrow |\alpha| \geq \rho$. Having a root bound for α , we can determine the sign of α by computing an approximation $\tilde{\alpha}$ for α s.t. $|\tilde{\alpha} - \alpha| < \frac{\rho}{2}$, namely, $\alpha = 0$ iff $|\tilde{\alpha}| < \frac{\rho}{2}$.

The root bound for an algebraic number started to be used in computer algebra systems and drastic improvements have been made. In order to make our discussion flexible, we use a conservative root bound introduced by Mignotte [MS99]: define the Mahler measure (or simply the measure) $M(e)$ of a polynomial $e(T) = e_n \prod_{i=1}^n (T - \zeta_i) \in \mathbb{Z}[T]$ with $e_n \neq 0$ as

$$M(e) = |e_n| \prod_{i=1}^n \max\{1, |\zeta_i|\}.$$

Moreover, define the *degree* $\deg \alpha$ and *measure* $M(\alpha)$ of an algebraic number α to be the degree and measure of a minimal polynomial for α over \mathbb{Z} . Since, over \mathbb{Z} , a minimal polynomial for an algebraic number is uniquely determined up to a sign, the degree and measure of α are well-defined. Then, $\frac{1}{M(\alpha)} \leq |\alpha| \leq M(\alpha)$.

3.2.2. Exact Computation for Real Algebraic Numbers. The root bound approach to exact sign determination for real algebraic numbers has been implemented in the libraries LEDA [MN99] and Core [KLY99]. These libraries support exact arithmetic and comparison over real algebraic numbers of the form $e(\xi_1, \dots, \xi_m)$ where e is an expression involving $+$, $-$, $/$ and $\sqrt[m]{}$, and each of ξ_1, \dots, ξ_m is a *real* root of some univariate polynomial with integer coefficients. To determine whether or not $e = 0$, we compute an approximation \tilde{e} for e to enough precision s.t. the root bound guarantees the sign.

The root bounds implemented in LEDA and Core are “constructive”, i.e., they are efficiently calculated without actually computing minimal polynomials for numbers, typically using some norms for minimal polynomials. They established recursive rules to bound the degree and measure of the minimal polynomial for a real algebraic number $f(\xi_1, \dots, \xi_m) \circ g(\xi_1, \dots, \xi_m)$ from those for $f(\xi_1, \dots, \xi_m)$ and $g(\xi_1, \dots, \xi_m)$ where f and g are some expressions and \circ is some operator.

LEDA and Core use *precision-driven computation* [DY95] [BFMS99] to compute an approximation \tilde{e} for e to prescribed precision p . Suppose e is a number of the form $e_1 \circ e_2$. Precision computation is applied to e as follows; First, calculate precisions p_1 and p_2 to which e_1 and e_2 will be approximated. Then, compute approximations \tilde{e}_1 and \tilde{e}_2 for e_1 and e_2 to precision p_1 and p_2 , respectively. Finally, compute $\tilde{e}_1 \circ \tilde{e}_2$ to obtain \tilde{e} .

3.2.3. Exact Computation for Complex Algebraic Numbers. We want to determine the exact sign of the coordinates of points expressed in terms of polynomials h_i 's evaluated at roots of the polynomial h . In general, roots of h are not real. Thus, the root bounds approach to exact sign determination for real algebraic numbers must adapt to an exact zero-test for complex algebraic numbers.

Let $e(\zeta_1, \dots, \zeta_m)$ be a complex algebraic number where e is given as an expression involving $+$, $-$, \cdot and $/$, and ζ_1, \dots, ζ_m are complex algebraic numbers. In order to test whether or not $e = 0$, we apply recursive rules to the real algebraic numbers

$$e_R(\Re\zeta_1, \dots, \Re\zeta_m, \Im\zeta_1, \dots, \Im\zeta_m) \quad \text{and} \quad e_I(\Re\zeta_1, \dots, \Re\zeta_m, \Im\zeta_1, \dots, \Im\zeta_m)$$

where e_R and e_I are expressions satisfying

$$\begin{aligned} e(\zeta_1, \dots, \zeta_m) &= e_R(\Re\zeta_1, \dots, \Re\zeta_m, \Im\zeta_1, \dots, \Im\zeta_m) \\ &\quad + \sqrt{-1}e_I(\Re\zeta_1, \dots, \Re\zeta_m, \Im\zeta_1, \dots, \Im\zeta_m). \end{aligned}$$

For example, if $e(\zeta) = \zeta^2$ then $e_R(\Re\zeta, \Im\zeta) = (\Re\zeta)^2 - (\Im\zeta)^2$ and $e_I(\Re\zeta, \Im\zeta) = 2\Re\zeta\Im\zeta$.

In order to complete the adaption, the base cases for recursion must be treated. Thus, our remaining task is stated as follows:

Let ζ be an algebraic number specified as a root of a univariate polynomial e with integer coefficients. For real numbers $\Re\zeta$ and $\Im\zeta$, we would like to compute 1) ‘‘constructive’’ bounds for the degrees and measures of $\Re\zeta$ and $\Im\zeta$ and 2) approximations to any prescribed precision.

For 2), Aberth’s method [**Abe73**] is used to compute approximations for the (real and imaginary parts of) roots of univariate polynomials. The method is implemented with floating point numbers with multi-precision mantissa in order to obtain an approximation to any given precision.

For 1), we first find univariate polynomials (with integer coefficients) R_e and I_e such that $R_e(\Re\zeta) = I_e(\Im\zeta) = 0$. We then calculate bounds on the degrees and measures of $\Re\zeta$ and $\Im\zeta$ from the degrees and coefficients of R_e and I_e .

PROPOSITION 3.3. *Let ζ be an algebraic number specified as a root of a polynomial $e(T) \in \mathbb{Z}[T]$. Write $\text{SylRes}_U(f, g)$ for the Sylvester resultant of univariate polynomials f and g w.r.t. variable U . Then*

- (1) $\Re\zeta$ is a real algebraic number and a root of

$$R_e(T) = \sum_{i=0}^m 2^i s_i T^i \in \mathbb{Z}[T]$$

where $\sum_{i=0}^m s_i T^i = \text{SylRes}_U(e(T - U), e(U))$.

- (2) $\Im\zeta$ is a real algebraic number and a root of

$$I_e(T) = \sum_{j=0}^{\lfloor \frac{m}{2} \rfloor} 2^{2j} (-1)^j s_{2j} T^{2j} \in \mathbb{Z}[T]$$

where $\sum_{i=0}^m s_i T^i = \text{SylRes}_U(e(T + U), e(U))$.

PROOF. If ζ is a root of e then its complex conjugate $\bar{\zeta}$ is also a root of e . Thus, the sum $\zeta + \bar{\zeta} = 2\Re\zeta$ of two roots of e is a root of $\text{SylRes}_U(e(T - U), e(U))$. Similarly, the difference $\zeta - \bar{\zeta} = 2\sqrt{-1}\Im\zeta$ of two roots of e is a root of $\text{SylRes}_U(e(T + U), e(U))$.

If 2ξ is a root of $\sum_{i=0}^m s_i T^i$ then ξ is a root of $\sum_{i=0}^m 2^i s_i T^i$.

If, for $\xi \in \mathbb{R}$, $\sqrt{-1}\xi$ is a root of $\sum_{i=0}^m s_i T^i$ then ξ is a root of $\sum_{j=0}^{\lfloor \frac{m}{2} \rfloor} (-1)^j s_{2j} T^{2j}$. \square

By Gauß's lemma, if α is a root of a polynomial $e(T) = \sum_{i=0}^n e_i T^i \in \mathbb{Z}[T]$ with $e_n e_0 \neq 0$ then $\deg \alpha \leq \deg e$ and $M(\alpha) \leq M(e)$. By Landau's theorem [MS99], for $e(T) \in \mathbb{Z}[T]$, $M(e) \leq \|e\|_2 = \sqrt{\sum_{i=0}^n |e_i|^2}$. Thus, we could use $\deg e$ and $\|e\|_2$ as “constructive” upper bounds on $\deg \alpha$ and $M(\alpha)$.

PROPOSITION 3.4. *Following the notation above*

- (1) $\deg \Re \zeta \leq \deg R_e \leq \deg^2 e$,
 $M(\Re \zeta) \leq M(R_e) \leq \|R_e\|_2 \leq 2^{2n^2+n} \|e\|_2^{2n}$,
- (2) $\deg \Im \zeta \leq \deg I_e \leq \deg^2 e$ and
 $M(\Im \zeta) \leq M(I_e) \leq \|I_e\|_2 \leq 2^{2n^2+n} \|e\|_2^{2n}$.

3.2.4. *Exact Sign Determination for the Exact RUR.* Let e be a rational function in n variables X_1, \dots, X_n with rational coefficients. Also, let Z' be a finite set of n -dimensional algebraic numbers. Assume we have univariate polynomials h, h_1, \dots, h_n with integer coefficients s.t. for every point $(\zeta_1, \dots, \zeta_n) \in Z'$, $(\zeta_1, \dots, \zeta_n) = (h_1(\theta), \dots, h_n(\theta))$ for some root θ of h . We would like to determine whether or not $e(\zeta_1, \dots, \zeta_n) = 0$ exactly.

Algorithm Exact_Sign:

Input: $e \in \mathbb{Q}(X_1, \dots, X_n)$ and $h, h_1, \dots, h_n \in \mathbb{Z}[T]$.

Output: The exact signs of the real and imaginary parts of $e(h_1(\theta), \dots, h_n(\theta))$ for every root θ of h .

1: Set up bivariate rational functions r_R and r_I with integer coefficients s.t.

$$e(h_1(\theta), \dots, h_n(\theta)) = r_R(\Re \theta, \Im \theta) + \sqrt{-1} r_I(\Re \theta, \Im \theta).$$

2: Recursively compute the root bounds for r_R and r_I . The base cases are given in Proposition 3.4.

3: Recursively compute approximations for $r_R(\Re \theta, \Im \theta)$ and $r_I(\Re \theta, \Im \theta)$ to a certain precision such that the root bounds allow us to determine their signs. The base case, i.e. computing approximations for $\Re \theta$ and $\Im \theta$ to a certain precision, is done by Aberth's method.

3.3. Exact Computation for the Exact RUR. In this section, we present our exact algorithm to compute the exact RUR for the affine roots of a system.

3.3.1. *Affine Roots.* Recall that Theorem 3.1 describes only those roots having non-zero coordinates. It can be shown [Roj99a, Roj00] that, if A_1, \dots, A_n are replaced by $\{\mathbf{o}\} \cup A_1, \dots, \{\mathbf{o}\} \cup A_n$ then the RUR for some finite *superset* Z'' of Z' is found. Those extra points in $Z'' \setminus Z$ can be removed simply by testing, for each $x \in Z''$, whether or not x is a root of the original system, i.e., $f_1(x) = \dots = f_n(x) = 0$. The last test reduces to the exact sign determination problem: for every root ζ of h in the exact RUR for Z'' , test whether or not $f_i(h_1(\zeta), \dots, h_n(\zeta)) = 0$ for $i = 1, \dots, n$.

3.3.2. *Non-square Systems.* Consider a system of m polynomials f_1, \dots, f_m in n variables with rational coefficients. We would like to compute the exact RUR for some finite subset Z' of the set Z of the roots of the system. If $m \neq n$ then we construct some square system s.t. the set of the roots of the square system is a super set of the zero set of the original system.

If $m < n$ then construct a square system by adding $m - n$ copies of f_m to the input system.

Otherwise, $m > n$. Let

$$g_i = c_{i1}f_1 + \cdots + c_{im}f_m, \quad i = 1, \dots, n,$$

where c_{11}, \dots, c_{nm} are generic rational numbers. It can be shown that there exist $c_{11}, \dots, c_{nm} \in \mathbb{Q}$ s.t. every irreducible component of the set \bar{Z} of all the common roots of g_1, \dots, g_n is either an irreducible component of the set Z of all the common roots of f_1, \dots, f_m or a point [Roj00]. We have already seen that we can compute the exact RUR for points in some finite subset \bar{Z}' of \bar{Z} s.t. \bar{Z}' contains at least one point from every irreducible component of \bar{Z} . Then, we can construct a finite subset Z' of Z containing at least one point on every irreducible component of Z by simply testing, for each $x \in \bar{Z}'$, whether or not x is a root of the original system, i.e., $f_1(x) = \cdots = f_m(x) = 0$. The last test reduces to the exact sign determination problem.

3.3.3. *The Exact RUR.* Our exact algorithm to compute the RUR is thus as follows:

Algorithm: RUR:

Input: $f_1, \dots, f_m \in \mathbb{Q}[X_1, \dots, X_n]$.

Output: $h, h_1, \dots, h_n \in \mathbb{Q}[T]$, forming the RUR for some finite subset of the roots of the system.

1: Form a square system (g_1, \dots, g_n) .

1-1: If $m < n$ then form a square system

$$g_1 = f_1, \dots, g_m = f_m, g_{m+1} = \cdots = g_n = f_m.$$

1-2: If $m = n$ then $g_i = f_i$, $i = 1, \dots, n$.

1-3: If $m > n$ then form a square system

$$g_i = c_{i1}f_1 + \cdots + c_{im}f_m, \quad i = 1, \dots, n$$

where c_{11}, \dots, c_{nm} are generic rational numbers.

2: For $i := 1, \dots, n$ do:

2-1: Set A_i to be the support of g_i . If g_i does not have a constant term then $A_i = \{\mathbf{o}\} \cup A_i$.

3: Use Algorithm **RUR_square** to compute $h, h_1, \dots, h_n \in \mathbb{Q}[T]$ forming the RUR for some finite subset \bar{Z}' of the set \bar{Z} of the affine roots of g_1, \dots, g_n .

4: If steps 1-3 and/or 2-1 are executed then use Algorithm **Exact_Sign** to test whether or not

$$f_i(h_1(\theta), \dots, h_n(\theta)) = 0, \quad i = 1, \dots, n$$

for every root θ of h .

Exactness immediately follows from the fact that both **RUR_square** and **Exact_Sign** are implemented exactly.

4. Applications

In this section, we describe some applications of the RUR to various geometric problems. A number of geometric problems involve solving systems of polynomials, and thus algebraic number computations. We first present some algorithms to support exact computation over algebraic numbers, and then focus on how those can be used in specific situations that arise during boundary evaluation in CAD applications.

4.1. Exact Computation for Algebraic Numbers.

4.1.1. *Positive Dimensional Components.* We present a generic algorithm to determine whether or not the set of roots of a given system of polynomials with rational coefficients has positive dimensional components.

Recall that Algorithm **RUR** finds the exact RUR for some finite subset Z' of Z which contains at least one point from every irreducible component of Z . If Z is infinite (i.e., has positive-dimensional components) then Z' depends on the polynomials f_1^*, \dots, f_n^* used to perturb the input system.

Suppose two distinct executions of Algorithm **RUR** find two finite subsets Z'_1 and Z'_2 of Z and their exact RUR's:

$$Z'_k = \left\{ \left(h_1^{(k)}(\theta_k), \dots, h_n^{(k)}(\theta_k) \right) \mid h^{(k)}(\theta_k) = 0 \right\}, k = 1, 2.$$

If ζ is an isolated point in Z then $\zeta \in Z'_1 \cap Z'_2$, and thus $\exists \theta_1$ and $\theta_2 \in \mathbb{C}$ s.t.

$$\zeta = \left(h_1^{(1)}(\theta_1), \dots, h_n^{(1)}(\theta_1) \right) = \left(h_1^{(2)}(\theta_2), \dots, h_n^{(2)}(\theta_2) \right).$$

Hence, $Z'_1 \setminus Z'_2 \neq \emptyset$ implies that Z has some positive dimensional components. We can compare Z'_1 and Z'_2 pointwise using Algorithm **Exact_Sign**.

Algorithm Positive Dimensional Components:

Input: $f_1, \dots, f_m \in \mathbb{Q}[X_1, \dots, X_n]$ and a (small) positive integer `Max_Counts`.

Output: *True* if the set Z of the roots of the system (f_1, \dots, f_m) has some positive dimensional components.

1: `Count` := 0, `Has_Pos_Dim_Compo` := *Probably_False*.

2: While `Count` < `Max_Counts` and `Has_Pos_Dim_Compo` = *Probably_False* do:

2-1: Use Algorithm **RUR** to compute the exact RUR for some finite subsets Z'_1 and Z'_2 of Z . Increment `Count`.

2-2: Use Algorithm **Exact_Sign** to compare Z'_1 and Z'_2 pointwise. If they are not the same then `Has_Pos_Dim_Compo` := *True*.

If Z has a positive dimensional component and polynomials f_i^* in Algorithm **RUR** are chosen generically, then almost always $Z'_1 \setminus Z'_2 \neq \emptyset$. Thus, `Max_Counts` is usually set to be 2.

4.1.2. *Root Counting.* We present an algorithm to count the number of the common roots of a given system of polynomials with rational coefficients.

Algorithm **Positive_Dimensional_Components** detects whether or not the set of the roots has positive-dimensional components. If the system has only finitely many roots then the number of roots of the system is the same as the number of the roots of the univariate polynomial h in the RUR.

Algorithm: Root Counting:

Input: $f_1, \dots, f_m \in \mathbb{Q}[X_1, \dots, X_n]$.

Output: The number of the roots of the system (f_1, \dots, f_m) .

1: Use Algorithm **Positive Dimensional Components** to test whether or not the set Z of the roots of the system (f_1, \dots, f_m) has some positive dimensional components. If so then return ∞ .

2: Otherwise, we may assume Z is finite. At step 1 of Algorithm **Positive Dimensional Components**, we computed the exact RUR for Z . Return the number of roots of the minimal polynomial h in the RUR for Z , namely, $\deg h$.

4.1.3. *Real Roots.* We present an algorithm to compute the real roots of a given system of polynomials with rational coefficients.

Assume that the system has only finitely many roots. We first compute the exact RUR for the roots of the system, and then use Algorithm **Exact_Sign** to determine the sign of the imaginary parts of the roots.

Algorithm Real Roots:

Input: $f_1, \dots, f_m \in \mathbb{Q}[X_1, \dots, X_n]$.

Output: The set R of the real roots of the system (f_1, \dots, f_m) .

1: Use Algorithm **RUR** to compute the exact RUR for the set Z of the roots of the system:

$$Z = \{(h_1(\theta), \dots, h_n(\theta)) \mid h(\theta) = 0\}.$$

2: Set $R := \emptyset$.

3: For $i := 1, \dots, n$ do:

3-1: Set up expressions h_{Ri} and h_{Ii} satisfying

$$h_i(\theta) = h_{Ri}(\Re\theta, \Im\theta) + \sqrt{-1}h_{Ii}(\Re\theta, \Im\theta).$$

3-2: Use Algorithm **Exact_Sign** to determine the sign of $h_{Ii}(\Re\theta, \Im\theta)$ at every root θ of h .

3-3: If $h_{I1}(\Re\theta, \Im\theta) = \dots = h_{In}(\Re\theta, \Im\theta) = 0$ then

$$R := R \cup \{(h_{R1}(\Re\theta, \Im\theta), \dots, h_{Rn}(\Re\theta, \Im\theta))\}.$$

It is important to note the algorithm **Real Roots** works correctly only if the system has finitely many common roots. In particular, the algorithm may not be able to find real points lying on some (complex) positive dimensional components.

4.2. Application to Boundary Evaluation. We now give a brief description of how the exact RUR can be applied to a specific geometric computation—boundary evaluation.

4.2.1. *Overview of Exact Boundary Evaluation.* Boundary evaluation is a key operation in computer aided design. It refers to the process of determining the boundary of a solid object produced as the result of an operation - usually a Boolean combination (union, intersection, or difference) of two input solids. It is the key element for conversion from a Constructive Solid Geometry (CSG) format to a Boundary Representation (B-rep) format. Achieving accuracy and robustness with reasonable efficiency remains a challenge in boundary evaluation.

Boundary evaluation involves several stages, but the key fundamental operations involve finding solutions to polynomial systems. The input objects are usually described by a series of rational parametric surfaces described as polynomials with rational coefficients. Implicit forms for these surfaces are often known, or else can be determined. Intersections of these surfaces form the edges of the solids, sometimes represented inside the parametric domain as algebraic plane curves. These curves represented in the patch domain and defined by the intersections of two surfaces are known as either *trimming curves* if they are specified in the input, or *intersection curves* if they arise during boundary evaluation. Intersection curves that are output become trimming curves when input to the next boundary evaluation operation.

Intersections of three or more surfaces form vertices of the solids. Such vertices may be represented in 3D space (as the common solution to three or more trivariate equations), within the parametric domain of the patches (as the common

solution of two or more bivariate equations), or a combination of these. The coordinates of these vertices are thus tuples of real algebraic numbers. The accuracy, efficiency, and robustness of the entire boundary evaluation operation is usually a direct result of the accuracy, efficiency, and robustness of the computations used to find and work with these algebraic numbers. Determining the signs of algebraic expressions evaluated at algebraic numbers thus becomes key to performing the entire computation.

4.2.2. *ESOLID and Exact Boundary Evaluation.* The ESOLID system was created in order to perform exact boundary evaluation [KCF⁺02, KCF⁺04]. ESOLID uses exact representations and computations throughout to guarantee accuracy and eliminate robustness problems due to numerical error (e.g. roundoff error and its propagation). ESOLID finds and represents points in the 2D domain only, using the MAPC library [KCMK00]. This 2D representation is sufficient for performing all of boundary evaluation. Though significantly less efficient than an equivalent floating-point routine, it runs at “reasonable” speed—at most 1-2 orders of magnitude slower than an inexact approach on real-world data. Unfortunately, ESOLID is designed to work only for objects in general position. Overcoming this limitation has been a major motivator of the work presented here.

The 2D vertex representation (coupled with techniques that produce bounding intervals in 3D) is sufficient for performing the entire boundary evaluation computation. MAPC represents points (with real algebraic coordinates) using a set of bounding intervals, guaranteed to contain a unique root of the defining polynomials. These intervals are found by using the Sylvester resultant and Sturm sequences to determine potential x and y values of curve intersections. Then, a series of tests are performed to find which x coordinates belong with which other y coordinates, thus forming an isolating interval. These isolating intervals can be reduced in size on demand as necessary. One nice aspect of this approach is that only roots in a particular region of interest are found, eliminating work that might be performed for roots outside of that region. Generally, a lazy evaluation approach is used; intervals are reduced only as necessary to guarantee sign operations. Often they need to be reduced far less than worst-case root bounds would indicate.

4.2.3. *Incorporating the Exact RUR.* The exact RUR could be incorporated directly into a boundary evaluation approach by following the ESOLID framework, but replacing the MAPC point representations with RUR representations. Only 2D point representations would be necessary. The following information would then be kept at each point:

- (1) The exact RUR: the univariate polynomials h, h_1 and h_2 with rational coefficients.
- (2) A bound on a unique (complex) root of h .
- (3) A bounding interval determined by evaluating the root of h within h_1 and h_2 .
- (4) The two original polynomials used to compute the exact RUR.

While only the first two items are necessary to exactly specify the root, the other information allows faster computation in typical operations.

Points are isolated using **Real Roots**. Whenever we need to use the points, the computation is phrased as an algebraic expression in which that point must be evaluated. From this, we can obtain root bounds for the algebraic numbers,

and thus the precision to which we must determine the root of h . In doing so, we guarantee accurate sign evaluation at any point in the boundary evaluation process.

Note that in practice, due to the relatively slower performance of the RUR method compared to the MAPC method, it is likely that a hybrid representation should be used instead. In particular, one would like to use the MAPC approach where it is most applicable, and the RUR where it is most applicable. Specifically, the RUR seems better suited for degenerate situations.

4.3. Degeneracy Detection in Boundary Evaluation. As stated previously, degeneracies are a major remaining obstacle in our exact boundary evaluation implementation. The exact RUR is able to cope with degenerate situations, however, and thus is more well-suited for detecting when a degenerate situation is occurring.

We outline here how the exact RUR method can be applied to detect each of the common degenerate situations that arise in boundary evaluation. Space does not permit a detailed review of how these cases fit within the larger boundary evaluation framework. See [Key00] for a more thorough discussion of the ways that each of these degeneracies arise and manifest themselves. We focus here on degeneracies that can be easily detected using the exact RUR method. Certain other degenerate situations are easily detected by simpler, more direct tests. For example, overlapping surfaces are immediately found in the vanishing of an intersection curve.

We focus here only on degeneracy detection, as there is more than one way one might want to handle the degeneracy (e.g. numerical perturbation, symbolic perturbation, special-case treatment).

4.3.1. General Degeneracy Consideration. The exact RUR method is particularly well-suited for problems that arise in degenerate situations. Unlike methods created under genericity assumptions (e.g. [KCMK00]), the exact RUR will find roots for systems of equations, even when they are in highly degenerate configurations. This makes it highly useful for treating degeneracies, beyond just detecting them.

An example from boundary evaluation is when intersection curves contain singularities (e.g. self-intersections, isolated point components, cusps). Methods such as that in [KCMK00] fail in such situations, while the exact RUR approach is perfectly capable of finding, e.g. the intersection between two curves at a self-intersection of one of the curves.

Note that in a practical sense, the exact RUR might not have been used to represent all points to begin with, due to its relatively poorer performance for generic cases (see section 5.2). Instead, the exact RUR might be used to check for coincidence only when it seems likely that a degenerate situation is occurring. The best approach for pursuing such a combination is a subject for future study, but a common operation that would be necessary in such cases would be the conversion of a point given in the existing system to one expressed as the exact RUR. We will limit our discussion to the 2D case, as this is what is necessary for boundary evaluation.

Degeneracies are detected during the boundary evaluation process by checking for irregular interactions. They occur when one of the following happens in 2D:

- (1) Three or more curves intersect at a single point. This includes the cases of singularities (where the curve and its two derivative curves intersect at a common point).
- (2) Two curves intersect tangentially.
- (3) Two curves share a positive-dimensional component.

All of these cause problems for a generic system solver (potentially resulting in crashes), and ESOLID in particular.

The exact RUR will find roots for systems of equations, even when they are in highly irregular configurations. The exact RUR can be used to detect these situations:

- (1) *Three or more curves meeting at a point.* Algorithm **Real_Roots** can be used to compute intersections of the system of curves. Recall that the exact RUR method works for curves having singularities. Algorithm **Exact_Sign** can also be used if several of the curves have already been intersected (generically). In that case, the test is for whether that point lies on another curve.
- (2) *Tangential intersections.* Algorithm **Real_Roots** will again be able find these intersections.
- (3) *Positive-dimensional components.* Algorithm **Positive Dimensional Components** will be able to determine whether there is a positive dimensional component. In general, though, other methods may be more efficient for detecting this degeneracy, in particular, in 2D.

Thus, the exact RUR provides a mechanism for detecting any of the (non-trivial) degenerate object intersections.

5. Experimental Results

We have implemented the exact RUR method in GNU C++. We use the Gnu Multiple Precision (GMP) arithmetic library. All the experiments shown in this section are performed on a 3 GHz Intel Pentium CPU with 6 GB memory.

We present here examples in order to indicate the time requirements of various parts of the exact RUR implementation, as well as to understand the time requirement in comparison to competing methods. These are *not* meant to provide a comprehensive survey of all potential cases, but rather to give greater intuitive insight into the exact RUR procedure.

5.1. Solving Systems in the Exact RUR. In this section, we show timing breakdowns for the application of the exact RUR to a few sample systems. We give a brief discussion of each case (a detailed description in one instance), and summarize the results.

5.1.1. Details of Examples. F_1 : Positive Dimensional Components

Consider a system F_1 of two polynomials

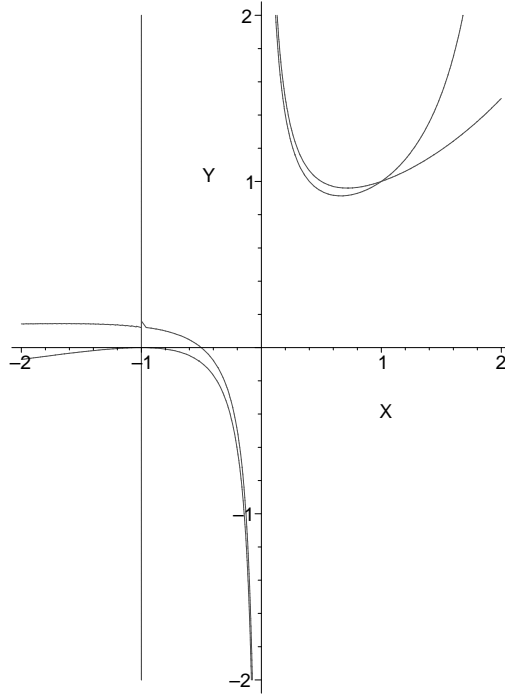
$$\begin{aligned} f_1 &= 1 + 2X - 2X^2Y - 5XY + X^2 + 3X^2Y, \\ f_2 &= 2 + 6X - 6X^2Y - 11XY + 4X^2 + 5X^3Y \end{aligned}$$

The roots of the system F_1 are the points $\{(1, 1), (\frac{1}{7}, \frac{7}{4})\}$ and the line $X = -1$.

We give a detailed description of Algorithm **Positive Dimensional Components** here as illustration.

Input System	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
# polynomial	2	2	2	3	2	2	2	2
# variable	2	2	2	3	2	2	2	2
max deg of monomials	5	3	3	2	2	2	2	2
max len of coeff	4	2	2	2	309	402	54	97
# roots of system	∞	2	2	∞	2	2	2	4
# roots of h	4	2	2	4	2	4	2	2
# real roots of system	2	2	2	-	2	2	0	4
max len of coeff of h_i	111	5	6	19	364	311	103	451
total time (sec)	2.59	0.0865	0.103	269.0	0.0926	0.104	0.183	0.687
% res matrix	0.8	19.7	18.8	0.1	14.1	12.5	11.8	3.1
% compute h	26.4	15.3	16.2	15.1	14.6	15.3	17.4	5.5
% compute q_i^\pm	70.4	60.7	64.4	84.8	58.0	59.6	67.5	22.5
% compute h_i	2.2	2.0	0.0	0	10.0	9.8	1.8	67.9
% approx roots of h	0.1	0.5	0.3	0	0.9	0.8	0.3	0.4
% evaluate h_i	0.1	1.8	0.3	0	2.4	2.0	1.2	0.6

TABLE 1. Timing breakdown for several examples.

FIGURE 1. F_1

Since F_1 is a square system, Algorithm **RUR** immediately calls Algorithm **RUR_square**.

If we choose $u_1 = \frac{9}{2}$, $u_2 = \frac{17}{2}$,

$$f_1^* = 17X^3Y + 74X^2Y + 94, \quad \text{and} \quad f_2^* = 105X^3Y + 93X^2 + 98XY,$$

at step 2-1, then we obtain the exact RUR for some finite subset Z'_1 of the roots of F_1 as

$$\begin{aligned} h^{(1)}(T) &= -284032T^4 - 4928704T^3 + 26141910T^2 + 440186393T - 1411684417, \\ h_1^{(1)}(T) &= \frac{7689907930880}{5870227699098039}T^3 + \frac{1037528584832}{124898461682937}T^2 \\ &\quad - \frac{1313640082212220}{5870227699098039}T - \frac{851151731612679}{1956742566366013}, \\ h_2^{(1)}(T) &= -\frac{23069723792640}{3326462362822221}T^3 - \frac{3112585754496}{707757949536643}T^2 \\ &\quad + \frac{27435113904634}{3326462362822221}T + \frac{7660365584514111}{3326462362822221}. \end{aligned}$$

If we choose $u_1 = \frac{9}{2}$, $u_2 = \frac{17}{2}$,

$$f_1^* = 17X^3Y + 110 + 112X^2Y, \quad \text{and} \quad f_2^* = 58X^3Y + 50X^2 + 63XY,$$

at step 2-1, then we obtain the exact RUR for some finite subset Z'_2 of the roots of F_1 as

$$\begin{aligned} h^{(2)}(T) &= -278656T^4 - 4324928T^3 + 38195682T^2 + 477531215T - 1790337263 \\ h_1^{(2)}(T) &= \frac{23827150611712}{20005826573410785}T^3 + \frac{21652317350528}{4001165314682157}T^2 \\ &\quad - \frac{4673278548948724}{20005826573410785}T - \frac{6694752830187523}{20005826573410785}, \\ h_2^{(2)}(T) &= -\frac{23827150611712}{37788783527553705}T^3 - \frac{21652317350528}{7557756705510741}T^2 \\ &\quad + \frac{227539310412994}{37788783527553705}T + \frac{6694752830187523}{37788783527553705}. \end{aligned}$$

We test whether or not the system $Z'_1 \cap Z'_2 = \emptyset$ using Algorithm **Exact_Sign**. For $i = 1, 2$, construct expressions r_{R_i} and r_{I_i} s.t.

$$h_i^{(1)}(\theta) - h_i^{(2)}(\theta) = r_{R_i}(\Re\theta, \Im\theta) + r_{I_i}(\Re\theta, \Im\theta).$$

The root bounds for r_{R_1} , r_{I_1} , r_{R_2} and r_{I_2} are all smaller than 64 bits. Now, we apply precision-driven computation to approximate the coordinates of the roots to the (absolute) precision 128 bits. We list the values of the real and imaginary parts of $h_i^{(1)}(\theta)$ for $i = 1, 2$:

$$\begin{array}{c} \left(\left(\Re h_1^{(1)}(\theta), \Im h_1^{(1)}(\theta) \right), \left(\Re h_2^{(1)}(\theta), \Im h_2^{(1)}(\theta) \right) \right) \\ \left(\left(-1, -7.329 * 10^{-54} \right), \left(-0.428, 8.903 * 10^{-54} \right) \right) \\ \left(\left(1, 4.222 * 10^{-52} \right), \left(1, -4.449 * 10^{-52} \right) \right) \\ \left(\left(0.1429, 5.432 * 10^{-52} \right), \left(1.75, -4.250 * 10^{-52} \right) \right) \\ \left(\left(-1, 3.770 * 10^{-54} \right), \left(0.173, 1.231 * 10^{-54} \right) \right) \end{array}$$

and the values of the real and imaginary parts of $h_i^{(2)}(\theta)$ for $i = 1, 2$:

$$\begin{array}{c} \left(\left(\Re h_1^{(2)}(\theta), \Im h_1^{(2)}(\theta) \right), \left(\Re h_2^{(2)}(\theta), \Im h_2^{(2)}(\theta) \right) \right) \\ \left(\left(-1, -1.299 * 10^{-51} \right), \left(-0.614, 1.418 * 10^{-51} \right) \right) \\ \left(\left(1, 1.022 * 10^{-49} \right), \left(1, -1.065 * 10^{-49} \right) \right) \\ \left(\left(0.1429, -4.076 * 10^{-50} \right), \left(1.75, 3.203 * 10^{-50} \right) \right) \\ \left(\left(-1, 2.030 * 10^{-51} \right), \left(0.144, 4.200 * 10^{-52} \right) \right) \end{array}$$

For each of these, the imaginary component is small enough to indicate that the root is real. Furthermore, points

$$\left((1, 4.222 * 10^{-52}), (1, -4.449 * 10^{-52}) \right)$$

and

$$\left((1, 1.429 * 10^{-49}), (1, -1.065 * 10^{-49}) \right)$$

are identical because their difference is smaller than the root bounds for r_{R1} and r_{I1} (and we know $(1, 1)$ is a root). Similarly, points

$$\left((0.1429, 5.432 * 10^{-52}), (1.75, -4.250 * 10^{-52}) \right)$$

and

$$\left((0.1429, -4.076 * 10^{-50}), (1.75, 3.203 * 10^{-50}) \right)$$

are identical because their difference is smaller than the root bounds for r_{R2} and r_{I2} . On the other hand, the other pairs are distinct. Note also the other roots found in the two iterations, while different from iteration to iteration, are consistent with the positive dimensional component, $X = -1$ (although we cannot actually determine this component). Thus, we conclude that the zero set of F_1 consists of 2 isolated points and some positive dimensional components.

F_2 and F_3 : Singularities

Consider the system F_2 of two polynomials

$$\begin{aligned} f_{21} &= X^3 - 3X^2 + 3X - Y^2 + 2Y - 2, \\ f_{22} &= 2X + Y - 3 \end{aligned}$$

An elliptic curve $f_{21} = 0$ has a cusp at $(1, 1)$ and intersects with the line $f_{22} = 0$ at this point.

We obtain the exact RUR for the zero set of F_2 as

$$\begin{aligned} h(T) &= -T^2 - 5424051198T - 6426597229148218232, \\ h_1(T) &= -\frac{2}{963579587}T - \frac{2533312437}{963579587}, \\ h_2(T) &= \frac{4}{963579587}T + \frac{7957363635}{963579587} \end{aligned}$$

which immediately implies that F_2 has 2 real roots including $(1, 1)$.

Consider the system F_3 of two polynomials

$$\begin{aligned} f_{31} &= X^3 - 3X^2 - 3XY + 6X + Y^3 - 3Y^2 + 6Y - 5, \\ f_{32} &= X + Y - 2 \end{aligned}$$

Folium of Descartes $f_{31} = 0$ has a self-intersection at $(1, 1)$ and intersects with the line $f_{32} = 0$ at this point.

We obtain the exact RUR for the zero set of F_2 as

$$\begin{aligned} h(T) &= T + 3284000871, \\ h_1(T) &= -T - 3284000870, \\ h_2(T) &= -T - 3284000870 \end{aligned}$$

which immediately implies that F_3 has only 1 real roots including $(1, 1)$.

Because the intersections of both systems are at singular points, we cannot apply methods, such as MAPC, that depend on general position.

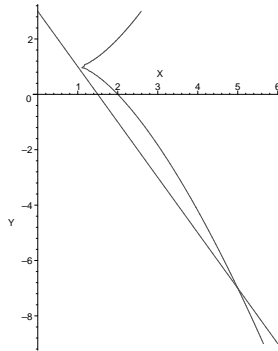


FIGURE 2. F2

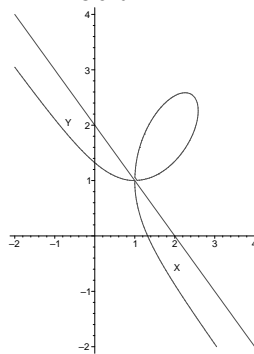


FIGURE 3. F3

F_4 : Complex Positive Dimensional Components

Consider the system F_4 of three polynomials

$$\begin{aligned} f_{41} &= Z - 1, \\ f_{42} &= X^2 + Y^2 + Z^2 - 4Z + 3, \\ f_{43} &= X^2 + Y^2 + Z^2 - 1 \end{aligned}$$

It is easy to see that the only real point on the zero set of F_4 is $(0, 0, 1)$. However, the exact RUR for the zero set of F_4 consists of 4 complex points each of which lies on some complex positive dimensional component (satisfying $X^2 + Y^2 = 0, Z = 1$). In general, there is no clear way to extract finitely many real points that lie on these complex positive dimensional components.

F_5 through F_8

Cases F_5 through F_8 are all drawn from cases encountered in an actual boundary evaluation computation. The source data is real-world data provided from the BRL-CAD [DM89] solid modeling system.

F_5 and F_6

Both F_5 and F_6 consist of an intersection of a line with an ellipse. Both have 2 roots and all roots are real.

Input System	F_5	F_6	F_7	F_8
# real roots by RUR	2	2	0	4
# roots of h	2	2	2	4
# real roots by MAPC	2	1	0	2
total time by RUR (sec)	0.0926	0.104	0.183	0.687
total time by MAPC (sec)	0.017	0.015	0.024	0.017

TABLE 2. Timings to Solve Systems F_5, F_6, F_7, F_8 by RUR and ESOLID/MAPC in seconds

F_7 : No Real Roots

The system F_7 consists of two ellipses. Rather than real intersections, these ellipses have 2 complex intersections. **Real_Roots** is used to distinguish real roots from the other roots.

F_8 : Burst of Coefficients of RUR

The system F_8 consists of two ellipses. F_8 has 4 roots and all of them are real.

For this example, we spent the most time computing coordinate polynomials. This was because the coefficients of the minimal polynomial and the coordinate polynomials become huge. This also slows down the following step which iteratively approximates the roots of the minimal polynomial.

5.1.2. *Summary of Timing Breakdowns.* While the examples shown above are not comprehensive, from these and other cases we have examined, we can draw the following conclusions:

- The performance of the method is reasonable for lower dimension/degree systems. However, for higher dimension/degree systems (e.g. F_4), the method is not practical.
- For lower dimension/degree systems, the most time consuming part of the algorithm is repeated computation of the determinant of the sparse resultant matrix. In fact, the size of the toric resultant matrix grows quite rapidly w.r.t. dimension/degree of the system in its current implementation.
- For higher dimension/degree systems, the most time consuming part is computing univariate polynomials forming the exact RUR, mainly because of their huge coefficients.
- Constructing the toric resultant matrix, and finding and evaluating over the roots of the minimal polynomial takes up an insignificant portion of the time.

5.2. RUR vs. MAPC. In this section, we show the timings for comparison of the exact RUR method with that used in the MAPC library within the ESOLID system. Note that the exact RUR method is able to find solutions in several cases where MAPC/ESOLID would fail (e.g. case F_1, F_2 , and F_3), and thus we can only compare a limited subset of cases.

It is important to note that the RUR implementation is a relatively straightforward implementation of the algorithm. For instance, no effort is made to control intermediate coefficient growth in several stages. The MAPC/ESOLID code, on the other hand, has been significantly optimized through the use of various speedup

techniques, such as floating-point filters and lazy evaluation approaches. It is likely that by aggressively applying such speedups to the RUR implementation, its performance can also be improved markedly. However, it is unlikely that even such improvements would change the overall conclusions.

One of the basic routines in ESOLID / MAPC is finding real intersections of 2 monotone pieces of curves.

A real algebraic number ξ is specified by a pair polynomials (f_1, f_2) and the region R s.t. ξ is the only intersection of f_1 and f_2 in R . We could naively use the exact RUR here to specify algebraic numbers.

We compare only the last steps of root isolation in ESOLID / MAPC with the exact RUR method. Because ESOLID / MAPC only finds roots over a particular region, it does not necessarily find all the roots of the system. This is in contrast to the exact RUR, which finds all roots (and would then select only those in the region). This describes the difference in the number of roots found in F_6 and F_8 .

In addition, the exact RUR loses efficiency by finding all the complex roots while ESOLID / MAPC finds only real roots. This phenomenon can be observed in the example of F_7 . The size of the coefficients of polynomials in the exact RUR grow independent of the location of the roots.

From these cases, the clear conclusion is that for generic cases that a method such as those MAPC/ESOLID can handle, the RUR has an unacceptably high performance cost. For this reason, it will be best to use the RUR in implementations only in a hybrid fashion, when the other methods will fail. An important caveat should be considered, in that our RUR implementation has not been fully optimized. Such optimizations should increase RUR performance significantly, though we still believe that fundamental limitations (such as finding all roots, including complex ones) will make the RUR less efficient in most generic cases encountered in practice.

6. Conclusion

We have presented the Rational Univariate Reduction as a method for achieving exact geometric computation in situations that require solutions to systems of polynomials. We compute the RUR

exactly. In addition, we have presented an approach for exact sign determination with root bounds for complex algebraic numbers, extending previous approaches that apply only to real numbers. This allows us to use the RUR for geometric operations, including determining whether or not there are positive dimensional components, and distinguishing real roots.

Our method is significant in the following sense:

- The method and generated answers are readily adapted to exact computation.
- The method finds complex roots and identifies real roots if necessarily.
- The method works even if the set of roots has some positive dimensional component,
- The method is entirely factorization-free and Sturm-free.

Finally, we have implemented the RUR approach described and presented the timing breakdown for various stages for a selected set of problems. These timings highlight the computational bottlenecks and give indications of useful avenues for future work.

6.1. Future Work. A primary avenue of future development will be in finding ways to optimize the computation and use of the RUR, particularly in reference to problems from specific domains. Experience has shown that by aggressively applying filtering and other speedup techniques, many exact implementations can be made far more efficient [KCF⁺02]. There are numerous potential avenues for such speedups, and we describe a couple of them below.

Our experiments show that for low-dimensional cases, most of the computation time is dedicated to evaluating the determinant of the sparse resultant matrix. There are thus two immediately obvious issues to tackle:

- Use some algorithm which produces sparse resultant matrices of smaller size [EC95] [DE02]. Although the matrix construction is not time-consuming, the size of the sparse resultant matrices directly affects the performance of the next step, namely, repeated computations of the determinant of the matrix.
- Computing the determinant of a sparse resultant matrix is the most time consuming step. Currently, we use a standard Gaussian elimination method implemented in multi-precision arithmetic. Improvements might include taking advantage of the sparse structure of the matrix itself, or finding the determinant using a filtered approach, resulting in only partially determined, but still exact, polynomial coefficients. Such coefficients would allow faster computation in most cases, and could still be later refined to greater or complete accuracy in the few cases where it is needed.
- We would like to put some controls over the size of coefficients of univariate polynomials forming the RUR. Experiments showed that, for higher dimensional cases, univariate polynomial ring operations tends to be the most expensive, because of the growth of coefficients. First, we should choose generic values more carefully based on the shaper estimate for bit-complexity of the subroutines. Next, we should make better choices for subroutines, e.g., subresultant method instead of Euclidean algorithm, etc., as this tends to dominate running time for higher degrees.

Finally, we are in the process of integrating the RUR completely into a solid modeling system [?] to support fully exact and robust geometric computation. We have shown that (with the current implementation) the naive use of the exact RUR is not attractive in terms of performance. However, we have also shown that the exact RUR is able to handle degeneracies. A practical solution would be to develop a hybrid system, incorporating both the RUR and a general-position system solver. Determining precisely how to put together such a hybrid system is a challenge for future work.

References

- [Abe73] O. Aberth, *Iteration methods for finding all zeros of a polynomial simultaneously*, Mathematics of Computation **27** (1973), no. 122, 339 – 344.
- [ARD02] P. Aubry, F. Rouillier, and M. Safey El Din, *Real solving for positive dimensional systems*, Journal of Symbolic Computation **34** (2002), no. 6, 543 – 560.
- [BCSS97] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and real computation*, Springer, 1997.
- [Ber75] D. N. Bernstein, *The number of roots of a system of equations*, Functional Analysis and its Applications **9** (1975), no. 2, 183 – 185.

- [BF00] D. A. Bini and G. Fiorentino, *Design, analysis, and implementation of a multiprecision polynomial rootfinder*, Numerical Algorithms **23** (2000), no. 2, 127 – 173.
- [BFMS97] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra, *A strong and easily computable separation bound for arithmetic expressions involving square roots*, Proc. of 8th ACM-SIAM Symposium on Discrete Algorithms, ACM, 1997, pp. 702 – 709.
- [BFMS99] ———, *Exact efficient computational geometry made easy*, Proc. of 15th Annual Symposium on Computational Geometry, ACM, 1999, pp. 341 – 350.
- [BFMS00] ———, *A strong and easily computable separation bound for arithmetic expressions involving radicals*, Algorithmica **27** (2000), no. 1, 87 – 99.
- [Bin96] D. A. Bini, *Numerical computation of polynomial zeros by means of Aberth’s method*, Numerical Algorithms **13** (1996), no. 3/4, 179 – 200.
- [BMS96] C. Burnikel, K. Mehlhorn, and S. Schirra, *The LEDA class real number*, Tech. Report MPI-I-96-1-001, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, 1996.
- [BPR03] S. Basu, R. Pollack, and Marie-Françoise Roy, *Algorithms in real algebraic geometry*, Springer, 2003.
- [Can90] J. F. Canny, *Generalized characteristic polynomials*, Journal of Symbolic Computation **9** (1990), no. 3, 241 – 250.
- [CE93] J. F. Canny and I. Z. Emiris, *An efficient algorithm for the sparse mixed resultant*, Proc. of 10th AAEECC, LNCS 673, Springer, 1993, pp. 89 – 104.
- [CE00] ———, *A subdivision-based algorithm for the sparse resultant*, Journal of ACM **47** (2000), no. 3, 417 – 451.
- [CK03] A. D. Chtcherba and D. Kapur, *Exact resultants for corner-cut unmixed multivariate polynomial systems using the dixon formulation*, Journal of Symbolic Computation **36** (2003), no. 3/4, 289 – 315.
- [CLO98] D. Cox, J. Little, and D. O’Shea, *Using algebraic geometry*, Springer, 1998.
- [D’A02] C. D’Andrea, *Macaulay style formulas for sparse resultants*, Transactions of the AMS **354** (2002), no. 7, 2595 – 2629.
- [DE01] C. D’Andrea and I. Z. Emiris, *Computing sparse projection operators*, Symbolic Computation: Solving equations in Algebra, Geometry and Engineering (E. L. Green et al., ed.), Contemporary Mathematics, Vol. 286, AMS, 2001, pp. 121 – 139.
- [DE02] ———, *Hybrid sparse resultant matrices for bivariate polynomials*, Journal of Symbolic Computation **33** (2002), no. 5, 587 – 608.
- [DE03] ———, *Sparse resultant perturbation*, Algebra, Geometry, and Software (M. Joswing and N. Takayama, eds.), Springer, 2003, pp. 93 – 107.
- [DM89] P. C. Dykstra and M. J. Muuss, *The BRL-CAD package an overview*, Tech. report, Advanced Computer Systems Team, Ballistics Research Laboratory, Aberdeen Proving Ground, MD, 1989, <http://ftp.arl.mil/brlcad/>.
- [DY95] T. Dubé and C. Yap, *The exact computation paradigm*, Computing in Euclidean Geometry (D. Du and F. Hwang, eds.), Lecture Notes on Computing, World Scientific, 2nd ed., 1995, pp. 452 – 492.
- [EC91] I. Z. Emiris and J. F. Canny, *A general approach to removing degeneracies*, Proc. of 32nd IEEE Symposium on the Foundations of Computer Science, IEEE, 1991, pp. 405 – 413.
- [EC92] ———, *An efficient approach to removing geometric degeneracies*, Proc. of 8th Annual Symposium on Computational Geometry, ACM, 1992, pp. 74 – 82.
- [EC95] ———, *Efficient incremental algorithm for the sparse resultant and the mixed volume*, Journal of Symbolic Computation **20** (1995), no. 2, 117 – 149.
- [ECS97] I. Z. Emiris, J. F. Canny, and R. Seidel, *Efficient perturbations for handling geometric degeneracies*, Algorithmica **19** (1997), no. 1/2, 219 – 242.
- [EM90] H. Edelsbrunner and E. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Transactions on Graphics **9** (1990), no. 1, 66 – 104.
- [Emi96] I. Z. Emiris, *On the complexity of sparse elimination*, Journal of Complexity **12** (1996), no. 2, 134 – 166.
- [Emi02] ———, *Enumerating a subset of the integer points inside a Minkowski sum*, Computational Geometry **22** (2002), no. 1-3, 143 – 166.

- [Emi03] ———, *Discrete geometry for algebraic elimination*, Algebra, Geometry, and Software (M. Joswing and N. Takayama, eds.), Springer, 2003, pp. 77 – 91.
- [For97] S. Fortune, *Polyhedral modeling with multiprecision integer arithmetic*, Computer-Aided Design **29** (1997), no. 2, 123 – 133.
- [GLS01] M. Giusti, G. Lecerf, and B. Salvy, *A Gröbner free alternative for polynomial system solving*, Journal of Complexity **17** (2001), no. 1, 154 – 211.
- [GV91] L. González-Vega, *A subresultant theory for multivariate polynomials*, Proc. of ISSAC '91, ACM, 1991, pp. 79 – 85.
- [GV97] L. Gonzalez-Vega, *Implicitization of parametric curves and surfaces by using multi-dimensional newton formulae*, Journal of Symbolic Computation **23** (1997), no. 2/3, 137 – 151.
- [GVR99a] L. Gonzalez-Vega, F. Rouillier, and M-F. Roy, *Symbolic recipes for polynomial system solving*, Some tapas of computer algebra (A. M. Cohen, H. Cuypers, and H. Sterk, eds.), Algorithms and computation in mathematics, Vol. 4, Springer, 1999, pp. 34 – 65.
- [GVR99b] ———, *Symbolic recipes for real solutions*, Some tapas of computer algebra (A. M. Cohen, H. Cuypers, and H. Sterk, eds.), Algorithms and computation in mathematics, Vol. 4, Springer, 1999, pp. 121 – 167.
- [HHK89] C. M. Hoffman, J. E. Hopcroft, and M. S. Karasick, *Robust set operations on polyhedral solids*, IEEE Computer Graphics and Applications **9** (1989), no. 6, 50 – 59.
- [Hof89] C. M. Hoffman, *The problems of accuracy and robustness in geometric computation*, IEEE Computer **22** (1989), no. 3, 31 – 41.
- [KCF⁺02] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha, *ESOLID: A system for exact boundary evaluation*, Proc. of 7th ACM Symposium on Solid Modeling and Applications, ACM, 2002, pp. 23 – 34.
- [KCF⁺04] ———, *ESOLID - a system for exact boundary evaluation*, Computer-Aided Design **36** (2004), no. 2, 175 – 193.
- [KCMK99] J. Keyser, T. Culver, D. Manocha, and S. Krishnan, *MAPC: A library for efficient and exact manipulation of algebraic points*, Proc. of 15th Annual Symposium on Computational Geometry, ACM, 1999, pp. 360 – 369.
- [KCMK00] ———, *Efficient and exact manipulation of algebraic points and curves*, Computer-Aided Design **32** (2000), no. 11, 649 – 662.
- [Key00] John C. Keyser, *Exact boundary evaluation for curved solids*, Ph.D. thesis, Department of Computer Science, University of North Carolina, Chapel Hill, NC, 2000.
- [Khe03] A. Khetan, *The resultant of an unmixed bivariate system*, Journal of Symbolic Computation **36** (2003), no. 3/4, 425 – 442.
- [Kho78] A. G. Khovanskii, *Newton polyhedra and the genus of complete intersections*, Functional Analysis and its Applications **12** (1978), no. 1, 51 – 61.
- [KKM99a] J. Keyser, S. Krishnan, and D. Manocha, *Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: I - representations*, Computer Aided Geometric Design **16** (1999), no. 9, 841 – 859.
- [KKM99b] ———, *Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: II - computation*, Computer Aided Geometric Design **16** (1999), no. 9, 861 – 882.
- [KLY99] V. Karamcheti, C. Li, and C. Yap, *A Core library for robust numerical and geometric computation*, Proc. of 15th Annual Symposium on Computational Geometry, ACM, 1999, pp. 351 – 359.
- [Kro31] L. Kronecker, *Leopold kronecker's werke*, Teubner, 1895 - 1931.
- [Kus76] D. N. Kushnirenko, *Newton polytopes and the Bezout theorem*, Functional Analysis and its Applications **10** (1976), no. 3, 82 – 83.
- [Lec02] G. Lecerf, *Quadratic newton iteration for systems with multiplicity*, Journal of Foundations of Computational Mathematics **2** (2002), no. 3, 247 – 293.
- [LY01] C. Li and C. Yap, *A new constructive root bound for algebraic expressions*, Proc. of 12th ACM-SIAM Symposium on Discrete Algorithms '01, ACM, 2001, pp. 476 – 505.
- [MN99] S. Mehlhorn and M. Näher, *Leda - a platform for combinatorial and geometric computing*, Cambridge University Press, 1999.

- [MOBP94] D. Michelucci M. O. Benouamer and B. Peroche, *Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation*, Computer-Aided Design **26** (1994), no. 6, 403 – 416.
- [MS99] N. Mignotte and D. Ștefănescu, *Polynomials: An algebraic approach*, Springer, 1999.
- [PS93] P. Pedersen and B. Sturmfels, *Product formulas for resultants and chow forms*, Mathematische Zeitschrift **214** (1993), no. 3, 377 – 396.
- [PY03] S. Pion and C. Yap, *Constructive root bound for k -ary rational input numbers*, Proc. of 19th Annual Symposium on Computational Geometry, ACM, 2003, pp. 256 – 263.
- [Roj94] J. M. Rojas, *A convex geometric approach to counting the roots of a polynomial system*, Theoretical Computer Science **133** (1994), no. 1, 105 – 140.
- [Roj96] ———, *Counting affine roots of polynomial systems via pointed Newton polytopes*, Journal of Complexity **12** (1996), no. 2, 116 – 133.
- [Roj97] ———, *Toric laminations, sparse generalized characteristic polynomials, and a refinement of Hilbert’s tenth problem*, Foundations of Computational Mathematics (F. Cucker and M. Shub, eds.), Springer, 1997, pp. 369 – 381.
- [Roj98] ———, *Intrinsic near quadratic complexity bounds for real multivariate root counting*, Proc. of 6th Annual European Symposium on Algorithms, LNCS 1461, Springer, 1998, pp. 127 – 138.
- [Roj99a] ———, *Solving degenerate sparse polynomial systems faster*, Journal of Symbolic Computation **28** (1999), no. 1/2, 155 – 186.
- [Roj99b] ———, *Toric intersection theory for affine root counting*, Journal of Pure and Applied Algebra **136** (1999), no. 1, 67 – 100.
- [Roj00] ———, *Algebraic geometry over four rings and the frontier to tractability*, Hilbert’s tenth problem : relations with arithmetic and algebraic geometry (J. Denef et al., ed.), Contemporary Mathematics, Vol. 270, AMS, 2000, pp. 275 – 321.
- [Roj03] ———, *Why polyhedra matter in non-linear equation solving*, Topics in Algebraic Geometry and Geometric Modeling (R. Goldman and R. Krasauskas, eds.), Contemporary Mathematics, Vol. 334, AMS, 2003, pp. 293 – 320.
- [Roj04] ———, *Arithmetic multivariate Descartes’ rule*, American Journal of Mathematics **126** (2004), no. 1, 1 – 30.
- [Rou99] F. Rouillier, *Solving zero-dimensional systems through the rational univariate representation*, Applicable Algebra in Engineering, Communication and Computing **9** (1999), no. 5, 433 – 461.
- [Roy96] M-F. Roy, *Computation in real algebraic geometry*, The Mathematics of Numerical Analysis (J. Renegar et al., ed.), Lectures in Applied Mathematics, Vol. 32, AMS, 1996, pp. 701 – 714.
- [Sei94] R. Seidel, *The nature and meaning of perturbations in geometric computing*, Proc. of 11th Annual Symposium on Theoretical Aspects of Computer Science '94, LNCS 775, Springer, 1994, pp. 3 – 17.
- [SI89] K. Sugihara and M. Iri, *A solid modelling system free from topological inconsistency*, Journal of Information Processing **12** (1989), no. 4, 380 – 393.
- [Stu94] B. Sturmfels, *On the Newton polytope of the resultant*, Journal of Algebraic Combinatorics **3** (1994), no. 2, 207 – 236.
- [Stu02] ———, *Solving systems of polynomial equations*, AMS, 2002.
- [SV00] A. J. Sommese and J. Verschelde, *Numerical homotopies to compute generic points on positive dimensional components*, Journal of Complexity **16** (2000), no. 3, 572 – 602.
- [SVW01] A. J. Sommese, J. Verschelde, and C. W. Wampler, *Numerical decomposition of the solution sets of polynomial systems into irreducible components*, SIAM Journal of Numerical Analysis **38** (2001), no. 6, 2022 – 2046.
- [SVW03] ———, *Numerical irreducible decomposition using phcpack*, Algebra, Geometry, and Software (M. Joswing and N. Takayama, eds.), Springer, 2003, pp. 109 – 129.
- [SW96] A. J. Sommsese and C. W. Wampler, *Numerical algebraic geometry*, The Mathematics of Numerical Analysis (J. Renegar et al., ed.), Lectures in Applied Mathematics, Vol. 32, AMS, 1996, pp. 749 – 763.
- [Yap90] C. Yap, *Symbolic treatment of geometric degeneracies*, Journal of Symbolic Computation **10** (1990), no. 3/4, 349 – 370.

- [Yap97] ———, *Towards exact geometric computation*, Computational Geometry **7** (1997), no. 1, 3 – 23.
- [Yu91] J. Yu, *Exact arithmetic solid modeling*, Ph.D. thesis, Department of Computer Science, Purdue University, West Lafayette, IN, 1991.

DEPARTMENT OF COMPUTER SCIENCE, TEXAS A&M UNIVERSITY, COLLEGE STATION, TX 77843-3112

Current address: Department of Computer Science, Texas A&M University, College Station, TX 77843-3112

E-mail address: keyser@cs.tamu.edu

DEPARTMENT OF MATHEMATICS, TEXAS A&M UNIVERSITY, COLLEGE STATION, TX 77843-3368

Current address: Department of Mathematics, Texas A&M University, College Station, TX 77843-3368

E-mail address: rojas@math.tamu.edu

DEPARTMENT OF COMPUTER SCIENCE, TEXAS A&M UNIVERSITY, COLLEGE STATION, TX 77843-3112

Current address: Department of Computer Science, Texas A&M University, College Station, TX 77843-3112

E-mail address: kouchi@cs.tamu.edu