The Exact Rational Univariate Representation and its Application

Koji Ouchi [a] John Keyser [a] J. Maurice Rojas [b]

[a] *Department of Computer Science, 3112 Texas A&M University, College Station, TX, 77843-3112*

[1] [b] *Department of Mathematics, 3368 Texas A&M University, College Station, TX, 77843-3368*

Abstract

We describe a method, based on 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 the 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.

Key words: Rational Univariate Representation, Sparse Resultants, Exact Computation, Algebraic Numbers, Solid Modeling, Degeneracies.

PACS:

[1] This work funded in part by NSF awards DMS-0138446 and CCR-0220047

1 Introduction

Solving systems of polynomials is a common operation in many geometric applications, particularly those involving curved objects. Polynomials regularly describe the relationships between even basic geometric objects, for example the (squared) distance between two points. For more complex curved geometric objects, polynomials are often used to describe the actual shapes. The solutions to systems of polynomials thus becomes a key operation in numerous geometric applications.

As an example, finding the faces, edges, and vertices in solid modeling applications such as boundary evaluation usually involves finding common solutions to the sets of polynomials that describe the surfaces of the solids. This root-finding procedure tends to dominate both the efficiency and the robustness of the overall boundary evaluation algorithm. Such root-finding is highly prone to robustness problems arising from both numerical error and degenerate data. In an exact boundary evaluation system, a typical boundary evaluation operation can involve finding hundreds of solutions to systems of equations, totalling more than 90% of the overall operation time [1]. While other geometric operations might not be so extreme, finding solutions to polynomial systems often remains a key to both robustness and efficiency.

Many techniques have been used to find solutions for such geometric systems. Techniques that are inexact (e.g. use standard floating-point computation) regularly encounter robustness problems due to buildup and propagation of roundoff or approximation error. For this reason, some researchers have turned to exact methods for computing solutions; our work falls into this category.

Unfortunately, for many exact techniques, performance becomes the major drawback, and degenerate situations may not be handled adequately.

We propose the use of the *Rational Univariate Reduction* (RUR), also referred to as the Rational Univariate Representation, as a method for finding and working with solutions of polynomial systems arising in geometric computations. Its ability to handle degenerate situations smoothly gives it a clear advantage over several prior representation schemes [2].

1.1 Overview of the RUR

Each coordinate of a finite subset of the set of roots of a system of polynomials can be represented by a univariate polynomial evaluated at a root of some other univariate polynomial. This representation for the roots of a system is called the Rational Univariate representation and this reduction is called the Rational Univariate Reduction. Provided an exact RUR is given, we can perform exact computation over (the coordinates of) the roots of a system.

If there are a finite number of solutions to the system, the RUR approach determines these roots exactly. Our method works even for positive-dimensional systems, i.e., the method finds some finite subset of the set of roots which contains at least one point from every irreducible component.

Using the RUR for geometric computation thus requires several steps:

- The input system, described as polynomials with rational coefficients, is reduced to the RUR. The RUR consists of a univariate *minimal polynomial*, along with n univariate *coordinate polynomials*, one for each dimension (vari-

able) in the system.

- The roots (real and complex) of the minimal polynomial are found and are represented using bounding intervals.
- These roots are evaluated within the n coordinate polynomials, giving n-dimensional bounding boxes around each root.
- For a given query/predicate, we can determine the root bounds that will guarantee correct sign evaluation, thus determining the maximum precision for which we must approximate the roots.

1.2 Main Results

We describe the use of the Rational Univariate Reduction as a technique for solving systems of equations arising in geometric computation. We have implemented the RUR for systems with rational coefficients, based on an *exact* implementation of the sparse resultant. We have applied our exact implementation to a series of geometric problems, motivated by cases arising in boundary evaluation, and have analyzed its performance and applicability in these situations.

The primary new results presented in this paper are:

- A complete description of a method for using the RUR in an exact geometric computation framework.
- An extension of the precision-driven computation model to handle input expressed as complex algebraic numbers, and propagation of complex values through the graph (Sections 3.2.3 and 3.2.4).
- An extension of the RUR method to find solutions such as those at the

origin, and those to non-square systems (Section 3.3).

- A description of how the RUR approach can be applied to geometric problems (Section 4.1), particularly boundary evaluation problems (Section 4.2.3) involving degeneracies (Section 4.3).
- A breakdown and brief analysis of timing results from an implementation of our RUR method (Section 5).

1.3 Organization

The rest of the paper is organized as follows:

- Section 2 gives a brief summary of related work.
- Section 3 gives a detailed description of the RUR and hot 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.

1.4 Notation

We will use the following notation in the remainder of this paper:

$\mathbb{Q}^*$ denotes the multiplicative group (or just the set) consisting of non-zero rational numbers. Similarly, $(\mathbb{C}^*) = \mathbb{C} \setminus \{0\}$ for the complex numbers.

$\mathbb{Q}[T]$ and $\mathbb{Q}[X_1, \ldots, X_n]$ denote the ring of univariate polynomials in variable T with rational coefficients, and the ring of polynomials in variables $X_1, \ldots, X_n$

with rational coefficients, respectively.

2 Related Work

The RUR method for solving systems of polynomials has existed for centuries, but has been explored for computational purposes only recently.

If a system is zero-dimensional (i.e. it has only finitely many roots), then the RUR can be computed via the multiplication table method. [3] [4]. The quotient ring $\mathbb{Q}[X_1, \ldots, X_n] / \langle f_1, \ldots, f_m \rangle$ is a finite dimensional vector space over $\mathbb{Q}$. The effect of multiplication of a polynomial f is an endomorphism and its matrix representation (w.r.t. some fixed basis) is called the multiplication table for f. The RUR of the system can be deduced from the traces of multiplication tables for some polynomials in the quotient ring [5] [6]. The algorithm uses a normal form basis which usually requires Gröbner basis computation.

Recently, a Gröbner-free algorithm has been proposed [7] in which a geometric resolution of a zero-dimensional system is computed by iterations on partial solutions. The most recent work even handles some systems with multiple roots [8]. Because of their iterative nature, it is hard to apply these algorithm to exact computation.

For square systems, the u-resultant method can be used to compute the RUR. The most traditional u-resultant method finds the RUR for the roots of the homogeneous system in projective space, and works only for zero-dimensional systems. Several attempts have been made to generalize the method. Among them, one method that relies on the u-resultant of some perturbed version of the system, called a GCP (Generalized Characteristic Polynomial), will find

all the "finite" isolated common roots (in $(\mathbb{C}^*)^n$) of the homogenized system even if there are infinitely many common roots at infinity [9].

Our method for computing the RUR uses the "sparse" version of the u-resultant [10]. The sparse resultant algorithm is historically the first practical algorithm to compute multivariate resultants [11] [12]. Various improvements have been made in computation of the sparse resultant [13] [14].

The root bound approach to exact sign determination for algebraic numbers has been proposed [15] [16] and implemented for real algebraic numbers in the libraries `LEDA` [17] and `Core` [18]. Burnikel et al. [19] [20] [21] established the rules for computing the root bounds for certain algebraic numbers (those defined by algebraic expressions evaluated at integers). Yap and Li [22] improve these bounds and also extend them to real algebraic numbers specified as roots of a polynomial in $\mathbb{Z}[T]$. Our work extends such approaches to work with complex algebraic numbers specified as roots of polynomials.

While the importance of robust computation has long been recognized [23], the application of exact computation to solid modeling has been quite limited. For polyhedra, Fortune provided one of the first completely exact approaches [24]. Notably, his work also addressed a limited number of degenerate cases. Computations on polyhedra require only rational number representations, however. Although the algebraic issues involved in extending the computation to curved surfaces are well understood [25], almost no practical implementations have been attempted. We know of only one exact implementation for boundary evaluation on curved solids, ESOLID [1], and it fails for most degeneracies. A motivation of our work has been to provide tools that will allow ESOLID to detect degenerate cases.

In terms of computational complexity, it can be shown our algorithm to compute the exact RUR fits to the Polynomial Hierarchy, assuming the generalized Riemmann's hypothesis [26] [27]. However, it is impossible to give a tight bound on the complexity for the exact sign determination for algebraic numbers because an iterative method is used to approximate the roots of the minimal polynomial.

3 Exact Rational Univariate Representation for Roots of Multivariate Polynomial Systems with Rational Coefficients

Consider a system of m polynomials $f_1, \ldots, f_m$ in n variables with coefficients belonging to the field of rational numbers. Let Z be the set of all the common roots of the system.

Let Z' be a finite subset of Z. Consider the finite extension L of the field $\mathbb{Q}$ of rational numbers obtained by adjoining all the coordinates of points in Z'. By the primitive element theorem, $L = \mathbb{Q}(\theta)$ for some $\theta \in L$. Furthermore, letting h be the minimal polynomial for θ over $\mathbb{Q}$, $\mathbb{Q}(\theta) \cong \mathbb{Q}[T]/h$. Thus, every coordinate of points in Z' is represented as a univariate polynomial with rational coefficients: there exist $h, h_1, \ldots, h_n \in \mathbb{Q}[T]$ s.t.

$$Z' = \{(h_1(\theta), \ldots, h_n(\theta)) \in \mathbb{C}^n \mid \theta \in \mathbb{C} \text{ with } h(\theta) = 0\}. \tag{1}$$

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 partic-

ular, if Z is zero-dimensional then $Z' = Z$.

In section 3.1, we give an overview of the RUR for $Z' \cap (\mathbb{C}^*)^n$ when a system is square. We also discuss how this can be implemented exactly. 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 RUR for Square Systems

In this section, we describe the exact algorithm to compute the RUR for the non-zero roots of a square system. The minimal polynomial in the RUR is derived from the toric perturbation which is a generalization of the "sparse" u-resultant for the system. We begin by summarizing the sparse resultant, followed by a summary of the toric perturbation. We then describe the RUR computation.

3.1.1 Sparse Resultants

We have developed a strictly exact implementation of the sparse resultant algorithm [11] [12], summarized here.

Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$. 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 lattice points in n-dimensional space. Hence

$$f = \sum_{a \in A} c_a X^a,\ c_a \in \mathbb{Q}^* \text{ where } X^a = X_1^{a_1} \cdots X_n^{a_n} \text{ for } a = (a_1, \ldots, a_n). \quad (2)$$

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

$$\mathbf{c}_i = (c_{ia} \in \mathbb{Q} \mid a \in A_i) \text{ s.t. } f_i = \sum_{a \in A_i} c_{ia} X^a,\ \ i = 0, 1, \ldots, n. \quad (3)$$

For such a system, there exists a unique (up to sign) irreducible polynomial $\mathrm{Res}_{A_0, A_1, \ldots, A_n}(\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n) \in \mathbb{Z}[\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n]$, called the *sparse resultant*, s.t.

$$\begin{aligned} &\text{the system } (f_0, f_1, \ldots, f_n) \text{ has a common root in } (\mathbb{C}^*)^n \\ &\implies \mathrm{Res}_{A_0, A_1, \ldots, A_n}(\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n) = 0. \end{aligned} \quad (4)$$

We also write $\mathrm{Res}_{A_0, A_1, \ldots, A_n}(f_0, f_1, \ldots, f_n)$ for $\mathrm{Res}_{A_0, A_1, \ldots, A_n}(\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n)$.

To determine the sparse resultant, we implement a version of the approach outlined by Canny and Emiris [11,12]. This algorithm constructs a square matrix N, called the *sparse resultant matrix* or *Newton matrix*, whose determinant is some multiple of the sparse resultant. The elements of N are the coefficients of $f_0, f_1, \ldots, f_n$. The rows and columns of N are labeled by monomials, or equivalently, lattice points belonging to the supports $A_0, A_1, \ldots, A_n$ 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 the coefficients, $\mathbf{c}_i$, of the input polynomials. It follows that the total degree of the determinant with respect to the coefficients of polynomial f_i, $\deg_{\mathbf{c}_i} \det N$, is well-defined.

The n-tuples used to label the rows and columns of N are chosen according

to the *mixed-subdivision* [11] [12] of the *Minkowski sum* of the convex hulls $Q_0, Q_1, \ldots, Q_n$ of $A_0, A_1, \ldots, A_n$. The numbers of those n-tuples (and the degree of $\det N$ as well) is predicted in terms of the *mixed-volume* [11,12] for the Minkowski sum of $Q_0, Q_1, \ldots, Q_n$. More precisely, writing MV_{-i} for the mixed-volume for the Minkowski sum of $Q_0, Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_n$, it is shown that [11,12]:

$$\deg_{\mathbf{c}_0} \det N = \mathrm{MV}_{-0} \text{ and } \deg_{\mathbf{c}_i} \det N \geq \mathrm{MV}_{-i},\ i = 1, \ldots, n. \tag{5}$$

The equalities hold if $\det N$ is the sparse resultant [28].

The algorithm is written as follows:

Algorithm: Sparse Resultant Matrix

Input: $A_0, A_1, \ldots, A_n \subseteq \mathbb{Z}^n$.

Output: N and $\mathrm{MV}_{-0}, \mathrm{MV}_{-1}, \ldots, \mathrm{MV}_{-n}$.

1. Compute the convex hulls $Q_0, Q_1, \ldots, Q_n$ of $A_0, A_1, \ldots, A_n$.

2. Compute the mixed-subdivision of the Minkowski sum Q of $Q_0, Q_1, \ldots, Q_n$.

2-1. Choose $\delta \in \mathbb{Q}^n$.

2-2. For every lattice point p lying in the interior of $Q + \delta$, express p as the Minkowski sum of points in $A_0, A_1, \ldots, A_n$ to decide the cell of the mixed subdivision of Q in which p lies.

2-3. Test whether or not δ is generic. If not then go to step 2-1.

3. Label the rows and columns of N.

4. Compute $\mathrm{MV}_{-0}, \mathrm{MV}_{-1}, \ldots, \mathrm{MV}_{-n}$.

We implement this algorithm exactly by using multi-precision rational arithmetic. Steps 1 and 2-2 reduce to linear programming problems, which are computed exactly using rational arithmetic. The genericity test in step 2-3

reduces to computation of the dimension of a cell of the mixed subdivision of Q, or equivalently, the number of basis elements needed to span the cell. Provided the mixed subdivision Q is computed exactly and δ is chosen generically, step 4 just counts the number of lattice points in the cells. Hence, by using multi-precision rational arithmetic we guarantee that the results are exact.

3.1.2 Toric Perturbations

We describe the toric perturbation for a square system and present our exact algorithm to compute the RUR for roots (with non-zero coordinates) of a square system.

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

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

$$\mathrm{TGCP}(s, \mathbf{u}) = \mathrm{Res}_{A_0, A_1, \ldots, A_n}(f_0, f_1 - sf_1^*, \ldots, f_n - sf_n^*) \in \mathbb{Q}[s][\mathbf{u}]. \qquad (6)$$

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

Theorem 3.1 [10]

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

(1) If $(\zeta_1, \ldots, \zeta_n) \in (\mathbb{C}^*)^n$ *is an isolated common root of* $f_1, \ldots, f_n$ *then* $u_0 + u_1\zeta_1 + \cdots + u_n\zeta_n$ *is a linear factor of* $\mathrm{Pert}_{A_0}(\mathbf{u})$.

(2) $\mathrm{Pert}_{A_0}(\mathbf{u})$ *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* $\mathrm{Pert}_{A_0}(\mathbf{u})$ *corresponding to a root* $(\zeta_1, \ldots, \zeta_n) \in W \subseteq Z$.

Thus, with a suitable choice of $f_1^*, \ldots, f_n^*$, 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 $\mathrm{Pert}_{A_0}(\mathbf{u})$ by setting u_0 to T and specializing $u_1, \ldots, u_n$ to some values. Immediately from (5),

Corollary 3.2

$$\deg_{u_0} \mathrm{Pert}_{A_0}(\mathbf{u}) = \deg_{\mathbf{u}} \mathrm{Res}_{A_0}(f_0, f_1 - sf_1^*, \ldots, f_n - sf_n^*) = \mathrm{MV}_{-0}, \quad (7)$$

$$\deg_s \mathrm{TGCP}(s, \mathbf{u}) = \sum_{i=1}^{n} \mathrm{MV}_{-i} \leq \dim N - \mathrm{MV}_{-0}. \quad (8)$$

3.1.3 The RUR Computation

Our exact algorithm to compute the RUR for roots having non-zero coordinates of a square system is written as follows:

Algorithm: RUR_square

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

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

1. Use **Sparse Resultant Matrix** to compute the sparse resultant matrix N and MV_{-0} for the system of polynomials with supports $A_0, A_1, \ldots, A_n$.

2. Compute TGCP $(s, \mathbf{u})$.

2-1. Choose generic $u_0, u_1, \ldots, u_n \in \mathbb{Q}$ and $f_1^*, \ldots, f_n^* \in \mathbb{Q}[X_1, \ldots, X_n]$.

2-2. Compute some multiple of TGCP $(s, \mathbf{u})$.

2-3. If TGCP $(s, \mathbf{u}) \equiv 0$ then go to 2-1.

3. Compute $h(T) = \mathrm{Pert}_{A_0}(T, u_1, \ldots, u_n)$ where $\mathrm{Pert}_{A_0}(\mathbf{u})$ is the non-zero coefficients of the lowest degree term in (the multiple of) TGCP $(s, \mathbf{u})$.

4. For $i = 1, \ldots, n$ do:

4-1. Compute $p_i^{\pm}(t) = \mathrm{Pert}_{A_0}(t, u_1, \ldots, u_{i-1}, u_i \pm 1, u_{i+1}, \ldots, u_n)$.

4-2. Compute the square-free parts $q_i^{\pm}(t)$ of $p_i^{\pm}(t)$.

4-3. Compute a linear gcd $g(t)$ of $q_i^{-}(t)$ and $q_i^{+}(2T - t)$ by using the first subresultant method [9] [29].

4-4. Set $h_i(T) = -T - \dfrac{\text{the linear term of } g(t)}{\text{the constant term of } g(t)} \bmod h(T)$.

Step 1 determines the sparse resultant matrix, but with the entries still undetermined.

Step 2 determines TGCP $(s, \mathbf{u})$ by evaluating the determinant of the matrix computed in step 1. Step 2-1 is performed by choosing random values for u_i and coefficients of f_i^*. From Corollary 3.2, we know a bound on the degree of TGCP $(s, \mathbf{u})$ at step 2-2, and can therefore compute some multiple of it using interpolation. More precisely, choose $\dim N - \mathrm{MV}_{-0}$ 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 TGCP $(s, \mathbf{u})$. Step 2-3 checks to make sure the generic choice made in step 2-1 was acceptable.

Step 3 determines the minimal polynomial, introducing the variable T . Again from Corollary 3.2, we know the degree of $\mathrm{Pert}_{A_0}(T, u_1, \ldots, u_n)$, and can compute it exactly using interpolation. More precisely, we choose $\mathrm{MV}_{-0} - 1$ 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 $\mathrm{Pert}_{A_0}(T, u_1, \ldots, u_n)$.

Step 4 determines the coefficient polynomials. Similar to steps 2-2 and 3, we use interpolation at step 4-1, to determine two univariate polynomials in $\mathbb{Q}[t]$.

At step 4-2, we compute the square-free part for each of the two polynomials from step 4-1. For example, we divide $p_i^+(t)$ by the gcd of $p_i^+(t)$ and its derivative (computed using Euclid's algorithm).

At step 4-3, we again introduce the variable T, but treat it as a constant when computing the gcd. To avoid symbolic computation with T,

we use the fact that g will be linear with probability 1 (dependant on the choices of u_i in step 2-1). In this case, g is the *first subresultant* for $q_i^-(t)$ and $q_i^+(2T - t)$ [29]. The coefficeints of the first subresultant for univariate polynomials are defined to be Sylvester resultants for some submatrices of Sylvester matrix.

Step 4-4, the coefficients of $g(t)$ from step 4-3, which are univariate polynomials in T, are used to compute the coordinate polynomials forming the RUR. Like step 4-2, step 4-4 involves only basic operations over the ring of polynomials with rational coefficients.

All of these stages can be implemented exactly by the use of multiprecision rational arithmetic. Thus, we obtain the RUR *exactly.* An example is shown in detail in section 5

In this section, we describe our method to support exact computation over the coordinates of the roots of a system, expressed in terms of the RUR. Note, in general, those numbers are not real. We begin by summarizing the method for finding root bounds for real algebraic numbers proposed in [20] and [18] in sections 3.2.1 and 3.2.2 We then extend this method to find bounds for complex algebraic numbers in section 3.2.3. Finally, we describe our method to determine the sign of numbers expressed in terms of the RUR exactly in section 3.2.4.

In this section, we assume all the polynomials are with integer coefficients for simplicity. Since we are interested in the roots of polynomials, 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 $\widetilde{\alpha}$ for α s.t.$|\widetilde{\alpha} - \alpha| < \frac{\rho}{2}$, namely, $\alpha = 0$ iff $|\widetilde{\alpha}| < \frac{\rho}{2}$.

We use the root bounds as introduced by Mignotte [30]: define the Mahler 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

$$M(e) = |e_n| \prod_{i=1}^{n} \max\{1, |\zeta_i|\}. \tag{9}$$

Define the *degree* $\deg \alpha$ and *Mahler measure* $M(\alpha)$ of an algebraic number α to

be the degree and Mahler 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 Mahler measure of an algebraic number are well-defined. Then

$$\frac{1}{M(\alpha)} \leq |\alpha| \leq M(\alpha). \tag{10}$$

Proposition 3.3 (Mignotte) [30]

Let α and β with $\alpha\beta \neq 0$ be algebraic numbers. Then

(1) $M(\alpha \pm \beta) \leq 2^{\deg\alpha \deg\beta} M(\alpha)^{\deg\beta} M(\beta)^{\deg\alpha}$.

(2) $M(\alpha\beta) \leq M(\alpha)^{\deg\beta} M(\beta)^{\deg\alpha}$.

(3) $M\left(\frac{1}{\alpha}\right) = M(\alpha)$.

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` [17] and `Core` [18]. These libraries support exact arithmetic and comparison operations over real algebraic numbers of the form $e(\xi_1, \ldots, \xi_m)$ where e is an expression involving $+, -, /$ and $\sqrt[k]{}$, and $\xi_1, \ldots, \xi_m$ are rational numbers. To determine whether or not $e = 0$, we numerically compute an approximation $\tilde{e}$ for e to enough precision s.t. the root bound allows us to make a decision.

In `LEDA` and `Core`, an expression e is represented as a Directed Acyclic Graph (DAG). The leaves of the DAG are rational numbers and the internal nodes are labeled by unary or binary operators. Every node f of e maintains a root bound and an approximation for a subexpression represented as a DAG rooted at f.

The root bounds implemented in `LEDA` and `Core` are upper and lower bounds on Mignotte's bound. Those bounds are "constructive", i.e., they are efficiently calculated without actually computing minimal polynomials for numbers, typically using some norms for minimal polynomials. They established the recursive rules, like Proposition 3.3, to bound the degree and Mahler measure of the minimal polynomial for a real algebraic number of the form $f(\xi_1, \ldots, \xi_m) \circ g(\xi_1, \ldots, \xi_m)$, where f and g are some expressions and $\circ$ is some operator, from those for $f(\xi_1, \ldots, \xi_m)$ and $g(\xi_1, \ldots, \xi_m)$.

They use *precision-driven computation* [15] [20] to compute an approximation $\widetilde{e}$ for e to prescribed precision p. Suppose e is the binary node $e_1 \circ e_2$. First, it calculates precisions p_1 and p_2 to which e_1 and e_2 will be approximated. Then, it computes approximations $\widetilde{e_1}$ and $\widetilde{e_2}$ for e_1 and e_2 to precision p_1 and p_2, respectively. Finally, compute $\widetilde{e_1} \circ \widetilde{e_2}$ to obtain $\widetilde{e}$.

3.2.3 Exact Computation for Complex Algebraic Numbers

We would like to determine the exact sign of the coordinates of points expressed in terms of coordinate polynomials (h_i's) evaluated at roots of the minimal polynomial (h). In general, roots of h are not real. Thus, we must establish the method for complex algebraic numbers.

The root bounds approach to exact sign determination for real algebraic numbers naturally adapts to complex algebraic numbers by dealing with their real and imaginary parts individually.

Let $e(\zeta_1, \ldots, \zeta_m)$ be a complex algebraic number where e is an expression involving $+, -, \cdot$ and $/$, and $\zeta_1, \ldots, \zeta_m$ be complex algebraic numbers. We

apply recursive rules to the real algebraic numbers

$$e_R\left(\Re\zeta_1,\ldots,\Re\zeta_m,\Im\zeta_1,\ldots,\Im\zeta_m\right) \text{ and } e_I\left(\Re\zeta_1,\ldots,\Re\zeta_m,\Im\zeta_1,\ldots,\Im\zeta_m\right)$$

where e_R and e_I are expressions satisfying

$$e\left(\zeta_1,\ldots,\zeta_m\right) =$$

$$e_R\left(\Re\zeta_1,\ldots,\Re\zeta_m,\Im\zeta_1,\ldots,\Im\zeta_m\right) + \sqrt{-1}e_I\left(\Re\zeta_1,\ldots,\Re\zeta_m,\Im\zeta_1,\ldots,\Im\zeta_m\right).$$

To complete the adaption, the base cases for recursion must be treated. Thus, our task can be 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 Mahler measures of $\Re\zeta$ and $\Im\zeta$ and 2) approximations to any prescribed precision.

For 2), we implement Aberth's method [31] to compute approximations for (real and imaginary) parts of roots of univariate polynomials. To have an approximation to any given precision, we use floating point numbers with multi-precision mantissa.

For 1), we first find univariate polynomials R_e and I_e with integer coefficients s.t. $R_e\left(\Re\zeta\right) = I_e\left(\Im\zeta\right) = 0$, and then, calculate bounds for degrees and measures of $\Re\zeta$ and $\Im\zeta$ from the degrees and coefficients of R_e and I_e.

Proposition 3.4 *Let ζ be an algebraic number specified as a root of a polynomial $e\left(T\right) \in \mathbb{Z}\left[T\right]$. Write* $\mathrm{SylRes}_U\left(f,g\right)$ *for the Sylvester resultant of uni-*

variate 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

$$\sum_{i=0}^{m} s_i T^i = \mathrm{SylRes}_U\left(e(T-U), e(U)\right),$$

and

(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

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

Proof If ζ is a root of e then its complex conjugate $\overline{\zeta}$ is also a root of e. Thus, the sum $\zeta + \overline{\zeta} = 2\Re\zeta$ of 2 roots of e is a root of $\mathrm{SylRes}_U\left(e(T-U), e(U)\right)$, and the difference $\zeta - \overline{\zeta} = 2\sqrt{-1}\Im\zeta$ of 2 roots of e is a root of $\mathrm{SylRes}_U\left(e(T+U), e(U)\right)$.

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 $\sum_{j=0}^{\lfloor \frac{m}{2} \rfloor} (-1)^j s_{2j} T^{2j}$. □

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 [30], 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 3.5 *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 RUR

We describe our method to determine the sign of numbers expressed in terms of the RUR exactly. This means we must be able to determine the sign of polynomials evaluated over complex algebraic numbers.

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

Algorithm: Exact_Sign

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

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

1. Construct bivariate rational functions r_R and r_I with rational coefficients s.t. $e(h_1(\theta), \ldots, h_n(\theta)) = r_R(\Re\theta, \Im\theta) + \sqrt{-1} r_I(\Re\theta, \Im\theta)$.

2. Recursively compute the root bounds for r_R and r_I. The base cases are

given in Proposition 3.5.

3. Use precision-driven computation to obtain approximations for $r_R(\Re\theta, \Im\theta)$ and $r_I(\Re\theta, \Im\theta)$ to certain precision s.t. the root bounds for them allow us to determine their signs. The base cases, i.e., computing approximations for $\Re\theta$ and $\Im\theta$ to certain precision is done by Aberth's method.

3.3 Exact Computation for RUR

In this section, we present our exact algorithm to compute the RUR for 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 [10] [27] that, if we replace $A_1, \ldots, A_n$ by $\{\mathbf{o}\} \cup A_1, \ldots, \{\mathbf{o}\} \cup A_n$ then we find the RUR for some finite *superset* Z'' of Z'. We can remove those extra points in $Z'' \setminus Z$, simply by testing, for each $x \in Z''$, 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: for every root ζ of h in the RUR for Z'', test whether or not $h_i(\zeta) = 0$ for $i = 1, \ldots, n$.

3.3.2 Non-square Systems

Consider a system of m polynomials $f_1, \ldots, f_m$ in n variables with rational coefficients. We would like to compute the RUR for some finite subset Z' of the set Z of common roots of the system. If $m \neq n$ then we construct some square system s.t. the set of 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 of polynomials $f_1, \ldots, f_m, f_{m+1}, \ldots, f_n$ by setting $f_m = f_{m+1} = \cdots = f_n$.

Otherwise, $m > n$. Let

$$g_i = c_{i1} f_1 + \cdots + c_{im} f_m, \quad i = 1, \ldots, n, \tag{11}$$

where $c_{11}, \ldots, c_{nm}$ are generic rational numbers. In implementations, these can be chosen randomly or from some predetermined set of numbers; later tests eliminate extraneous roots thus introduced. It can be shown that there exist $c_{11}, \ldots, c_{nm} \in \mathbb{Q}$ s.t. every irreducible component of the set $\overline{Z}$ of all the common roots of $g_1, \ldots, g_n$ is either an irreducible component of the set Z of all the common roots of $f_1 \ldots, f_m$ or a point [27]. We have already seen that we can compute the RUR for points in some finite subset $\overline{Z}'$ of $\overline{Z}$ s.t. $\overline{Z}'$ contains at least one point from every irreducible component of $\overline{Z}$. 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 $x \in \overline{Z}'$, 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: for every root ζ of h in the RUR, test whether or not $h_i(\zeta) = 0$ for $i = 1, \ldots, n$.

3.3.3 The Exact RUR

The algorithm is written as follows:

Algorithm: RUR

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

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

roots of the system.

1. Form a square system $g_1, \ldots, g_n$.

1-1. If $m < n$ then form a square system $g_1 = f_1, \ldots, g_m = f_m, g_{m+1} = f_m, \ldots, g_n = f_m$.

1-2. If $m = n$ then $g_1 = f_1, \ldots, g_n = f_n$.

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

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

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

2. For $i = 1, \ldots, n$ 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 **RUR_square** to compute $h, h_1, \ldots, h_n \in \mathbb{Q}[T]$ forming the RUR for some finite subset $\overline{Z}'$ of the set $\overline{Z}$ of affine roots of $g_1, \ldots, g_n$.

4. If steps 1-3 and/or 2-1 are executed then use **Exact_Sign** to test, whether or not $f_1(h_1(\theta), \ldots, h_n(\theta)) = \cdots = f_m(h_1(\theta), \ldots, h_n(\theta)) = 0$ 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 the application of the RUR to various geometric problems. A number of geometric problems involve solving systems of polynomials, and thus computation with algebraic numbers. We first describe the general procedure for using the RUR on such problems. We then focus on

how the RUR can be used in specific situations that arise during boundary evaluation in computer-aided design applications.

4.1 Exact Computation for Algebraic Numbers

We present here some algorithms to support exact geometric computation for algebraic numbers. We will also refer to these later when we describe specific problems related to boundary evaluation.

4.1.1 Positive Dimensional Components

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

Recall 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 $f_1^*, \ldots, f_n^*$ used to perturb the input system.

Suppose two distinct executions of Algorithm **RUR** find two distinct finite subsets Z_1' and Z_2' of Z and their RUR's:

$$Z_1' = \left\{ \left(h_1^{(1)}(\theta_1), \ldots, h_n^{(1)}(\theta_1) \right) \mid h^{(1)}(\theta_1) = 0 \right\}$$

and

$$Z_2' = \left\{ \left(h_1^{(2)}(\theta_2), \ldots, h_n^{(2)}(\theta_2) \right) \mid h^{(2)}(\theta_2) = 0 \right\}$$

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.

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

Hence, $Z_1' \setminus Z_2' \neq \emptyset$ implies that Z has some positive dimensional components.

We can test $Z_1' \setminus Z_2' \neq \phi$ since Algorithm **Exact_Sign** to tell whether or not $h_i^{(1)}(\theta_1) = h_i^{(2)}(\theta_2)$ exactly. Therefore, we have the following generic algorithm to tell, probabilistically, whether or not the set of the roots of a system has some positive dimensional components.

Algorithm: Positive Dimensional Components

Input: $f_1, \ldots, f_m \in \mathbb{Q}[X_1, \ldots, X_n]$, and `Max_Counts`, a .

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

1. `Has_Pos_Dim_Compo` := $Probably\_False$, `Count` := 1.

2. While `Has_Pos_Dim_Compo` = $Probably\_False$ and `Count` < `Max_Counts` do:

2-1. Use **RUR** to compute the exact RUR for some finite subsets Z_1' andZ_2' of Z. Increment `Count`.

2-2. Use **Exact_Sign** to compare points in Z_1' and Z_2' pairwise. If they are not the same then `Has_Pos_Dim_Compo` := $True$.

Assuming the f_i^* from Algorithm **RUR** are chosen generically, only one iteration of the loop is needed. In practice, since the f_i^* are chosen randomly, `Max_Counts` is usually set quite low, e.g. to 2 or 3.

4.1.2 Root Counting

We present an algorithm to count the number of common roots of a given system of polynomials with rational coefficients. The above algorithm detects whether or not the set of the roots has a positive-dimensional component. 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 minimal polynomial in the RUR.

Algorithm: Root Counting

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

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

1. Use **Positive Dimensional Components** to test whether or not the set Z of the roots of the system $(f_1, \ldots, f_m)$ 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 in the RUR counted in some appropriate method.

4.1.3 Computing/Counting Real Roots

We present an algorithm to compute the real common roots of a 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 imaginary parts of coordinate polynomials in the RUR.

Algorithm: Real Roots

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

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

1. Use **RUR** to compute the exact RUR for the root Z of the system:

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

2. For $i := 1, \ldots, n$ do:

2-1. Set up expressions h_{Ri} and h_{Ii} satisfying

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

(see section 3.2.3).

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

3 Return

$$\left\{ (h_{R1}(\Re\theta, \Im\theta), \ldots, h_{Rn}(\Re\theta, \Im\theta)) \mid \begin{array}{rcl} h(\theta) & = & h_{I1}(\Re\theta, \Im\theta) \\ & = & \cdots \\ & = & h_{In}(\Re\theta, \Im\theta) = 0 \end{array} \right\}.$$

That is, if the sign of the imaginary portion of every coordinate is 0, then return the real portion of the coordinates.

4.2 Application to Boundary Evaluation

We now describe how the RUR can be applied to a specific geometric computation—boundary evaluation.

4.2.1 Overview of 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 operations tend to 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 [1]. 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). 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. For real-world data, this general-position assumption is often violated. Overcoming this limitation has been a major motivator of the work presented here.

ESOLID finds and represents points in the 2D domain only, using the MAPC library [2]. The 2D vertex representation (coupled with techniques that produce bounding intervals in 3D) is sufficient for performing the entire boundary evaluaion 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 RUR

The 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:

- The RUR: the minimal polynomial along with the coordinate polynomials
- A bound on a unique (complex) root of the minimal polynomial
- A bounding interval determined by evaluating the root of the minimal polynomial within the coordinate polynomials.
- The two original polynomials used to compute the RUR.

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

Points would be isolated using Algorithm **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 determine the precision to which we must

determine the root of the minimal polynomial. In doing so, we can guarantee accurate sign evaluation at any point in the boundary evaluation process.

For example, a common operation in ESOLID involves finding the intersections of two curves within some region. With the RUR approach, the RUR (h, h_1, h_2) would be computed, and roots of the minimal polynomial found - each of these corresponding to a potential point. Then, to determine which of those roots lie in the region of interest, we would form algebraic expressions comparing with the region boundaries. For example, if the lower x bound is at point a, we would want roots such that $h_1 - a > 0$, so we would need a root bound for the expression $h_1 - a$. This root bound would be propagated backward to determine the root bound for h, and the roots of h would be determined to at least that precision. These values would then be propagated backward to determine the x-coordinate of each point (as a bounded range), and we would know for certain whether the coordinate was greater than, less than, or equal to a. Later, as this point is used in further computations, the approximation of the root of h (and thus the bounds on the coordinate values) might be refined to a greater and greater level - following the standard precision-driven approach.

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, which is the topic of the next section.

As stated previously, degeneracies are a major remaining obstacle in our exact boundary evaluation implementation. The 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 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 [32] 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 RUR approach. 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 RUR method is particularly well-suited for problems that arise in degenerate situations. Unlike methods created under genericity assumptions (e.g. [2]), the 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 [2] fail in such situations, while an 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 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 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 an RUR. We will limit our discussion to the 2D case, as this is what is necessary for boundary evaluation.

By assumption, an existing point, x, will be known as the (unique) intersection of two (or more) polynomials, f_1, f_2 in region R. We wish to find the RUR representation of x. We first use Algorithm **RUR** to compute the exact RUR for the set Z of the roots of the system (f_1, f_2):

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

Next, use Algorithm **Exact_Sign** to compare the boundaries of R and the coordinates of points in Z to find θ s.t.

$$x = (h_1(\theta), \ldots, h_n(\theta)), \quad h(\theta) = 0.$$

4.3.2 Coincident Points

Let x_1 and x_2 be 2 points. We would like to know whether or not x_1 and x_2 are the same. If these points are described using the RUR, the comparison is straightforward: Algorithm **Exact_Sign** can be used to compare the coordinates of the two points.

4.3.3 Points Lying on Curves/Surfaces

Such cases will arise when four or more surfaces meet at a point. Let x be a point and f be a polynomial representing some surface or some curve. We would like to know whether or not $f(x) = 0$.

The RUR expresses the coordinates of x in terms of polynomials evaluated some root of some other polynomial.

$$x = (h_1(\theta_j), \ldots, h_n(\theta_j)), \quad h(\theta_j) = 0.$$

Now, construct an expression for $f(x) = f(h(\theta_j), \ldots, h_n(\theta_j))$ and use Algorithm **Exact_Sign** to determine its sign. If the sign is 0, the point lies on the curve.

4.3.4 Overlapping Curves, Curve Lying on A Surface

Let f_1 and f_2 be polynomials representing two curves. We would like to know whether or not f_1 and f_2 overlap. This can be done straightforwardly; use Algorithm **Positive Dimensional Components** to see whether or not the set of the of the system (f_1, f_2) has some positive dimensional components. A similar approach can be used to determine whether three surfaces, expressed in implicit form, overlap (thus detecting the curve lying on surface case).

4.3.5 Surfaces Tangent at a Point

When surfaces are tangent at a point, the intersection curve has a singularity. As stated above (Section 4.3.1, Algorithm **Real Roots** can handle these cases smoothly. In the event one wishes to detect such cases and treat them separately, the topological characteristics of the intersection curves can be readily detected when examining intersection curve topology (as a stage in boundary evaluation).

4.3.6 Tangentially Intersecting Curves

This can arise when a curve intersects a surface tangentially, appearing as two 2D curves intersecting tangentially. Like the previous example, Algorithm **Real Roots** encounters no unusual computational difficulty in such a situation. If one wants to detect such cases, a relatively simple test can be used to check the relative curve topology.

5 Experimental Results

We have implemented the RUR method in GNU C++. We use the Gnu Multiple Precision (GMP) arithmetic library to support multi-precision arithmetic. Every subroutine including all linear programming and basic linear algebra operations have been written using GMP. All the experiments shown in this section are performed on a 1.9 GHz Intel Pentium CPU with 512 MB memory operated by Linux Kernel 2.4.20.

We present here a detailed example and timing results for a few cases in order to indicate the relative time requirements of various parts of the 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 RUR procedure.

5.1 Solving Systems in RUR

In this section, we show timing breakdowns for the application of the 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

We summarize here the examples used to generate the timing breakdowns given in Table 5.1.

F_1: Positive Dimensional Components

Consider a system F_1 of 2 polynomials

$$f_1 = 1 + 2X - 2X^2Y - 5XY + X^2 + 3X^2Y,$$
$$f_2 = 2 + 6X - 6X^2Y - 11XY + 4X^2 + 5X^3Y.$$

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

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

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

Input System	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
Num of Poly	2	2	2	2	2	2	2	2
Num of Var	2	2	2	3	2	2	2	2
Max Deg of Poly	5	3	3	2	2	2	2	2
Max Bit of Coeff	4	2	2	2	309	402	54	97
Num of Roots of System	∞	2	2	1	2	2	2	4
Num of Roots of Min Poly	4	2	2	1	2	4	2	2
Num of Real Roots of System	2	2	2	1	2	2	0	4
Max Bit of Coeff of Coord Poly	111	5	6	1	364	311	103	451
Total Time in Seconds	6.50	0.233	0.454	459.0	0.317	0.621	0.648	1.772
% for Sparse Res Matrix	1.2	20.8	11.5	0.1	13.4	10.0	13.0	3.8
% for Min Poly	28.2	33.1	56.0	15.3	13.2	14.8	31.5	4.1
% for $q_i^{\pm}$'s	67.9	41.6	30.3	84.6	52.4	57.9	51.0	15.7
% for Coord Poly	2.5	2.4	1.1	0	19.1	15.8	3.3	75.2
% for Approx Roots of Min Poly	0.2	1.6	0.8	0	1.4	1.1	0.8	1.0
% for Eval Coord Poly	0.0	0.5	0.3	0	0.5	0.4	0.4	0.2

Table 1

Timing breakdown for several examples.

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

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

at step 2-1, then we obtain the RUR for some finite subset Z_1' of the roots of F_1 as

$$\begin{aligned}
h^{(1)}(T) &= -284032T^4 - 4928704T^3 + 26141910T^2 + 440186393T - 1411684417, \\
h_1^{(1)}(T) &= \frac{7689907930880}{5870227699098039}T^3 + \frac{1037528584832}{124898461682937}T^2 \\
&\quad - \frac{1313640082212220}{5870227699098039}T - \frac{851151731612679}{1956742566366013} \\
h_2^{(1)}(T) &= -\frac{23069723792640}{33264623628222221}T^3 - \frac{3112585754496}{707757949536643}T^2 \\
&\quad + \frac{27435113904634}{33264623628222221}T + \frac{7660365584514111}{33264623628222221}.
\end{aligned}$$

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

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

at step 2-1, then we obtain the 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_{Ri} and r_{Ii} s.t.

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

The root bounds for r_{R1}, r_{I1}, r_{R2} and r_{I2} 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 $i = 1, 2$:

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

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

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

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

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

and

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

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). Similary, points

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

and

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

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 polynomials

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

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

Consider the system F_3 of polynomials

$$f_{31} = X^3 - 3XY + Y^3,$$
$$f_{32} = X - Y.$$

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

Since both f_{31} and f_{32} do not have a constant term, Algorithm **RUR** first finds the trivial root, the origin, adds the origin to the supports of f_{31} and f_{32} at step 2-1 and continues. At step 3, the algorithm finds 2 roots, but, at step 4, one of them will be removed.

Because the intersections of both systems at singular points, we cannot apply methods that depend on smoothness of the curve (such as the MAPC/ESOLID method).

F_4: Spheres Tangent with Each Other

For this case, we find intersections of 2 spheres that meet tangentially.

The minimal polynomial is linear, and thus, we can obtain coordinate polynomials immediately.

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 [33] 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.

F_7: No Real Roots

We would like to find intersections of 2 ellipses. Rather than real intersections, these ellipses have 2 complex intersections. We use Algorithm **Real_Roots** to distinguish real roots from the other roots.

In this case, $\mathrm{Pert}_{A_0}(\mathbf{u})$ is a constant term of $\mathrm{TGCP}(s, \mathbf{u})$, and thus, we actually do not have to interpolate at step 2-2. We still must interpolate at steps 3 and 4-1, which are the most time consuming parts.

F_8: Burst of Coefficients of RUR

We would like to solve a system of 2 ellipses with supports

$$(2,0), (1,0), (0,2), (0,1), (0,0)$$

and

$$(2,0), (1,1), (1,0), (0,2), (0,1), (0,0),$$

respectively. The system 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 approximate the roots of the minimal polynomial.

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, 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 sparse 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 RUR, mainly because of their huge coefficients. The size of the coefficients of univariate polynomials forming the RUR depends on the size of the input as well as the size of the values randomly chosen during the execution. Our current implementation makes no attempt to control this coefficient growth.
- Constructing the sparse resultant matrix, and finding and evaluating over the roots of the minimal polynomial takes up an insignificant portion of the time. Conceivably, finding and evaluating over the roots of the minimal polynomial could take a higher percentage of time if the root is involved in a computation where a very tight root bound is required.

In this section, we show the timings for comparison of the RUR method with that used in the MAPC library within the ESOLID system. Note that the RUR approach 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.

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.

Input System	F_5	F_6	F_7	F_8
Number of Polynomials	2	2	2	2
Number of Variables	2	2	2	2
Max. Total Degree of Polynomials	2	2	2	2
Max. Bit Length of Coefficients	309	402	54	97
Number of Real Roots of System	2	2	0	4
Number of Roots of Min. Poly.	2	2	2	4
Number of Real Roots ESOLID finds	2	1	0	2
Total Time by RUR in Seconds	0.317	0.621	0.648	1.772
Total Time by ESOLID/MAPC in Seconds	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

In addition, the exact RUR loses efficiency by finding all the complex roots while ESOLID / MAPC find 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 which a method such as those MAPC/ESOLID can handle, the RUR has an unacceptably high performance rate. 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, i.e. computing every coefficient 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. We have shown that the RUR works even in exceptional cases, such as where the system has roots with zero coordinates, and for non-square systems.

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 [1]. 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 2 immediately obvious issues to tackle:

- Use some algorithm which produces sparse resultant matrices of smaller size [13] [14]. Although the matrix constuction 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.

Finally, we plan to fully integrate the RUR into a solid modeling system (such as ESOLID) that supports 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 exactly how to put together such a hybrid system is a challenge for future work.

References

[1] J. Keyser, T. Culver, M. Foskey, S. Krishnan, D. Manocha, ESOLID: A system for exact boundary evaluation, in: Proc. of 7th ACM Solid Modeling, ACM, 2002, pp. 23 – 34.

[2] J. Keyser, T. Culver, D. Manocha, S. Krishnan, Efficient and exact manipulation of algebraic points and curves, Computer-Aided Design 32 (11) (2000) 649 – 662.

[3] H. J. Stetter, Matrix eigenproblems are at the heart of polynomial system solving, ACM SIGSAM Bulletin 30 (1996) 22 – 25.

[4] R. M. Corless, Editor's corner: Gröbner bases and matrix eigenproblems, ACM SIGSAM Bulletin 30 (1996) 26 – 32.

[5] F. Rouillier, Solving zero-dimensional systems through the rational univariate representation, Applicable Algebra in Engineering, Communication and Computing 9 (5) (1999) 433 – 461.

[6] L. Gonzalez-Vega, F. Rouillier, M. F. Roy, Symbolic recipes for polynomial system solving, in: A. M. Cohen, H. Cuypers, H. Sterk (Eds.), Some tapas of computer algebra, Algorithms and computation in mathematics, V. 4, Springer-Verlag, 1999.

[7] M. Giusti, G. Lecerf, B. Salvy, A Gröbner free alternative for polynomial system solving, Journal of Complexity 17 (1) (2001) 154 – 211.

[8] G. Lecerf, Quadratic newton iteration for systems with multiplicity, Journal of FoCM 2 (3) (2002) 247 – 293.

[9] J. Canny, Generalized characteristic polynomials, in: Proceedings of ISSAC '88, LNCS 358, Springer-Verlag, 1988, pp. 293 – 299.

[10] J. M. Rojas, Solving degenerate sparse polynomial systems faster, Journal of Symbolic Computation 28 (1/2) (1999) 155 – 186.

[11] J. F. Canny, I. Emiris, An efficient algorithm for the sparse mixed resultant, in: Proceedings of AAECC 10 '93, LNCS 673, Springer-Verlag, 1993, pp. 89 – 104.

[12] J. F. Canny, I. Z. Emiris, A subdivision-based algorithm for the sparse resultant, Journal of ACM 47 (3) (2000) 417 – 451.

[13] I. Z. Emiris, J. F. Canny, Efficient incremental algorithm for the sparse resultant and the mixed volume, Journal of Symbolic Computation 20 (2) (1995) 117 –

149.

[14] C. D'Andrea, I. Z. Emiris, Hybrid sparse resultant matrices for bivariate polynomials, Journal of Symbolic Computation 33 (5) (2002) 587 – 608.

[15] T. Dubé, C. Yap, The exact computation paradigm, in: D. Du, F. Hwang (Eds.), Computing in Euclidean Geometry, 2nd Edition, Lecture Notes on Computing, World Scientific, 1995, pp. 452 – 492.

[16] C. Yap, Towards exact geometric computation, Computational Geometry 7 (1) (1997) 3 – 23.

[17] S. Mehlhorn, M. Näher, LEDA - A Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1999.

[18] V. Karamcheti, C. Li, C. Yap, A Core library for robust numerical and geometric computation, in: Proc. of 15th Annual Symposium on Computational Geometry, ACM, 1999, pp. 351 – 359.

[19] C. Burnikel, R. Fleischer, K. Mehlhorn, S. Schirra, A strong and easily computable separation bound for arithmetic expressions involving square roots, in: Proc. of 8th ACM-SIAM Symposium on Discrete Algorithms, ACM, 1997, pp. 702 – 709.

[20] C. Burnikel, R. Fleischer, K. Mehlhorn, S. Schirra, Exact efficient computational geometry made easy, in: Proc. of 15th Annual Symposium on Computational Geometry, ACM, 1999, pp. 341 – 350.

[21] C. Burnikel, R. Fleischer, K. Mehlhorn, S. Schirra, A strong and easily computable separation bound for arithmetic expressions involving radicals, Algorithmica 27 (1) (2000) 87 – 99.

[22] C. Li, C. Yap, A new constructive root bounds for algebraic expressions, in: Proceedings of 12th ACM-SIAM Symposium on Discrete Algorithms '01, ACM, 2001, pp. 476 – 505.

[23] C. Hoffmann, J. Hopcroft, M. Karasick, Robust set operations on polyhedral solids, IEEE Computer Graphics and Applications 9 (6) (1989) 50–59.

[24] S. Fortune, Polyhedral modelling with multiprecision integer arithmetic, Computer Aided Design 29 (2) (1997) 123–133.

[25] C. Hoffmann, Geometric and Solid Modeling, Morgan Kaurmann Publishers, Inc., San Mateo, California, 1989.

[26] P. Koiran, Hilbert's nullstellensatz is in polynomial hierarchy, Journal of Complexity 12 (4) (1996) 273 – 286.

[27] J. M. Rojas, Algebraic geometry over four rings and the frontier to tractability, Contemporary Mathematics 270 (2000) 275 – 321.

[28] P. Pedersen, B. Sturmfels, Product formulas for resultants and chow forms, Mathematische Zeitschrift 214 (3) (1993) 377 – 396.

[29] L. Gonzáez-Vega, A subresultant theory for multivariate polynomials, in: Proceedings of ISSAC '91, ACM, 1991, pp. 79 – 85.

[30] N. Mignotte, D. Stefanescu, Polynomials: An Algebraic Approach, Springer-Verlag, 1999.

[31] O. Aberth, Iteration methods for finding all zeros of a polynomial simultaneously, Mathematics of Computation 27 (122) (1973) 339 – 344.

[32] J. C. Keyser, Exact boundary evaluation for curved solids, Ph.D. thesis, Department of Computer Science, University of North Carolina, Chapel Hill (2000).

[33] P. C. Dykstra, M. J. Muuss, The BRL-CAD package an overview, Tech. rep., Advanced Computer Systesms Team, Ballistics Research Laboratory, Aberdeen Proving Ground, MD, http://ftp.arl.mil/brlcad/ (1989).