NURBS Interpolation of Surface-to-Surface Intersections

Trân Quôc-Nam* and Carl Van Geem†
Research Institute for Symbolic Computation (RISC–Linz),
Johannes Kepler University
A–4040 Linz, Austria
E-mail: {tqnam, cvangeem}@risc.uni-linz.ac.at

October 2, 1997

Abstract

We present a hybrid symbolic-numerical algorithm for piecewise interpolation and representation of surface-to-surface intersections – a fundamental and difficult problem in Computer Aided Geometric Design (CAGD) and solid modeling. Our method helps to get a fast and accurate insight of the intersections – This is the main contribution of the paper. The method overcomes the main difficulties of numerical tracing, namely finding a sample point for every real branch and marching through singularities by using techniques for real root isolation of polynomials with real algebraic number coefficients. On the other hand, the method does not require the expensive computation of birational mapping, projections, re-constructions, etc. as in a pure symbolic method. Further it utilizes non-uniform rational B-spline (NURBS) to piecewise interpolate the intersection in order to improve the efficiency of the algorithms and the smoothness of the plots. Experiments from surface-to-surface intersection examples found in the literature show a dramatic speed up of the method.

1 Introduction

In this paper, we describe a hybrid symbolic-numerical algorithm for piecewise interpolation and representation of surface-to-surface intersections – a fundamental and difficult problem in Computer Aided Geometric Design (CAGD) and solid modeling. Given two trivariate polynomials with rational number coefficients, the algorithm produces a space graph which is a non-uniform rational B-spline (NURBS) piecewise interpolation and homeomorphic to the real algebraic set of the intersection (in fact, a space curve) in Euclidean space. Due to its importance, there have been persistent efforts in devising algorithms for this problem (see [BFJP87], [BHLH88], [ACM88], [Hof89], [KPP90], [W$^+$91], [KP91], [Pat92, Pat93], [Tau94] and [Tra95a]). The main issue in the intersection problem is the efficient discovery and description of all features of the solution with high precision commensurate with the tasks required by the underlying geometric modeler. Reliability and efficiency of intersection algorithms are two basic prerequisites for their effective use in any geometric modeling system. It is closely associated with the way the algorithm handles such features as near singular cases, small loops, etc.

*Supported by the Austrian Science Foundation (FWF) project HySaX, Proj. No. P11160-TEC
†Partially supported by the CGAL Project, ESPRIT IV, No. 21957

In contrast to symbolic methods (see [Win93, Wal93]), our method avoids the use of expensive computations such as algebraic set decomposition, computation of birational mappings, projections, reconstructions, etc. while still ensuring the reliability of the method especially near the singularities. The method requires only rational arithmetic such as resultants, greatest common divisors, etc.

On the other hand, our algorithm overcomes the main difficulties of numerical tracing, namely finding a sample point for every real branch and tracing through singularities by using techniques for real root isolation of polynomials with real algebraic number coefficients.

A numerical tracing scheme technically requires formulating and solving a linear system of equations. Such a system is formulated both for determining an approximation to the curve at a given point, as well as when refining an estimate of the location of a point on the curve with Newton iterations. At a curve singularity, the linear system is singular, and nearby the condition number of the system is so large that roundoff errors destroy all accuracy of the estimate. To overcome this difficulty, we use a new technique of the first author's, namely the "Extended Newton Method" for determining the roots of an arbitrary system of equations iteratively, where the equations can be nonlinear algebraic or transcendental, the number of equations of the system can be greater than, less than or equal to the number of variables (see [Tra94, Tra97]).

Furthermore, in order to improve the efficiency of the algorithms and the smoothness of the plots, we utilize NURBS to piecewise interpolate the intersection. We use NURBS because it provides a concise formulation for curves that are composed of many polynomial pieces including conic sections and rational Bézier curves, thereby automatically controlling the overall curve smoothness. In addition, the algorithms are fast, numerically stable and invariant under common affine and projective transformations.

The symbolic tools are implemented in the software system CASA, while the NURBS tools are implemented in the software system CGAL and will become a CGAL application module. The software system CASA has been developed at RISC-Linz over the last seven years. Meanwhile, RISC-Linz is one of the seven academic developing sites of the CGAL library. Our method has been implemented as an interface between those two systems. It calls symbolic or numerical procedures when needed.

Experiments from examples found in the literature show a considerable speed up of our method in comparison with symbolic methods. Meanwhile, the method still works well near the singularities which is much better in comparison with numerical methods.

2 Preliminaries

In this section we give a brief introduction about the software systems CASA and CGAL, as well as about some tools which will be used in our method.

2.1 The Software System CASA

Constructive methods in algebra and algebraic geometry are gaining more and more importance with the availability of computers and computer algebra softwares. Since its first version (see [GKWW91]), CASA has been designed to perform computations and utilize reasoning about algebraic and geometrical objects in the classical affine and projective spaces over algebraically closed fields of characteristic zero. More precisely, the field has to be a computable field in the sense of the underlying computer algebra system Maple, i.e. all the arithmetic operations have to be available in the system. Usually, the field of computation is the rational numbers $\mathbb{Q}$ or a finite algebraic extension thereof.

The system has been developed since 1990 (see [GKWW91], [Win93], [MW96], [Wal93], [Tra95a, Tra95b, Tra96, Tra97]). With its newest version – version 2.3 – in May 1997 (see [TW97]), CASA is the ongoing product of the Computer Algebra group at the Research Institute for Symbolic Computation (RISC-Linz), the University of Linz, Austria.

In the system, an algebraic set – a central notion in algebraic geometry – can be represented in four different ways: (a) *Implicit representation:* An algebraic set is the set of common zeros of a system of polynomial equations. To give an algebraic set in implicit form means to give finitely many polynomials. (b) *Projected representation:* As a consequence of the primitive element theorem every irreducible d–dimensional algebraic set in n–dimensional space is, after a suitable linear transformation of coordinates, birationally projectable onto an irreducible d–dimensional algebraic set in $(d+1)$–dimensional space, which can be specified by a single polynomial in $d+1$ variables. This can be generalized to unmixed-dimensional algebraic sets. An algebraic set in projected form is given by a polynomial and a tuple of rational functions (specifying the birational mapping). (c) *Parametric representation:* Some irreducible algebraic sets can be parametrized by rational functions. An algebraic set in parametric form is given by a tuple of rational functions that parametrizes the algebraic set. (d) *Representation by places or Puiseux expansion:* All algebraic curves can be locally parametrized by a set of power series that are convergent around a point of the curve. An algebraic set is given by places if for each branch passing through a certain point on the algebraic set a tuple of power series that parametrizes the algebraic set around the point is specified.

The system provides a variety of operations on algebraic sets. As the efficiency of these operations is tightly bound to the way algebraic sets are represented, conversion routines have been provided to support various views on one object, to deepen the understanding of its principles, and to speed up algorithms working on algebraic sets.

CASA also works with the polynomial ideals corresponding to these geometric objects. The basic operations available in CASA include: ideal theoretic operations $+, *, \cap, /$; intersection, union, and difference of algebraic sets; computing tangent cones and tangent spaces; computation of the dimension of an algebraic set; decomposition into irreducible components; computation of the singularities, genus, neighborhood graphs and adjoint curves of an algebraic curve and so on. Besides these basic operations, the following more advanced operations are available: rational parametrization of rational curves over an optimal algebraic extension field, implicitization of parametrically given algebraic sets, Puiseux series expansions, multivariate resultants and Dixon's resultants, Gröbner bases of ideals and modules, Gröbner walk, computation of syzygy-bases and many more. The major goal of CASA was to design a system which provides a comfortable, easy to use, efficient, flexible and mathematically exact working environment for computational algebra and constructive algebraic geometry where all basic theoretical concepts map easily to available data structures. CASA is built on the kernel of Maple and is fully independent of the operating system; hence, it can be used on every hardware where Maple is running.

There are software systems which partially cover some of the fields of CASA, however, to the authors' knowledge, currently no other system provides all the functionalities of CASA.

2.2 CGAL Library

Only a few of the many algorithms developed over the past two decades in computational geometry found their way into practice. Reasons for this are the dissimilarity between fast floating-point arithmetic normally used in practice and exact arithmetic over the real numbers assumed in theoretical papers in the field of computational geometry, the lack of explicit handling of degenerate cases in these theoretical papers, and the inherent complexity of many (theoretically) efficient solutions. For these reasons there is a definite need for correct and efficient implementations of geometric algorithms. It is the ambitious goal of the CGAL-project to provide this, i.e. to make the large body of geometric algorithms, developed within the field of computational geometry, available for industrial application. A large part of the CGAL-project is devoted to the development of a **C**omputational **G**eometry **A**lgorithms **L**ibrary written in C++ ([FGK+96], [Ove96], [Sch96]).

The CGAL library consists of three layers: the Kernel provides the geometric primitives underlying geometric computations, the Basic Library provides a collection of implementations of a large number of standard geometric algorithms, such as convex hull and (Delaunay) triangulation algorithms, and Application Modules in which collections of geometric routines are provided that are useful for some specific application domain, such as Geographic Information Systems, Computer Graphics, Robotics, and Computer Vision. The tool for NURBS which we use here is under development at RISC-Linz and will become an application module for CGAL as well. Besides these three layers interfaces to other packages are provided, for example for visualization packages and for I/O tools. This includes simple things like writing a point to standard output, to a more advanced interface for 2D visualization with the windows in LEDA (developed at MPI, Saarbrücken), or for the 3D visualization tool geomview (developed at the Geometry Center, Mineapolis).

The primary design goals for the CGAL Kernel and Basic Library are robustness, efficiency, generality, ease of use, and openness. Degenerate cases are treated as well as inaccuracies during various numerical computations. Efficiency is a necessity for applications, and thus the CGAL library intents to provide performance-optimized robust specializations of geometric primitives for fast floating-point number types. The user can add ones own geometric predicates to CGAL. This is due to the fact that the whole library is templated with number types.

2.3 CGAL_NURBS

The Application Module for NURBS is being developed at RISC-Linz. It currently provides some basic operations for 2- and 3-dimensional NURBS curves and for 3-dimensional NURBS tensor product surfaces.

We put most emphasis on efficiency and short computation time, more than on precision. The algorithms are provided only for number types like double, although the overall design is with template classes and functions where the user can define ones own number type as well as using the standard one provided as part of the CGAL application module.

We chose to follow the design for NURBS in an object oriented fashion as it is also described in [PT97]. For curves in the CGAL application module we have implemented a class for the control polygon, a class for the knot vector, and a class for a 2- and 3-dimensional NURBS curve. We provide as basic algorithms the computation of a point on the curve for a given parameter value, the change of a weight, refinement of the knot vector, and removal of a knot. We chose to write only one class both for 2- and 3-dimensional curves since the operations we intended to provide on them are identical up to the dimension. The class is a template class, the template parameter is a class (the NURBS traits class, as it is called commonly in

CGAL) containing the type of point which is to be used in the NURBS curve class. The type of point implicitly defines the dimension of the curve. Since all operations for 2-dimensional points (CGAL_Point_2) in the Kernel and Basic Library of CGAL are also provided for 3-dimensional points (CGAL_Point_3) in the Kernel and Basic Library of CGAL, we could implement the NURBS curve without caring about the dimension of the points. Also the number type over which the computation is done is defined in the NURBS traits class.

For surfaces the application module provides in addition a class for the control polyhedron and a class for the NURBS surface. The basic algorithms present allow the computation of a point on the surface for a given parameter value and the computation of the derivative.

The control points are stored in homogeneous representation where the homogenizing coordinate represents the weight. We did so because in all computations we use the homogeneous representation.

It is our intention to extend functionality in future. In particular we plan to extend the number of basic algorithms, to compute the intersection of two NURBS curves, and to compute off-sets. Obviously the latter two are far from trivial and object of ongoing research in the field of Geometric Modeling and Computer Aided Geometric Design.

2.4 Interaction CGAL-CASA

The CASA system does a system call to a CGAL routine, with the degree for the 3-dimensional NURBS curve, the control points and their weights, and the knot vector, as well as the number of points to be computed on the curve as parameter values. The CGAL routine returns the points computed on the curve to the standard output. Then CASA can use this information for further visualization within CASA or for further computation.

The visualization in CASA does not allow interaction between the system and a user. Therefore additionally for 2-dimensional NURBS curves we implemented a visualization tool in CGAL (using also the facilities of LEDA windows). This allows the user to make changes to the NURBS curve – currently by changing a weight, refining the knot vector or by removing a knot – before returning to CASA. For 3-dimensional curves and surfaces an interface to geomview will be made in a near future. We did implement two simple programs which computes a given number of points on 3-dimensional curves and surfaces respectively and returns these points to the standard output. This allows CASA to read in those points and visualize the 3-dimensional curves or surfaces with the Maple tools.

2.5 Extended Newton's Method

In this section, we give some basic definitions and theorems of a method of the first author's for iteratively finding the roots of an arbitrary system of nonlinear algebraic equations. The method is complete for systems of nonlinear algebraic equations by using symbolic techniques to normalize the system when it is not underdetermined or does not have a full-rank Jacobian matrix. Details of this method can be found in [Tra97].

Let $x = [x_1,x_2,\ldots,x_k]^T$ be a k-dimensional column vector with components $x_1,x_2,\ldots,x_k$ in $\mathbb{C}$ and $f(x) = [f_1(x), \ldots,f_h(x)]^T$ an h-dimensional vector valued function, i.e. a column vector with components $f_1(x),\ldots,f_h(x)$ in $\mathbb{C}$. To deal with the norm of a vector, we can choose any one of the norms, for example: $\|x\|_\infty \equiv \max_{1\leq i\leq k} |x_i|$, $\|x\|_1 \equiv \sum_{i=1}^{k} |x_i|$ or $\|x\|_p \equiv \sqrt[p]{\sum_{i=1}^{k} |x_i|^p}$ where $p > 1$. We define the Jacobian matrix of f with respect to x to be the $h \times k$ matrix:

$$J_f(x) \equiv J(x) \equiv \left(\frac{\partial f_i(x)}{\partial x_j}\right), i = 1 \dots h, j = 1 \dots k. \tag{1}$$

When $h = k$ and with the assumption that $\det |J(x)| \neq 0$ for x in a closed and bounded set $\mathcal{S}$, we will denote the inverse Jacobian matrix of f at x by $J_f^{-1}(x)$. In this case, Newton's method is a well-known iterative method for finding the roots of the equation $f(x) = 0$. The iteration can be written in the form $x_{n+1} = g(x_n)$, where

$$g(x) = x - J_f^{-1}(x) \cdot f(x). \tag{2}$$

Unfortunately, when $h \neq k$, the Jacobian matrix of f is not a square matrix and therefore cannot be inverted. Hence, Newton's method fails to apply to such cases. We extend Newton's method such that the extended method can be applied to these cases as well.

To get around the problem of a nonsquare matrix, we will use what we call the half-inverse of the Jacobian matrix.

Definition 2.1 *A pseudo inverse of a matrix A is a matrix $A^\perp$ satisfying:*

$$\begin{array}{rclrcl} AA^\perp A & = & A, & (A^\perp A)^* & = & A^\perp A, \\ A^\perp A A^\perp & = & A^\perp, & (AA^\perp)^* & = & AA^\perp. \end{array}$$

where $(\cdot)^*$ denotes the conjugate transpose of the matrix.

There are several techniques for obtaining the pseudo inverse, which is unique, of a numerical matrix; for example, Hestenes' technique or Greville's technique (see [BO71]). However, those techniques are pure numerical techniques, i.e. they can only be applied to each individual numerical matrix, and we have to take into account the problems of rounding error, word length, precision of arithmetic, and so on. Moreover, when we use such techniques in our iterative method for finding the roots of a system of equations, we have to apply the approximate process for obtaining the pseudo inverse matrix $J^\perp$ at each iterate $x^{(n)}$. That means, it is time consuming and has accumulated errors.

To avoid such problems, we try to find a structure formula and use symbolic computation in an effort to speed up our method. We next define half-inverse matrices and show up two structure formulas where the first one is the well-known Penrose formula.

Definition 2.2 *The pseudo inverse of a matrix J is called a half-inverse of J, denoted by $J^\pm$, if $JJ^\pm = I$ or $J^\pm J = I$.*

Since $J^\pm$ is a matrix of order $k \times h$, we can simulate Newton's method with the iterative function g being defined as:

$$g(x) = x - J^\pm \cdot f(x) \tag{3}$$

where $g(x)$ and x are vectors of the same dimension k.

Lemma 2.1 *Given an underdetermined system of equations ($h < k$). For all x, if $\operatorname{rank}(J(x)) = h$ then $J^+(x) = J^* \cdot (J \cdot J^*)^{-1}(x)$ is the half-inverse of matrix $J(x)$. In other words, the method is well defined. Moreover, if α is a fixed point of g then $f(\alpha) = 0$.*

PROOF. See [Tra97]. □

Lemma 2.2 *Given an overdetermined system of equations (i.e., $h > k$). For all x, if $\operatorname{rank}(J(x)) = k$ then $J^-(x) = (J^* \cdot J)^{-1} \cdot J^*(x)$ is the half-inverse of matrix $J(x)$.*

PROOF. Similar to the proof of the previous lemma with J and J^* to be exchanged. □

2.5.1 Regularization

When the system of equations is not underdetermined or does not have a full-rank Jacobian matrix then the method may not be well defined or we may not get a root of the system. In this section, we will use a well-known symbolic method, namely Ritt's characteristic set method, to transform an irregular system of equations into a regular one when the equations are algebraic. We follow the presentation of [Tra97], where any missing details can be found.

Definition 2.3 *A system of equations is called a regular system if it is underdetermined and its Jacobian matrix has full rank everywhere.*

Algorithm 2.1 (Regularization) .

In: *A system H of polynomials in $\mathbb{C}[\tilde{x}]$*

Out: *A regularized system TR of H.*

1. $T := H$;
   $TR[i] :=$ *the smallest polynomial in class i, $i = 1 \ldots k$. (or 0 if H has no polynomial of class i).*

2. **repeat**
   $P := \emptyset$;
   for all $g \in T$
   $r = \text{prem}(g, TR)$;
   if $r \neq 0$ *then*
   if $class(r) = 0$ *then* **Return(1)**.
   else if $r \prec TR[\text{class}(r)]$ *then* $TR[\text{class}(r)] = r$.
   $P := P \cup \{r\}$;
   $T := T \cup P$;
   until $P = \emptyset$.

3. **Return**($TR \setminus \{0\}$)

The following theorem serves as a bridge between regularization and our method for finding the roots of an arbitrary system of equations.

Theorem 2.1 *Given an irregular system H of algebraic equations. Let $TR = f_1, \ldots, f_h$ be a triangular form of H obtained, for example, by Algorithm 2.1. Then TR is a regular system of algebraic equations. Moreover, there are infinitely many points in the affine space $A^k(\mathbb{C})$ such that the Jacobian matrix of TR evaluated at those points have full rank.*

Note that we have a constructive way to find points in the affine space $A^k(\mathbb{C})$ such that the Jacobian matrix of TR evaluated at those points has full rank by choosing points which are not a root of the leading coefficients of TR (we even have a bound for the coordinates of the points).

2.5.2 Extended Algorithm

We find a root of the function f w.r.t. the initial point $x^{(0)}$ by the following algorithm:

Algorithm 2.2 (Main)

Inputs: *A function $f : \mathbb{C}^k \to \mathbb{C}^h$, an initial point $x^{(0)}$, a tolerance ϵ and a maximum number of iterates M.*

Output: A complex fixed point of the function g.

1. $X := x^{(0)}$; $L := 0$;
2. **If** $h = k$ *and* J *is invertible* **then**
   $g(x) = x - J^{-1} \cdot f(x)$
3. **else**
   if f *is a regular system* **then**
   $g(x) = x - J^{\pm} \cdot f(x)$
   else
   regularize *the system.*
   if *"1" is in the regularized system then*
   Return*("the system has an empty zero set")*
4. $Y := g(X)$;
5. **While** $\|Y - X\| \geq \epsilon$ **do**
   (a) *If* $L > M$ *then* **Return***("failure").*
   (b) $X := Y$;
   (c) $Y := g(X)$;
   (d) $L := L + 1$;
6. **Return***(Y).*

When the sequence $\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}, \ldots\}$ converges to the fixed point of g, say α in a neighborhood of $x^{(0)}$, we will have $f(\alpha) = 0$. Note that, in general, if the system is irregular then we may not have $f(\alpha) = 0$.

When $h = k$, in case J^+, J^- and J^{-1} exist, we have $J^+ = J^- = J^{-1}$. That means our method is similar to Newton's method when the system has n equations in n variables. Furthermore, we have $J \cdot J^+ = I$, $J^- \cdot J = I$.

3 Main Algorithm

In this section, we elaborate the main algorithm. But before that, we demonstrate the main ideas of the algorithm by an example. Consider the space algebraic curve defined by

$$p_1 := x^4 + y^4 + z, \quad p_2 := z + y^2.$$

We first compute all extremal points (including singularities) of the algebraic set in a direction, say x. The system of equations needed is $\{p_1(x, y, z) = 0$, $p_2(x, y, z) = 0$, $\det(J_x) = 0\}$ where $J_x = \left(\frac{\partial p_i}{\partial \xi_j}\right)$, $i = 1..2$, $j = 1..2$; $\xi_1 = y$, $\xi_2 = z$ and $\det(J_x) \not\equiv 0$. The actual system of equations now becomes

$$\mathcal{S} = \begin{cases} x^4 + y^4 + z = 0, \\ z + y^2 = 0, \\ 4y^3 - 2y = 0. \end{cases}$$

We triangularize the system by using primitive polynomial remainder sequences (see CASA's manual for `bsolve` and `tsolve`), getting two systems of equations

$$[-1/4 + x^4, -y^2 + 1/2, -x^4 - y^4 - z] \quad \text{and} \quad [2x^4, y, -x^4 - y^4 - z].$$

Note that the union of the two new algebraic sets is equivalent to the algebraic set of the system $\mathcal{S}$. Isolate all of the real roots of the first equation of the triangularized systems, which are univariate, using algorithms based on Descartes' lemma (see [Usp48]) getting

$$[0,0], \quad [\frac{-46341}{65536}, \frac{-741455}{1048576}], \quad [\frac{741455}{1048576}, \frac{46341}{65536}].$$

These are three intervals containing three extremal points with respect to direction x (the width of the interval can be made as small as we need). The advantages of the above computations are that they are very efficient, moreover they help to overcome the known problems of Maple's solver. A system of several equations can also be solved in the same manner. That means, we are able to visualize intersections of several surfaces as well. However, to keep the presentation simple, we only present the two-surface case.

For each interval $\mathcal{I}_i = [x_{i_1}, x_{i_2}]$ on the x axis between two extremal points, we compute all common solutions of the intersection and the hypersurface $x = \frac{x_{i_1}+x_{i_2}}{2}$. For example, the common solutions of the intersection and the hypersurface $x = -1/4$ are the solutions of the system

$$[1/256 + y^4 + z, z + y^2].$$

To solve the system, we can use the same scheme as above or compute the resultant of the two polynomials with respect to z getting a univariate polynomial in y

$$A_i(y) := 1 + 256y^4 - 256y^2$$

Isolate all of the real roots of the univariate polynomial using algorithms based on Descartes' lemma getting a list of intervals together with the defining irreducible polynomial. In this example, the defining irreducible polynomial for all real roots is $A_i(y)$. Using these intervals and their irreducible polynomials, we can compute the corresponding z-values by using algorithms for univariate polynomials with real algebraic coefficients (see [Tra95b]). They are

$$\begin{array}{lrr}
[-.2500000000, & -.9980372563, & -.9960708618] \\
[-.2500000000, & -.0626229122, & -.0039291382] \\
[-.2500000000, & .0626229122, & -.0039291382] \\
[-.2500000000, & .9980372563, & -.9960708618]
\end{array}$$

Again, the advantages of the above computations are that they are very efficient, moreover they help to overcome the known problems of Maple's solver. These solutions are used as the starting points for tracing the intersection.

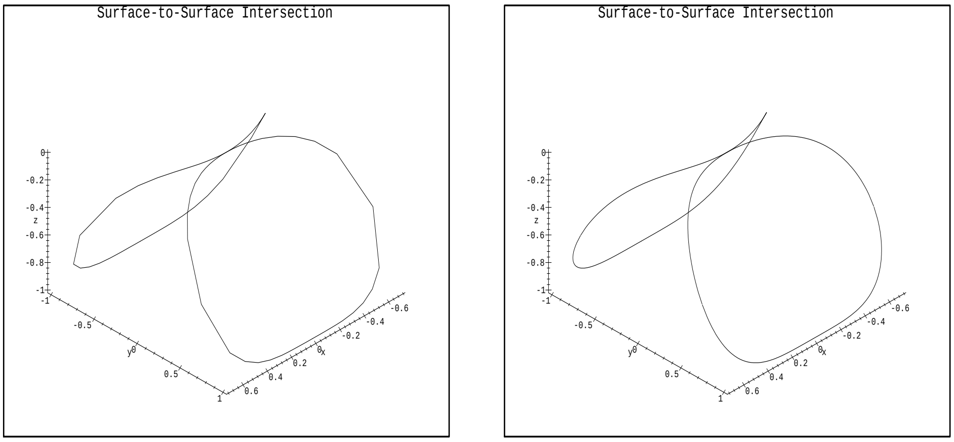
From each starting point, trace the simple path of the intersection, which is defined by a system of two equations in three variables, by using the "Extended Newton Method" to the nearby two extremal points on two directions. Again, we are able to deal with the intersections of several surfaces using the same scheme as above.

We only compute some points on the intersection because the computation is not efficient. We rather utilize NURBS to piecewise interpolate the intersection by invoking CGAL's application module for NURBS. The computation is very fast, for example it costs only 0.02 second to compute 1000 points needed for visualizing the intersection. Figure 1 shows the graphs of the intersection without and with NURBS piecewise interpolation. The time consuming for two graphs are nearly the same thanks to fast computation of points on NURBS curves.

Algorithm 3.1 (Main)

Inputs: A one-dimensional algebraic set of the intersection of two surfaces defined implicitly by two trivariate polynomials $p_1(x,y,z)$*,* $p_2(x,y,z) \in \mathbb{Q}[x,y,z]$*.*

Figure 1: Plotting without and with NURBS piecewise interpolation



Output: A space graph of the algebraic set $A = \{(x, y, z) \in \mathbb{R}^3 : p_1(x, y, z) = p_2(x, y, z) = 0\}$.

1. *Compute all extremal points (including singularities) of the algebraic set* A *in a direction, say* x, *by triangularizing the system of* 3 *equations in* 3 *variables* $\{p_1(x, y, z) = 0,\ p_2(x, y, z) = 0,\ \det(J_x) = 0\}$ *where* $J_x = \left(\frac{\partial p_i}{\partial \xi_j}\right)$, $i = 1..2$, $j = 1..2$; $\xi_1 = y$, $\xi_2 = z$ *and* $\det(J_x) \not\equiv 0$. *Isolate all of the real roots of the first equation of the triangularized systems using algorithms based on Descartes' lemma.*

2. *For each interval* $\mathcal{I}_i$ *on the* x *axis between two extremal points,*

   (a) *Compute the resultant* $A_i(y) = \mathrm{res}_z(p_1(x_i, y, z),\ p_2(x_i, y, z))$ *where* x_i *is the rational middle point of the interval.*

   (b) *Compute the greatest square free divisor* $\bar{A}_i(y)$ *of* $A_i(y)$.

   (c) *Compute* $\mathrm{Lc} := \mathrm{gcd}(\mathrm{gcd}(\mathrm{lcoeff}_z(p_1(x_i, y, z)), \mathrm{lcoeff}_z(p_2(x_i, y, z))), \bar{A}_i(y))$.

   (d) *Isolate all of the real roots of* $\bar{A}_i(y)$ *by using algorithms based on Descartes' lemma for univariate polynomials with integral coefficients getting a list of intervals* $\mathrm{Ical\_I_i}$

   (e) *Remove intervals in* $\mathrm{Ical\_I_i}$ *which is also an interval for a real root of* Lc.

3. *For all* i, *for each interval* $\mathrm{Ical_{i,j}}$ *in* $\mathrm{Ical\_I_i}$, $\mathrm{Ical_{i,j}}$ *and* $\bar{A}_i(y)$ *define a real algebraic root of* $\bar{A}_i(y)$, *say* α.

   (a) *Compute* $q = \mathrm{gcd}(p_1(x_i, \alpha, z),\ p_2(x_i, \alpha, z))$.

   (b) *Isolate all of the real roots of* $q(x_i, \alpha, z)$ *by using algorithms for univariate polynomials with real algebraic coefficients up to the precision* ϵ, *getting a list of intervals* $\mathrm{Ivl\_I_{i,j}}$.

4. *From each initial point defined by a member of the list of intervals* $\mathrm{Ivl\_I_{i,j}}$, *trace the simple path of the intersection, which is defined by a system of two equations in three variables, by using the "Extended Newton Method" to the nearby two extremal points on two directions.*

5. *Piecewise interpolate each branch by using C^1 cubic NURBS (see Section 4 for more details).*

6. *Represent the space curve.*

Remark on the input specification

When the intersection is not simple, the ranges for variables may automatically be defined from the extremal points, which are finite. A tolerance ϵ may also automatically be defined from the ranges for variables and the size of the pixel which is fixed for each screen.

Remark on Step 1

We may have an intersection without any extremal point or singularity, namely the simple space curve.

Remark on Step 2

We have to take into account the cases when the $\mathrm{lcoeff}_z(p_1(x_i, y, z))$ and $\mathrm{lcoeff}_z(p_2(x_i, y, z))$, which are polynomial in y, have a common real root. In fact, we do not need to compute the primitive part of $p_1(x_i,\ y, z)$ and $p_2(x_i, y, z)$ because they will not have a common factor, which is a univariate polynomial in y (otherwise, the intersection will have dimension greater then 1).

Remark on Step 3

This step, in fact, computes the starting point for all paths of the intersection between two extremal points. Therefore, we have to isolate them exactly. Otherwise we could loss some paths of the intersection.

Remark on Step 4

Every path between two extremal points is simple. That means that we do not have to trace through singularities.

4 NURBS Piecewise Interpolation

Remember that the main goal of the method is to help getting a fast and accurate insight of the intersection. While Step 1–4 of the algorithm ensure the accuracy, we speed-up the algorithm by reducing the number of points computed by Step 4 and utilizing NURBS to piecewise interpolate the intersection. We use NURBS because it provides a concise formulation for curves that are composed of many polynomial pieces including conic sections and rational Bézier curves, thereby automatically controlling the overall curve smoothness.

Once the NURBS curves are defined, a system call to CGAL library will quickly return a large number of points needed for visualizing the intersection.

To interpolate the intersection, we use a local algorithm because local algorithms are more geometric in nature, use only local data for each step and usually computationally less expensive then global methods. More precisely, we construct a C^1 cubic NURBS for each branch of the intersection. The main reasons for the use of C^1 cubic NURBS are that cubics easily handle 3-dimensional data and inflection points without special treatment and the tangent vectors are easy to compute (except at the end points where we need to approximate rather than expensively compute them).

Let $\{Q_k\}$, $k = 0, \ldots, n$, be a set of points on the intersection. We follow [PT97] to construct a cubic Bézier curve segment $C(u)$ between each pair Q_k, Q_{k+1}. Denote the Bézier control points by

$$P_{k,0} = Q_k, \quad P_{k,1}, \quad P_{k,2}, \quad P_{k,3} = Q_{k+1}.$$

We must determine suitable locations for $P_{k,1}$ and $P_{k,2}$ along T_k and T_{k+1} respectively, where T_k is the tangent vector at Q_k.

If Q_k is not an endpoint of the branch, it is easy to compute T_k from the equations of the intersection. Q_k is an endpoint means it may be an extremal point or even a singularity. In this case, we approximate the tangent line by

$$T_k = \frac{V_k}{||V_k||}, \quad \text{where} V_k = (1-\beta_k)q_k + \beta_k q_{k+1}, \beta_k = \frac{||q_{k-1} \times q_k||}{||q_{k-1} \times q_k|| + ||q_{k+1} \times q_{k+2}||}$$

for $k = 2, \ldots, n-2$.

We now construct a C^1 continuous cubic with equal speed at each Q_k and at the midpoint of each Bézier segment. Set $\bar{u}_0 = 0$. The $\bar{u}_{k+1}$, $k = 0, \ldots, n-1$, and the two inner control points of $C_k(u)$ are computed as follows:

$$P_{k,1} = P_{k,0} + \frac{1}{3}\alpha T_0, \quad P_{k,2} = P_{k,3} - \frac{1}{3}\alpha T_3,$$

where α is the positive root of the equation

$$a\alpha^2 + b\alpha + c = 0 \quad \text{with} \left\{ \begin{array}{lcl} a & = & 16 - ||t_0 + T_3||^2, \\ b & = & 12(P_3 - P_0) \cdot (T_0 + T_3), \\ c & = & -36||P_3 - P_0|| \end{array} \right.$$

Set $\bar{u}_{k+1} = \bar{u}_k + 3||P_{k,1} - P_{k,0}||$. This algorithm yields n Bézier segments, each having speed equal to 1 at their end- and midpoint with respect to their parametric range, $[\bar{u}_k, \bar{u}_{k+1}]$. Thus a C^1 continuous cubic B-spline curve interpolating the Q_k is defined by the control points

$$Q_0, P_{0,1}, P_{0,2}, P_{1,1}, P_{1,2}, \ldots P_{n-1,1}, P_{n-1,2}, Q_n.$$

and the knot vector

$$U = \{0, 0, 0, 0, \frac{\bar{u}_1}{\bar{u}_n}, \frac{\bar{u}_1}{\bar{u}_n}, \frac{\bar{u}_2}{\bar{u}_n}, \frac{\bar{u}_2}{\bar{u}_n}, \ldots, \frac{\bar{u}_{n-1}}{\bar{u}_n}, \frac{\bar{u}_{n-1}}{\bar{u}_n}, 1, 1, 1, 1\}$$

5 Correctness and Convergence of the Main Algorithm

In this section, we will prove some theorems which are concerned with the correctness and convergence of the main algorithm.

In Step 3, we use the most efficient algorithm for isolating the roots of a polynomial namely the method based on Descartes' lemma. But this method requires that the polynomial has to be square free. Meanwhile the greatest square free divisor computation is very expensive for polynomials with real algebraic coefficients. Fortunately, we can avoid that expensive computation. In fact, the correctness of the step is guaranteed by the following theorem.

Theorem 5.1 *For all $\mathcal{I}_i$, for all α, $q(x_i, \alpha, z)$ is square free in $\mathbb{Z}[\alpha][z]$.*

Proof. Otherwise the intersection would have an extremal point at $x = x_i$. $\square$

Meanwhile, the correctness and convergence of Step 4 come from the following theorems where all of the proofs can be found in [Tra97].

Given a function $f : \mathbb{C}^k \to \mathbb{C}^h$, and a point $x^{(0)} = [x_1^{(0)}, x_2^{(0)} \ldots x_k^{(0)}]^T$, we construct a sequence:

$$x^{(n+1)} = g(x^{(n)}) \tag{4}$$

where g is defined by equation (3). Now we consider some conditions in which the sequence will converge:

Theorem 5.2 *Let $g(x)$ has a fixed point $x = \alpha$. Let the components $g_i(x)$ have continuous first partial derivatives and satisfy*

$$\left|\frac{\partial g_i(x)}{\partial x_j}\right| \le \frac{\lambda}{k}, \ \ \lambda < 1; \tag{5}$$

for all x in

$$||x - \alpha||_\infty \le \rho \tag{6}$$

Then:

(i) For any $x^{(0)}$ satisfying (6) all the iterates $x^{(n)}$ of (4) also satisfy (6).

(ii) For any $x^{(0)}$ satisfying (6), the iterates converge to the root α of $x = g(x)$ which is unique in (6).

We have the following corollary for other norms.

Corollary 5.1 *The Theorem 5.2 is still true for sum-norm and p-norm.*

If the initial iterate $x^{(0)}$ is sufficiently close to the root α of $f(x) = 0$, then the previous theorems can be used to prove that the iterates, $x^{(n)}$, defined in (4) converge to the root of f. However, we do not know from these results if a given initial iterate $x^{(0)}$ is close enough to the unknown root, α. Instead, from Theorems 5.2, we can derive some conditions for the method to be convergent which depend only on properties of the original function f instead of g; and these conditions may be explicitly checked without a knowledge of α.

Theorem 5.3 *Let the initial iterate $x^{(0)}$ be such that the Jacobian matrix $J(x^{(0)})$ defined in (1) has a half–inverse with norm bounded by*

$$||J^+(x^{(0)})|| \le a \tag{7}$$

Let the difference of the first two iterates be bounded by

$$||x^{(1)} - x^{(0)}|| = ||J^+(x^{(0)}) \cdot f(x^{(0)})|| \le b \tag{8}$$

Let the components of $f(x)$ have continuous second derivatives which for all x in $||x - x^{(0)}|| \le 2b$; $i, j = 1..k$, satisfy

$$\sum_{j'=1}^{k} \left|\frac{\partial^2 f_i(x)}{\partial x_j\, \partial x_{j'}}\right| \le \frac{c}{k}, \tag{9}$$

Let the Jacobian matrix $J(x^{(n)})$ has a norm bound on its half–inverse (which always exists)

$$||J^+(x^{(n)})|| \le ||(I - A_n)^{-1}||.||J^+(x^{(n-1)})|| \tag{10}$$

for all $n = 1, 2\ldots$, where $A_n = J^+(x^{(n-1)}).(J(x^{(n-1)}) - J(x^{(n)}))$, and therefore $J(x^{(n)}) = J(x^{(n-1)})(I - A_n)$.

If the constants a, b and c are such that

$$a \cdot b \cdot c \le 1/2. \tag{11}$$

Then

(i) The iterates are uniquely defined and lie in the "2b–sphere" about $x^{(0)}$:

$$||x^{(n)} - x^{(0)}|| \leq 2b.$$

(ii) The iterates converge to some vector, say $\lim_{n\to\infty} x^{(n)} = \alpha$, for which $f(\alpha) = 0$, and

$$||x^{(n)} - \alpha|| \leq \frac{2b}{2^n}. \tag{12}$$

Lemma 5.1 *If the function $f(x)$ has two derivatives and $(J^* \cdot J)$ is non-singular at the root then the convergence of the extended method is quadratic.*

6 Experiments

The method has been implemented in CASA. It calls symbolic procedures from CASA or numerical procedures from CGAL when needed. All the experiments were carried out on a SUN Ultrasparc-1 workstation under Solaris 2.5 with 128MB RAM.

Figure 2 shows the graphs produced by the program for the following examples taken from literature.

1. $x - 1,\ y - 1$
2. $x^2 + y^2 + z^2 - 5,\ xyz - 1$
3. $z - 2x^4 - y^4,\ z - 3x^2y + y^2 - x^3$
4. $z - 2x^4 - y^4,\ z^2 - 3x^2y - y^3$
5. $z^2 + 3x^4 - y^4,\ z + x^3 - 3x^2y + y^2 - 2y^3$
6. $y^4 - 384y^2 + 400x^2 - x^4 + z^3,\ x^2 + y^2 + z^2 - 30$

Figure 3 reports the time comparison between the new hybrid method and a pure symbolic method.
Although Maple provides a function which is claimed to plot 2D sets in an implicit representation its output for more complicated varieties is by far not satisfactory. Fortunately, a plane algebraic curve can be seen as a special case in this method as the intersection with the plane $z = 0$ (Actually, we have a much faster algorithm for plane algebraic curves). Figure 4 shows the graphs produced for the following examples:

1. $\frac{93392896}{15625}x^6 + (\frac{94359552}{625}y^2 + \frac{(91521024}{625}y - \frac{249088}{125}x^4 + (\frac{1032192}{25}y^4 - 36864y^3 - \frac{7732224}{25}y^2 - 207360y + \frac{770048}{25}x^2 + (65536y^6 + 49152y^5 - 135168y^4 - 72704y^3 + 101376y^2 + 27648y - 27648$
2. $2x^4 - 3x^2y + y^2 - 2y^3 + y^4$
3. $y^4 - 96a^2y^2 + 100a^2x^2 - x^4$
4. $8(y^2 - x^2) - 8(y^3 + x^3) + 2y^4 + x^5$

7 Conclusion

We have developed a hybrid symbolic-numerical algorithm for tracing surface-to-surface intersections – a fundamental and difficult problem in Computer Aided Geometric Design (CAGD) and solid modeling. By using numerical techniques and exact methods rationally, we try to satisfy two basic prerequisites for the effective and practical use of the method, namely reliability and efficiency. The method help to get a fast and accuracy insight of the intersections. It can also be used to analyze the topology of plane algebraic curves as well. Fortunately, in this case the algorithm is simpler and more efficient due to avoiding computations with real algebraic number at Step 3 of the main algorithm. Therefore plane algebraic curves with high degree can be traced. Theoretically, the method can be extended to tracing one-dimensional algebraic sets of the intersection of more than two surfaces or more general, $n-1$ equations in n variables. In that case, Step 2, and Step 3 of the main algorithm have to change to another exact method for solving zero-dimensional systems of n equations in n variables, for example the Gröbner Bases method.

Acknowledgments

We thank Daniela Văsaru and Markus Wabro for their help in implementing the NURBS application module in CGAL. We also thank Wolfgang Freiseisen for his help in integrating the application module into CGAL.

References

[ACM88] D. S. Arnon, G. E. Collins, and S. McCallum. An adjacency algorithm for cylindrical algebraic decompositions of three-dimentional space. *J. Symbolic Computation*, 5:163–187, 1988.

[BFJP87] R.E. Barnhill, G. Farin, M. Jordan, and B.R. Piper. Surface/surface intersection. *Comp. Aided Geom. Design*, 4:3–16, 1987.

[BHLH88] C. L. Bajaj, C. M. Hoffmann, R. E. Lynch, and J. E. H. Hopcroft. Tracing Surface Intersections. *Computer Aided Geom. Design*, 5:285–307, 1988.

[BO71] T. L. Boullion and P. L. Odell. *Generalized Inverse Matrices*. Wiley–Interscience, New York, 1971.

[FGK+96] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schoenherr. The cgal kernel: A basis for geometric computation. In *Proceedings of the workshop on Applied Computational Geometry*, Philadelphia, Pennsylvania, May 27–28 1996.

[GKWW91] R. Gebauer, M. Kalkbrener, B. Wall, and F. Winkler. CASA: A Computer Algebra Package for Constructive Algebraic Geometry. In S. M. Watt, editor, *ISSAC '91*, pages 403–410, Bonn, Germany, July 1991.

[Hof89] C. M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann Publishers, Inc., 1989.

[KP91] D. J. Kriegman and J. Ponce. A new curve tracing algorithm and some applications. In P. J. Laurent, Le Méhauté, and L. Schumaker, editors, *Curves and Surfaces*, 1991.

[KPP90] G. A. Kriezis, P. V. Prakas, and N. M. Patrikalakis. A method for intersecting algebraic surfaces with rational polynomial patches. *Computer Aided Design*, 22(10):645–654, 1990.

[MW96] M. Mňuk and F. Winkler. CASA - a system for computer aided constructive algebraic geometry. In *Proceedings of the International Symposium DISCO'96*, pages 297–307, 1996.

[Ove96] Mark H. Overmars. Designing the computational geometry algorithms library cgal. In *Proceedings of the workshop on Applied Computational Geometry*, Philadelphia, Pennsylvania, May 27–28 1996.

[Pat92] N. M. Patrikalakis. Interrogation of surface intersections. In R. E. Barnhill, editor, *Geometry Processing for Design and Manufactoring*, pages 161–185. SIAM, Philadelphia, 1992.

[Pat93] N. M. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics & Applications*, 1(1):89–95, 1993.

[PT97] Les Piegl and Wayne Tiller. *The NURBS Book*. Monographs in Visual Communications. Springer, Berlin, 2nd edition edition, 1997.

[Sch96] Stefan Schirra. Designing a computational geometry algorithms library. Lecture Notes for Advanced School on Algorithmic Foundations of Geographic Information Systems, CISM, Udine, September 16–20 1996.

[Tau94] Gabriel Taubin. Rasterizing algebraic curves and surfaces. *IEEE Computer Graphics and Applications*, March 1994.

[Tra94] Quoc-Nam Tran. Extended Newton's method for finding the roots of an arbitrary system of equations. In M. H. Hamza, editor, *Proceedings of the Twelfth IASTED International Conference: Applied Informatics*. IASTED, 1994.

[Tra95a] Quoc-Nam Tran. A hybrid symbolic-numerical method for tracing surface-to-surface intersections. In A. H. M. Levelt, editor, *Proceedings of ISSAC-95*, Montreal, Canada, 1995. ACM.

[Tra95b] Quoc-Nam Tran. On the symbolic-numerical methods for finding the roots of an arbitrary system of non-linear algebraic equations. In *Proceedings of the First Asian Technology Conference in Mathematics, ATCM-95*, Singapore, 1995.

[Tra96] Quoc-Nam Tran. *A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization*. PhD thesis, Research Institute for Symbolic Computation (RISC–Linz), University of Linz, Austria, 1996.

[Tra97] Quoc-Nam Tran. Extending newton's method for finding the roots of an arbitrary system of equations and its applications. *International Journal of Modeling and Simulation*, 17(4), 1997. To appear.

[TW97] Quoc-Nam Tran and Franz Winkler. An overview of CASA – a system for computational algebra and constructive algebraic geometry. In T. Racio T. V. Effelterre and F. Winkler, editors, *The Symbolic and Algebraic Computation (SAC) Newsletter*, volume 2. Stichting CAN, Computer Algebra Nederland, Universiteit van Amsterdam, the Netherlands, 1997.

[Usp48] J.V. Uspensky. *Theory of Equations*. McGraw-Hill Book Company, Inc., Cambridge, 1948.

[W+91] Y. Wang et al. Intersection of parametric surfaces for next generation geometric modeling systems. In J. Turner, J. Pegna, and M. Wozny, editors, *Product Modeling for Computer Aided Design and Manufacturing*, pages 75–96. Elsevier, 1991.

[Wal93] Bernhard Wall. *Symbolic Computation with Algebraic Sets*. PhD thesis, RISC-Linz, Universität Linz, 1993.

[Win93] Franz Winkler. Algebraic computation in geometry. In G. Jacobs, N. E. Oussous, and S. Steinberg, editors, *IMACS'93 Conference, Lille, special issue J. Math. in Comp. Simulation*, 1993.

Figure 2: Surface-to-Surface Intersections

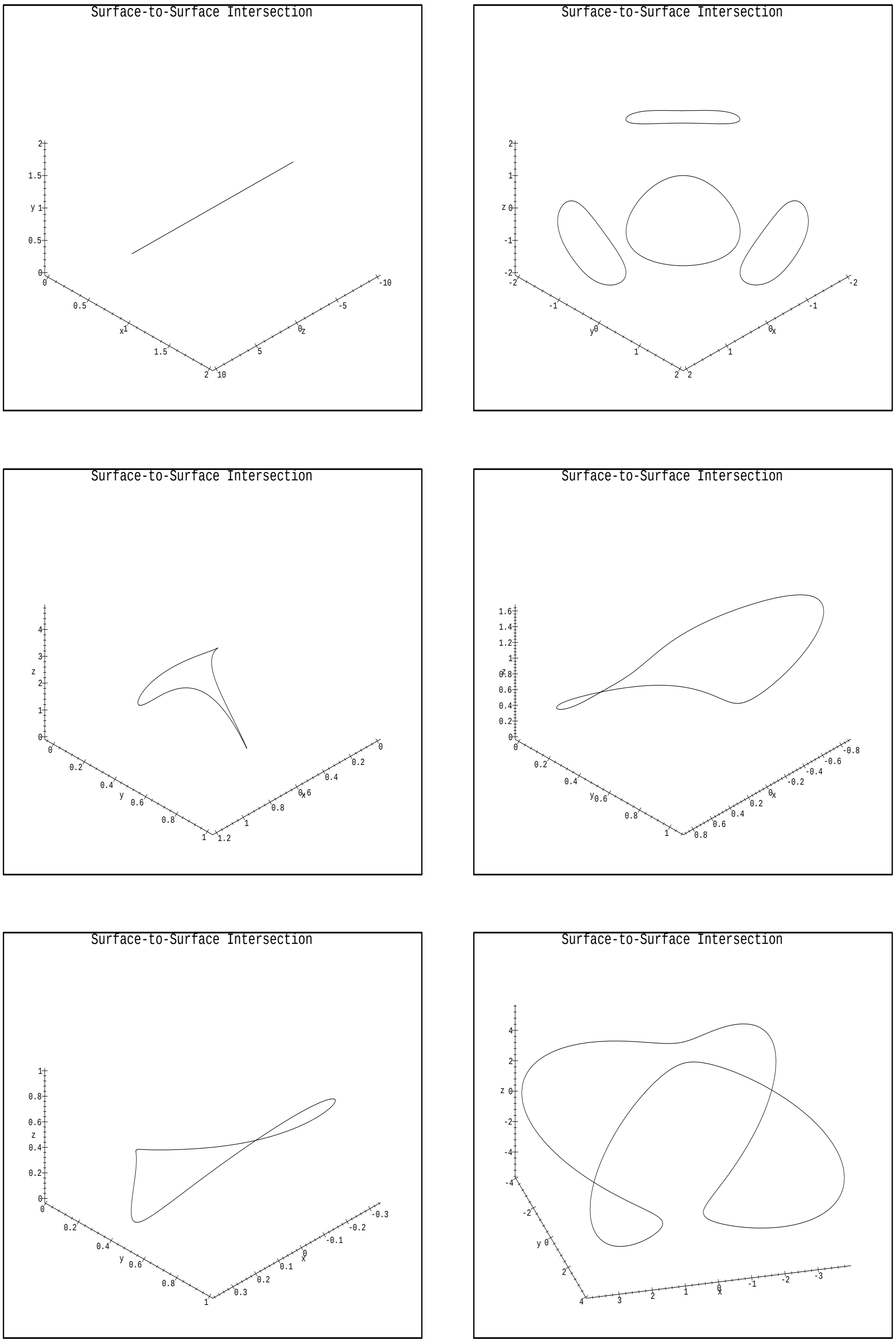


Figure 3: Time Comparison

	Exact Method (second)	Hybrid Method (second)
Example 1	11.120	0.230
Example 2	89.090	3.250
Example 3	> 50000.000	1.500
Example 4	> 50000.000	2.780
Example 5	> 50000.000	3.720
Example 6	> 50000.000	13.650

Figure 4: Plane Algebraic curves