Efficient B-rep Generation of Low Degree Sculptured Solids using Exact Arithmetic *

John Keyser keyser@cs.unc.edu

Shankar Krishnan krishnas@cs.unc.edu

Dinesh Manocha manocha@cs.unc.edu

Department of Computer Science,
CB#3175 Sitterson Hall,
University of North Carolina,
Chapel Hill, NC 27599-3175

Abstract

We present efficient representations and algorithms for exact boundary computation on low degree sculptured CSG solids using exact arithmetic. Most of the previous work using exact arithmetic has been restricted to polyhedral models. In this paper, we generalize it to higher order objects, whose boundaries are composed of rational parametric surfaces. The use of exact arithmetic and representation guarantees that a geometric algorithm is numerically robust. Furthermore, it can be combined with symbolic perturbation techniques to handle degeneracies.

Most of the current modelers use finite precision arithmetic for boundary computation. Exact arithmetic is rarely used due to the perception that its overhead is very high. In this paper, we present compact data structures for representing the boundary as algebraic curves as well as the intersection points as algebraic numbers. In addition, we present efficient algorithms for computing the intersection curves of trimmed parametric surfaces, decomposing them into multiple components for efficient point location queries inside the trimmed regions, and computing the boundary of the resulting solid using topological information and component classification tests. We also employ a number of previously developed algorithms like algebraic curve classification, multivariate Sturm sequences, and multivariate resultants. We have implemented these algorithms and highlight their performance on Boolean combinations of degree two (e.g. quadrics) and degree four solids. For a single Boolean operation, the performance of the algorithm depends on the degrees of

*Supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract P-34982-MA, NSF Grant CCR-9319957, NSF Grant CCR-9625217, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization

the solid, their relative position and the output complexity of the final B-rep (in terms of number of faces, edges and vertices). It is about one order of magnitude slower than B-rep computation algorithm implemented using floating point arithmetic (on low degree solids).

1 Introduction

Constructive Solid Geometry (CSG) and Boundary Representations (B-rep) are two major approaches to represent solids [Bra75, RV82, RV85, Hof89, Man88]. While the CSG implicitly represents a solid as an algebraic expression, B-rep explicitly stores an object as a set of surfaces. Necessary for any B-rep are ways of storing and computing the surfaces of an object, curves on a surface or between two surfaces, and points on a surface or on a curve. For example, in the polyhedral domain, the surfaces, curves, and points correspond to faces, edges, and vertices. Note that this terminology may be used interchangeably from here on out to refer to the corresponding parts in a curved surface domain.

The Boolean combinations (union, difference, intersection) of objects are some of the common operations which are performed on solid models. This is especially true when trying to convert from CSG to B-rep. Most of the current solid modeling systems are based on B-reps. Computing the B-rep of the resulting solid (after performing Boolean operations) is an important operation in these systems. In this paper the objects correspond to sculptured solids, whose boundary can be represented using trimmed rational parametric surfaces. This is a wide family of objects used in geometric and solid modeling and can exactly represent quadrics, tori and free-form solids.

The first systematic study of CSG to B-rep conversion appeared in [RV85] and nowadays the algorithms for conversion are relatively well understood [Hof89, Man88, CB89, MB91, Men92, Sar83, Wei85, KM96]. However, the problem of *robust* and *accurate* computation of the boundary is considered one of the difficult problems in geometric and solid modeling [Hof96, ea95, For96]. It is important that the computed B-rep be accurate, or at least topologically consistent, and this can be jeopardized by even small amounts of error in the representation of the model or in finite-precision computations (e.g. round-off errors). An example is when error results in incorrect choices being made regarding the connectivity of the faces or curves, thus generating an incorrect and inconsistent model.

A number of approaches have been proposed for robust and accurate B-rep computation. Most of them are restricted to polyhedral modelers. One of the most common approaches is based on using *tolerances* with floating-point arithmetic [Jac95]. If two geometric elements are within the given tolerance, they are considered incident. However, it is hard to decide a global tolerance value for all computations. To circumvent these problems, combinations of symbolic reasoning [HHK89] and adaptive tolerances [Seg90] have been proposed. Other algorithms include those based on redundancy elimination [RV89, FBZ93].

B-rep computation algorithms involve accurate evaluation of the sign of arithmetic expressions. Algorithms based on floating-point arithmetic are at times ambiguous, when the value of the expression is close to zero. If this ambiguity is not properly addressed, the resulting algorithm becomes *unreliable*. Many algorithm based on *exact arithmetic* have been proposed for reliable numeric computation for polyhedral domains [SI89, For95, BMP94, Hof89]. These algorithms use a fixed upper bound on the bit-length of arithmetic required to evaluate geometric predicates.

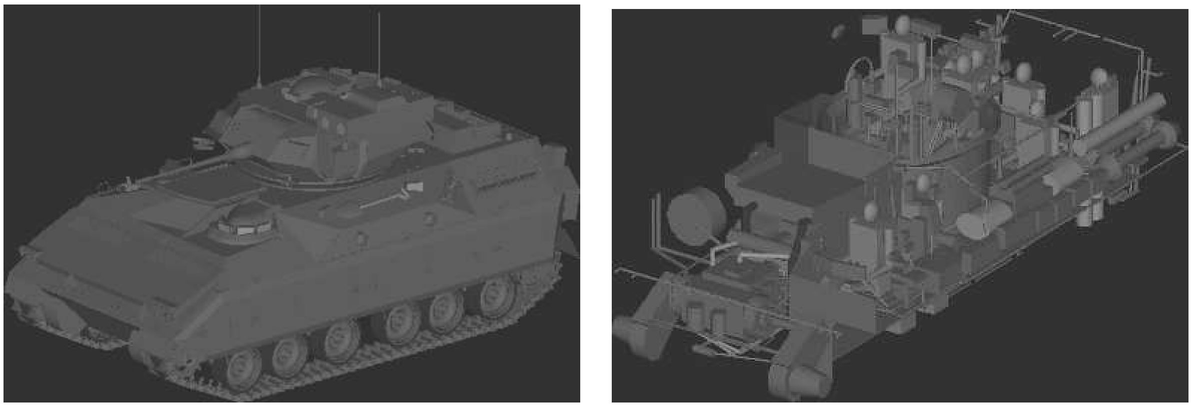

Figure 1: Exterior and Interior of a Bradley fighting vehicle

In particular, Fortune has presented an efficient algorithm based on exact arithmetic which has a small performance overhead as compared to a floating-point based implementation [For95]. Besides reliable computation, exact arithmetic allows the use of symbolic perturbation to handle degeneracies [Yap90]. The perturbation scheme greatly simplifies the implementation of the solid modeler.

There is relatively little work on robust B-rep computation algorithms for curved primitives. Algorithms to handle degenerate intersections between quadrics have been presented in [MG91, SJ91, FNO89]. For arbitrary degree sculptured solids, it is difficult to compute *tight bounds* on the error generated due to floating-point arithmetic. As a result, it is hard to extend algorithms based on tolerances to curved models. Furthermore, exact arithmetic for curved domains is perceived to be *extremely slow* and *complex*. This is due to a number of reasons. Exact arithmetic involves computations on algebraic numbers and most of the current implementations of such arithmetic (e.g. those available as part of computer algebra systems) are extremely slow. Moreover, many representations (e.g. edges, points) and predicates (e.g. inside/outside tests, orientations) that are well-understood in the linear domain become rather hard in the curved domain. Other techniques for arithmetic on algebraic numbers are based on computing bit-length estimates required for reliable expression evaluation. However, in the worst case, these bounds may require bit-lengths which are exponential function in terms of the degree of the algebraic functions [Can88, Yu92]. Overall, no good solutions are known for efficient and robust B-rep computation on curved solids.

Main Contribution: We present efficient representations and algorithms for exact boundary computation on Boolean combinations of sculptured solids. Our contributions include:

- **Representation:** We present efficient and exact representations for points, edges and surfaces using algebraic sets along with topological representation

- **Kernel Routines:** We identify all the lower-level routines where the algorithms based on floating point arithmetic are susceptible to failure. These include sign-evaluation of geometric predicates, orientation of points with respect to curves and component classification. We present fast (and in a few cases optimized) algorithms to perform such tests *reliably* using exact arithmetic. We refer to the resulting set of routines as *kernel routines*. The efficiency and reliability of the overall algorithm is governed by these routines. Our use of lazy evaluation and bounding volumes help to improve the speed of some of these routines.

- **B-rep Computation:** Given kernel routines, we present an algorithm for B-rep computation on top of such routines.

- **Handling Degeneracies:** We identify most of the cases where degeneracies can effect our algorithm, and propose some techniques to identify and resolve them.

- **Implementation and Performance:** We describe the performance of a preliminary implementation of our algorithm.

The resulting algorithm and system work well on low-degree solids (composed of polyhedra,quadrics, tori, low-degree solids of revolution). In practice, most of the curved primitives of solid modeling systems are indeed low-degree. For example, the Bradley fighting vehicle (shown in Fig. 1), designed using BRL-CAD system [MDA+88], is composed of more than 5000 solids, each defined using $3-12$ Boolean operations on such low-degree solids. Our algorithm is able to compute the exact B-rep of one Boolean operation between two low-degree solids in $40-100$ seconds on a HP 712/100 workstation. As compared to algorithms implemented in floating point arithmetic, our algorithm performs slightly more than one order of magnitude slower on low degree solids on an average.

Organization: The rest of this paper is organized as follows. Section 2 discusses our representation for solids. Section 3 gives an overview of our algorithm and discusses the kernel routines which form the basis of our algorithm. Section 4 discusses how each of the major steps are performed. In Section 5, we present some analysis of our approach along with some performance results and an illustrating example. Section 6 discusses degeneracies and Section 7 concludes with a discussion of possible areas for extensions and future work. Following our list of references, we present an Appendix (Section 8) which gives some background information regarding our use of multivariate Sturm sequences and resultants.

2 Background and Representation of Solids

In this section, we present our representation for a solid. Our algorithms assume that solids are specified in this format, and they return solids in this format. We also present some back-

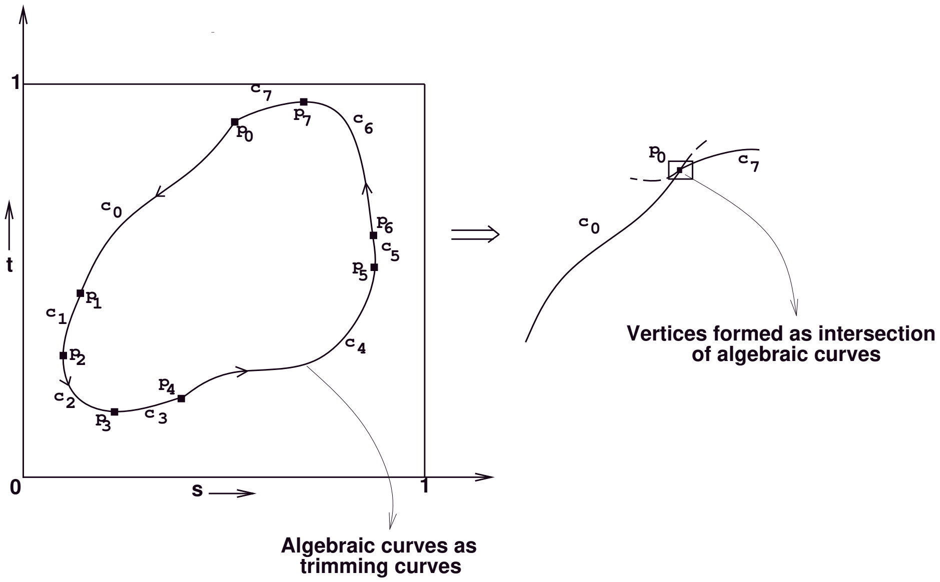

Figure 2: Representation of a trimmed patch as algebraic curve segments

ground material that we use to compute the B-rep. . These include a number of algorithms from computational algebra and numerical analysis. In particular, we shall briefly discuss our representation of algebraic numbers, techniques for root isolation using multivariate Sturm sequences, and multipolynomial resultant computation.

2.1 Representation of Solids

Every solid is represented as a set of *trimmed* parametric surface patches which define the solid boundary. We represent each surface patch $\mathbf{F}(s,t)$ as a *rational function* with *rational* coefficients. This kind of parametrization is possible for all quadric surfaces like spheres, cylinders etc., surfaces of revolution, and also for tori. The domain of the patch is the unit square in the (s,t)-plane $(0 \leq s,t \leq 1)$. If we are given a different domain, we can always reparameterize to $(0 \leq s,t \leq 1)$.

Assumptions: *Topological* information of the solid is maintained in terms of an adjacency graph. It is similar to the winged-edge data structure [Hof89]. To start with, we assume that each of the input objects has *manifold* boundaries, and the Boolean operation is *regularized*. While it is possible to generate non-manifold objects from regularized Booleans on manifold solids, we assume for the sake of simplicity that this does not occur. It is a well-known fact that, while dealing with topological representation of curved objects, global resolution of edge ambiguities cannot be guaranteed at times [Hof89]. Some of these issues are addressed in Section 6. Given these assumptions, it can be shown that an unambiguous topological representation is possible for a solid.

A trimmed patch consists of a sequence of curves defined in the domain of the patch such that they form a closed curve ($\mathbf{c_i}$'s in Fig. 2). In the figure, the $\mathbf{c_i}$ refer to the algebraic curve segments forming the trimming boundary. The portion of the patch that lies in the interior of this closed curve is retained. Most of these trimming curves correspond to intersection curves between two surfaces. Therefore, these curves are typically algebraic curves that do not admit a rational parametrization ([AB88]). We represent these curve segments ($\mathbf{c_i}$) by their algebraic equation and the two endpoints ($\mathbf{p_i}$ and $\mathbf{p_{i+1}}$). The endpoints are found by solving a set of polynomial equations, and are actually algebraic numbers (see Fig. 2). Exact representation of these numbers is discussed later in this section.

This representation of a solid lends itself to a description in terms of *faces, edges, and vertices* analogous to the polyhedral case. Each *face* is a trimmed patch. Each of the trimming curves form an *edge*, and are formed as an intersection of two surfaces (faces). Finally, endpoints of edges form the *vertices*. They can be represented as an intersection of three surfaces. Fig. 3 shows an example solid and the face connectivity structure that we maintain. We also maintain the two faces that are adjacent to each edge, and an anticlockwise order of faces around each vertex.

Representation of algebraic numbers: It was mentioned earlier that each of the vertices in the solid are defined as roots of a set of polynomial equations with rational coefficients. Because of the rational parametrization of the surface, each of these equations is either univariate or bivariate. A vertex in the patch domain is therefore the common solution of two equations, $f(s,t) = 0$ and $g(s,t) = 0$. These are usually algebraic numbers, and cannot be represented exactly as finite precision numbers. Notice that an algebraic number can be defined as the solution of an equation, $f(s) = 0$, within some interval, $a \leq s < b$. In our algorithm, we represent each algebraic coordinate as an arbitrarily small rational rectangle (i.e. an axis-aligned rectangle whose four vertices have rational coordinates). The rational rectangle is guaranteed to isolate each common root of $f(s,t)$ and $g(s,t)$ (taking into account the multiplicities of roots). The root isolation algorithm uses *multivariate Sturm sequences* as proposed by Milne [Mil92].

2.2 Multipolynomial Resultants

Elimination theory investigates the conditions under which sets of polynomials have common roots. Usually, it concerns itself with sets of n homogeneous polynomials in n unknowns, and finds the relationship between the coefficients of the polynomials which can be used to determine whether the polynomials have a non-trivial common solution.

Definition 1 *[Sal85] A* **resultant** *of a set of polynomials is an expression involving the coefficients of the polynomials such that the vanishing of the resultant is a necessary and sufficient condition for the set of polynomials to have a common non-trivial root.*

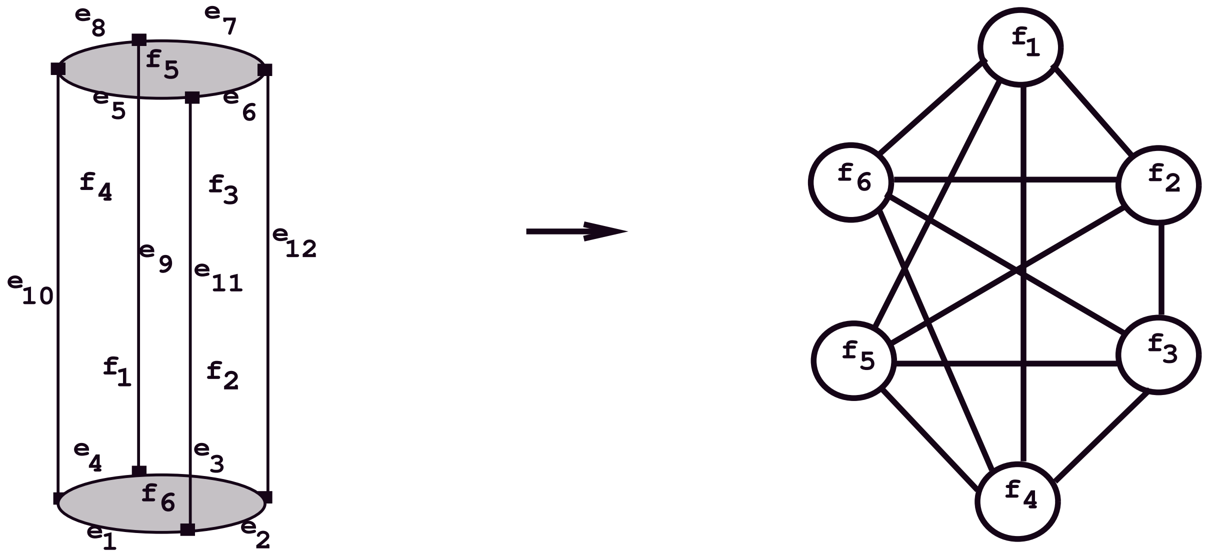

Figure 3: A cylinder and its face connectivity structure

[Mac02] provided a general method for eliminating n variables from n homogeneous polynomials. The resultant is expressed as a ratio of two determinants. However, a single determinant formulation exists for $n = 2$ and 3 [Sal85, Dix08]. In our application, it is sufficient to compute resultants for the cases when $n = 2$ and 3. Sylvester's method [Sal85] can be used to express the resultant of two polynomials of degree m and n respectively as a determinant of a matrix with $(m+n)$ rows and columns. For the polynomials,

$$f^n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \tag{1}$$

and

$$g^m(x) = b_m x^m + b_{m-1}x^{m-1} + \ldots + b_1 x + b_0 \tag{2}$$

where $n > m$, the Sylvester's resultant is

$$\begin{vmatrix} a_n & a_{n-1} & \ldots & & a_0 & 0 & \ldots & 0 \\ 0 & a_n & a_{n-1} & \ldots & a_0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & a_n & a_{n-1} & & \ldots & a_0 \\ 0 & \ldots & & 0 & b_m & b_{m-1} & \ldots & b_0 \\ 0 & \ldots & 0 & b_m & b_{m-1} & \ldots & b_0 & 0 \\ b_m & b_{m-1} & \ldots & b_0 & 0 & \ldots & & 0 \end{vmatrix} \tag{3}$$

The problem of computing the implicit representation of a parametric surface $(\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)))$ involves eliminating s and t from the three polynomials

$$X(s,t) - xW(s,t) = 0, \quad Y(s,t) - yW(s,t) = 0, \quad Z(s,t) - zW(s,t) = 0.$$

We use Dixon's resultant [Dix08] to compute the implicit form as described in [Sed83].

Resultant computation: We use an algorithm based on multivariate interpolation [MC93] to compute the resultant of a set of polynomials efficiently. The main bottleneck in most resultant algorithms is the symbolic expansion of determinants. Most of the computer algebra systems use symbolic algorithms like polynomial manipulations for resultants, which are very expensive. Further, the magnitude of intermediate expressions grows quickly, and the memory requirements are high. The algorithm in [MC93] performs all computations over finite fields, and uses a probabilistic algorithm based on the Chinese Remainder Theorem to recover actual coefficients.

2.3 Multivariate Sturm sequences

Here, we describe briefly the algorithm proposed by Milne [Mil92] to compute the number of common real solutions of n polynomials in n variables inside an n-dimensional rectangle. This algorithm is an extension of the univariate case [Her80] which constructs a polynomial sequence, and measures sign variations of this sequence at the endpoints of the interval. We restrict ourselves to the case when $n = 2$.

Given two polynomials, $f_1(s,t)$ and $f_2(s,t)$, we construct the *volume function*, $V(u,s,t)$, as follows:

$$V(u,s,t) = \frac{Res_{a_2}(Res_{a_1}(f_1(a_1,a_2), f_3), Res_{a_1}(f_2(a_1,a_2), f_3))}{u^{deg(f_1(s,0))deg(f_2(s,0))}},$$

where $f_3(u,s,t,a_1,a_2) = u + (s-a_1)(t-a_2)$, Res_x refers to the resultant of two polynomials after eliminating x, and deg refers to the degree of the polynomial. We use the Sylvester resultant [Sal85] to eliminate one variable from two polynomials.

Given a square-free polynomial $p(x)$ we can construct a Sturm sequence of polynomials $S_i = -remainder(S_{i-2}(x), S_{i-1}(x))$, where $S_1(x) = p(x)$ and $S_2(x) = p'(x)$. Treating the volume function V as a univariate polynomial in u, we construct its Sturm sequence $S_i(u,s,t)$. The Sturm sequence is specialized at $u = 0$ to give a sequence of bivariate polynomials $M(s,t)$.

Definition 2 *Given a sequence of polynomials $M(s,t)$ of length n, the* **var** *operator at (a_1,a_2) ($var(M(a_1,a_2))$) gives the number of sign changes between consecutive terms of the sequence evaluated at (a_1,a_2). Correspondingly, the* **per** *operator is defined as $per(M(a_1,a_2)) = n - 1 - var(M(a_1,a_2))$.*

Given the bivariate sequence $M(s,t)$ and a rational axis aligned rectangle $\Gamma = [a_1,b_1] \times [a_2,b_2]$, the number of real roots of f_1 and f_2 inside Γ is given by

$$\frac{\mathbf{per}(M(b_1,b_2)) + \mathbf{per}(M(a_1,a_2)) - \mathbf{per}(M(b_1,a_2)) - \mathbf{per}(M(a_1,b_2))}{2}.$$

The justification for various steps and extension to arbitrary dimensions can be found in [Mil92].

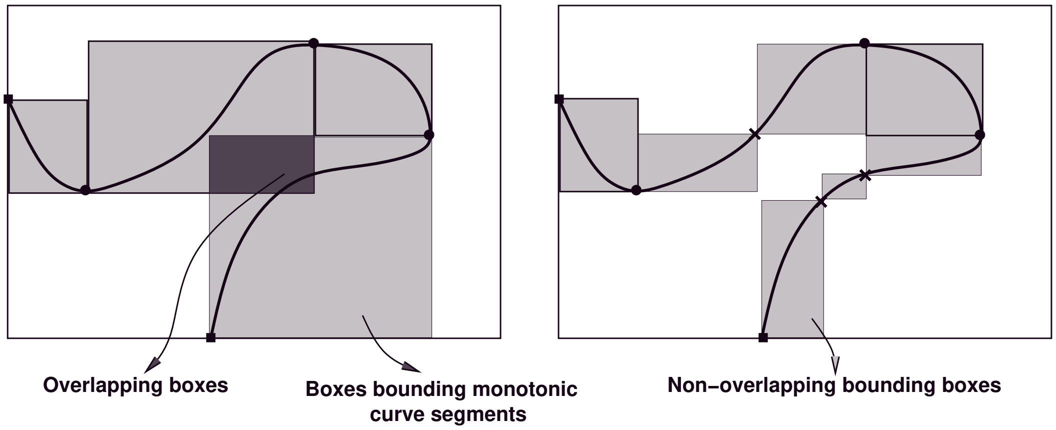

Figure 4: Bounding boxes around monotonic curve segments

2.4 Topological resolution of algebraic curves

The intersection curve between two surfaces is typically a high degree algebraic curve with a number of components. Topological resolution involves identifying critical points like turning points and singularities and establishing a unique connectivity between them. A number of efficient (poly-log time) algorithms have been developed for special kinds of algebraic curves. We use one such algorithm by [AF88] which achieves this for regular curves. The algorithm initially computes all the turning points of the curve. This is achieved in our case by taking partial derivatives and solving for common roots with the original curve inside a rational box. A crucial step in establishing connectivity between the various turning points is to find the sign of the slope of the curve at certain irrational (algebraic) points. We present an exact algorithm to perform this step in section 3.3.2.

The identification of turning points divides the intersection curve into a set of monotonic curve segments. For each of these curves, we compute a bounding box (see Fig. 13) around it. Bounding boxes are needed to disambiguate two such curve segments represented by the same algebraic equation. However, as seen from Fig. 13 , not all the bounding boxes are non-overlapping nor are single curve segments restricted within a bounding box. We perform a subdivision of these boxes until these two criteria are satisfied. We use an algorithm described in [KM94] to perform this subdivision. The bounding boxes defined here are used to classify a point efficiently with respect to a curve.

3 Algorithmic overview and Kernel routines

In this section, we give a brief overview of our algorithm. In Section 2.2, we identify the steps that are susceptible to failure while using finite precision arithmetic. Finally, we describe the kernel routines and present efficient and optimized algorithms using exact arithmetic.

3.1 Computation of Boolean operation

The general outline for performing a Boolean operation between two solids (represented as described in Section 2) follows. The overall algorithm is decomposed into two stages.

I . Intersection curve computation (for each pair of patches):

1. Obtain the intersection curve(s) between the two patches.
2. Find the points where the intersection curve meets the patch boundary.
3. Decompose the intersection curve into a set of monotonic curves.
4. Find the points where the intersection curve meets the trimming boundary, and subdivide the trimming and intersection curves.
5. Compute adjacency information for remaining intersection curves.

II . Curve merging and boundary computation (for each patch):

1. Merge intersection curves together in each patch to partition the patch domain.
2. Compute the adjacency graph and components separated by intersection curves.
3. Shoot rays from one solid to the other solid to classify components as inside or outside of the solid.
4. Propagate the information from step 3 in the adjacency graph to find the final solid.

3.2 Need for exact arithmetic

While dealing with polyhedral solids, most geometric predicates are evaluated by signs of arithmetic expressions. For example, [For95] classifies a point with respect to a plane as the sign of a 4×4 determinant. This is because three planes intersect at one point generically. The 4×4 determinant actually evaluates the fourth plane at the intersection point. This can be extended to the curved surface domain by evaluating the resultant of the four surfaces (by eliminating the 3 variables x, y, z). Three surfaces will, in general, intersect in a number of points. Each of these points will have a sign relative to which side of the fourth surface they lie on. The product of these signs is the sign of the determinant.

We shall now identify the parts where our algorithm is susceptible to failure when using floating point arithmetic. In order to prevent these failures, we use exact arithmetic. Most of these errors finally boil down to either point orientation tests or comparison between two floating point numbers.

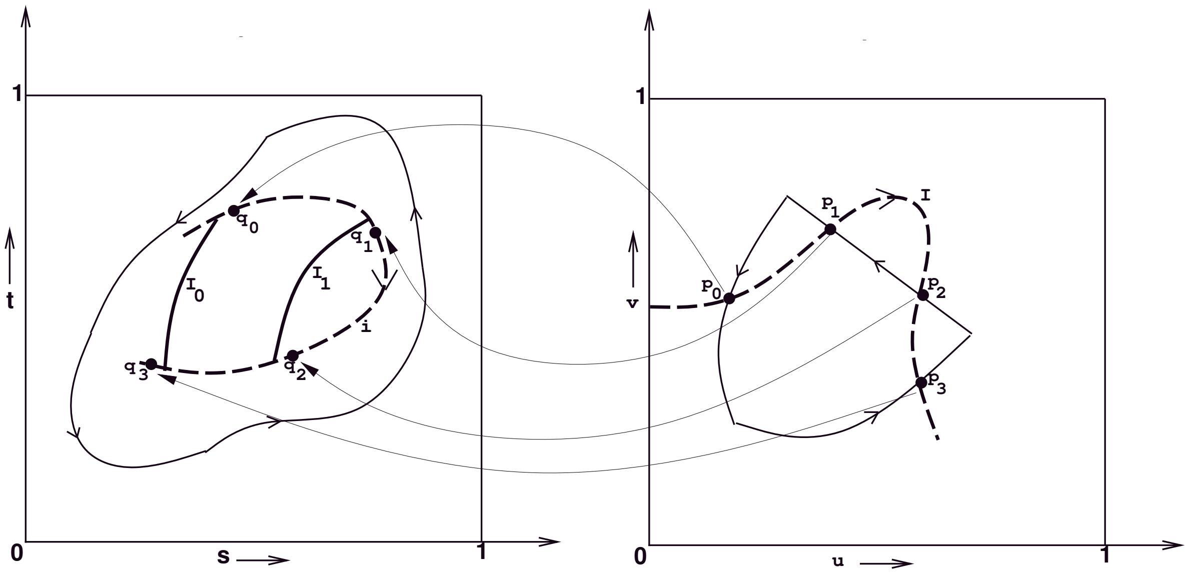

Figure 5: Inaccurate point inversion for curve merging

3.2.1 Computing trimmed patch intersections

Most algorithms using floating point arithmetic for computing the intersection curve use techniques like curve tracing or subdivision. As a result, these curves are approximated as piecewise linear curves or splines to within a fixed tolerance (which are either too conservative or arbitrarily chosen), or as algebraic curves with floating point coefficients. Since most of the surface patches we are dealing with are trimmed, we need to compute portions of the intersection curve that lie inside the trimmed boundaries of both the patches. Fig. 4shows one such example. The curve $\mathbf{I}$ shown in dotted lines is the intersection curve in both the domains. $\mathbf{I_0}$ and $\mathbf{I_1}$ are the intersection curves obtained from other surfaces on the left patch. To compute the actual intersection curve for trimmed patches, we need to compute the intersection points of the curve with the trimming boundary. $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ are four such points on the right patch. Since the boundary curves and the intersection curve are not accurate, neither are the $\mathbf{p_i}$'s. They may not even lie on the actual intersection curve. Corresponding to the $\mathbf{p_i}$'s, we need to compute $\mathbf{q_i}$'s on the other patch to determine which portions of the intersection curve to retain. This process is called *inversion*. One of two cases is possible: (a) there may not be any corresponding point on the other patch (because $\mathbf{p_i}$'s do not lie on the intersection curve), or (b) they could be positioned such that the curve segments $\mathbf{q_0q_1}$ and $\mathbf{q_2q_3}$ do not match up with $\mathbf{I_0}$ and $\mathbf{I_1}$ for curve merging. It is hard to perform this computation reliably using floating point arithmetic.

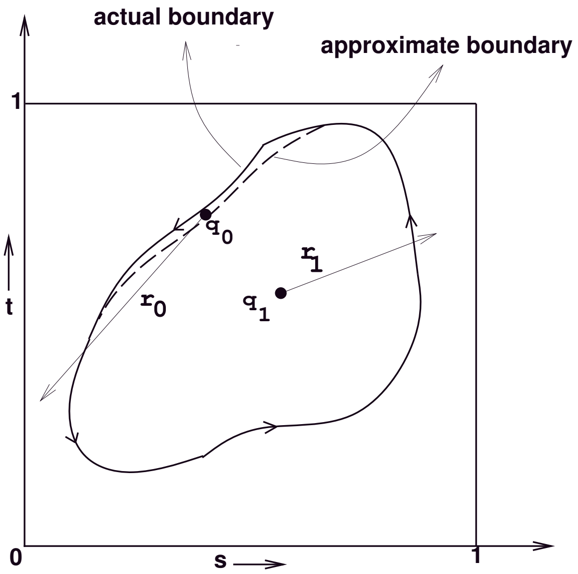

Figure 6: Inaccurate point classification

3.2.2 Component classification

Another part where floating point errors result in failure of the algorithm is during component classification. As we will describe later, we use ray shooting for this purpose and not sign evaluation as done by polyhedral modelers. The entire computation boils down to classifying whether a point lies inside or outside the trimming region. Fig. 5shows an example. In most cases, classifying points like $\mathbf{q_1}$ is not a problem. One ray-shooting query will answer it. However, consider a point like $\mathbf{q_0}$ which lies very close to the boundary. Approximate representations of the trimming boundary further exacerbates this problem. Depending on the choice of ray directions and the tolerances used we may get different classifications. This error could result in topologically inconsistent answers.

There are a number of other similar problems which plague floating point modelers, and resolving these situations is no different from the ones highlighted. We believe that using exact arithmetic and representation is essential for reliable B-rep computation.

3.3 Exact computation of kernel routines

In this section, we will present efficient algorithms to implement the kernel routines in exact arithmetic. In particular, given algebraic curves and points, we present efficient algorithms for comparing two algebraic numbers, evaluating signs of slopes for resolution of regular algebraic curves, and classifying points with respect to a region.

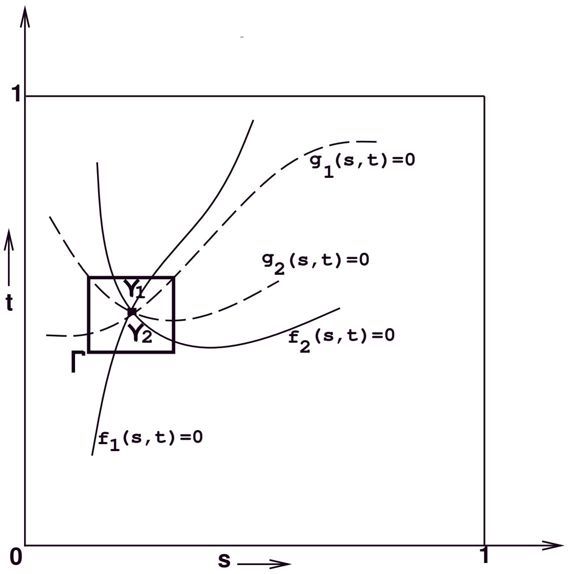

Figure 7: Algebraic number comparison

3.3.1 Comparison between algebraic numbers

It was mentioned earlier in Section 2 that each of the vertices in the solid are defined as roots of a set of polynomial equations with rational coefficients. Since we are dealing with rational parametric surfaces, each of these equations is bivariate. A vertex in the domain, is therefore, the common solution of two equations, $f_1(s,t) = 0$ and $f_2(s,t) = 0$. These are usually irrational numbers, and cannot be represented exactly. In our algorithm, we represent each algebraic number as a small (based on lazy evaluation) rational rectangle. The rational rectangle is guaranteed to isolate each common root of $g_1(s,t)$ and $g_2(s,t)$ (taking into account the multiplicities of roots).

Consider a rational rectangle, Γ, that isolates two algebraic numbers, γ_1 and γ_2 (see Fig. 6). Let γ_1 be the common root of $f_1(s,t)$ and $f_2(s,t)$, and γ_2 the common root of $g_1(s,t)$ and $g_2(s,t)$. There is an exact procedure to answer the question of equality of γ_1 and γ_2. Consider the new polynomial $p(s,t) = f_1^2(s,t) + f_2^2(s,t) + g_1^2(s,t) + g_2^2(s,t)$, and one of the original polynomials (say $f_1(s,t)$). It is easy to show that $p(s,t)$ and $f_1(s,t)$ have a common isolated root in Γ iff $\gamma_1 = \gamma_2$. We check for the common root using Sturm sequences. Further, this method can be extended to arbitrary dimensions.

3.3.2 Derivative sign computation at algebraic points

Fig. 7shows an example of an algebraic curve in a small region. Once the turning points are computed, they are separated using rational grid lines (s_1, s_2, t_1, and t_2 in the figure). In order to determine the topology of the curve (see Section 8.3), we need to compute the sign of the slope of the curve at points where the grid lines intersect the curve (p_4 in figure). Since p_4 is algebraic (a root of a univariate polynomial), we can compute it only within an interval (say, $[t_{41}, t_{42}]$). It is not possible to compute the sign of the slope directly (equivalent to computing signs of partial derivatives) at p_4. Let $f(s,t)$ be the equation of the curve. We know that there is a unique root of $f(s_1,t)$ in the interval $[t_{41}, t_{42}]$. Let $g(t)$ be the partial with respect to s at $s = s_1$. We know that p_4 is not a common root of $f(s_1,t)$ and $g(t)$ (because it is not a turning point). We refine the interval $[t_{41}, t_{42}]$ such that there is no root of $g(t)$ within the interval. The sign of $g(t)$ can be obtained by evaluating it at any rational point in the interval.

3.3.3 Point classification

Classifying a component with respect to a solid amounts to classifying a point with respect to the trimmed domain. Problems associated with point classification using floating point arithmetic were highlighted earlier. We now describe our algorithm to exactly check if a point (with algebraic number coordinates) lies inside or outside the trimming boundary. Initially, we assume that the actual point does not lie exactly on any algebraic curve that is part of the trimming boundary. The point is actually obtained by performing a ray-surface intersection with a random ray. The probability of this ray hitting the boundary curve of a trimmed patch is extremely small. This case is similar to four surfaces intersecting at a point and will be discussed when we handle degeneracies.

Given that the point is an algebraic number, we represent it with a rational rectangle of small dimensions. We must ensure that the rectangle lies in at most one of the bounding boxes of the trimming curve (see Section 8). If it lies in more than one, the rectangle should be further refined so that it only lies in one. Each of the four vertices of the rectangle is classified with respect to the trimming boundary. If all the results are same, actual point classification is done. The more interesting case is when some vertices of the rectangle yield different results. In this case, we refine our rectangle until consistency is achieved. We know this is assured because the point does not lie on any curve. In practice, if after a few levels of refinement we cannot classify the point, we choose a different ray.

4 Exact B-rep Computation Algorithm

In this section, we briefly describe each step of our algorithm. It is implemented on top of the kernel routines.

Obtaining the intersection curve for the two patches: We want to find the intersection curve in the domain of each of the two patches. To find the intersection curve in the domain of patch 1, we substitute the parameterization of patch 1 into the (precomputed) implicit representation of patch 2. A similar procedure obtains the intersection curve in the domain of patch

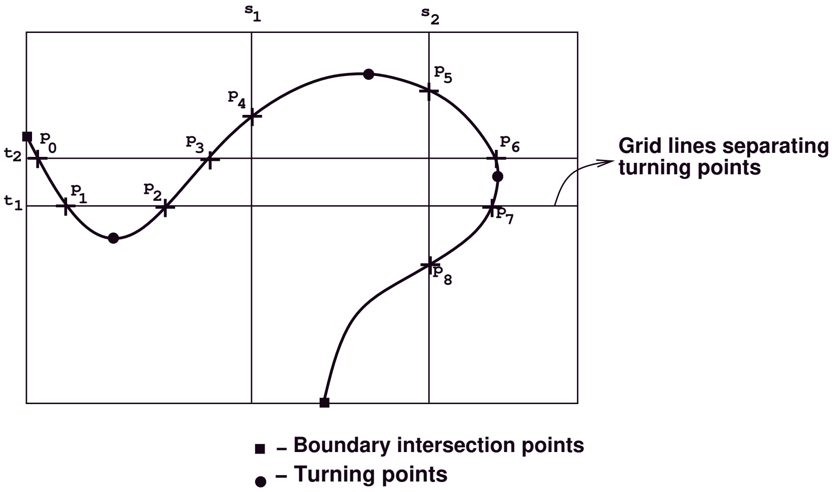

Figure 8: Topological resolution of algebraic curves in finite domain

2.

4.1 Curve-surface intersection

Curve-surface intersection is used to find the points where the intersection curve meets the patch boundary. We want to find the intersection points of the curve and surface in both domains. Finding the intersection points of a curve with the boundaries of its own patch is straightforward - just substitute in the value for s or t and use a univariate Sturm sequence to isolate the roots. The *inversion* problem is to find the corresponding points in the domain of patch 2.

The inversion problem can be solved by considering the patch boundary as a curve in space. Then, we can compute the intersections of this space curve with the other patch (a curve-surface intersection). For example, consider the domain boundary $t = 0$. Then, the space curve corresponding to the patch boundary will be given by $X(s,0), Y(s,0), Z(s,0)$, and $W(s,0)$, where X, Y, Z, W give the parameterization of the first patch. The second patch will have the parameterization $\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)$. Then, to solve for the points of intersection we need to solve for solutions of the equations (5) in their respective domains.

$$\begin{aligned}
\overline{X}(u,v)\ W(s,0) \ &- \ X(s,0)\ \overline{W}(u,v) = 0 \\
\overline{Y}(u,v)\ W(s,0) \ &- \ Y(s,0)\ \overline{W}(u,v) = 0 \\
\overline{Z}(u,v)\ W(s,0) \ &- \ Z(s,0)\ \overline{W}(u,v) = 0
\end{aligned} \tag{4}$$

These equations may be solved using a trivariate Sturm sequence as described in [Mil92]. This will give solutions bounded in u, v, and s. However, the volume function computation involves three successive elimination steps (Sylvester resultant). Depending on the size of the coefficients, this process could be computationally intensive.

A more practical, though less exact, approach to solve this problem is to isolate all the roots in the s domain using univariate Sturm sequence. This is followed eliminating s from equation (5) to produce two independent equations in u and v. This is a bivariate Sturm sequence problem and is solved by a single Sylvester resultant computation. This gives all the solutions in (u, v) space. Determining the correspondence between the (u, v) pairs and s roots is done by comparing each of them in 3-space. We found that in practice, this method was significantly more efficient.

4.2 Pruning intersection curves using trimming boundary

Once the starting points of the intersection curves are determined, we resolve the topological type of the curve using the algorithm in [AF88]. This is briefly discussed in the Appendix. At the end of this method, we have the intersection curve for each patch divided into monotonic segments within the patch domain.

We now have to trim the curve based upon the trimming boundary. Basically, we need to intersect the intersection curves with the trimming curves (represented as an algebraic curve) and throw away the sections of the intersection curve which are outside the trimming boundary.

Finding the points of intersection between the trimming curve and the intersection curve is relatively simple - simply use the bivariate Sturm sequence again. It is also relatively simple to find the corresponding points on the other patch - the trimming curve has a surface associated with it, and this surface intersected with the second patch gives another curve in the domain of that second patch, From this we obtain the intersection points with the intersection curve in the second patch, and figure out which points correspond by matching the intervals in 3-space. The actual pruning step is carried out by determining the orientation (inside/outside) of one point (we choose the starting point on the intersection curve) using 2D ray-shooting. Propagating this information to adjacent sections of the curve clearly identifies the curve segments that lie inside the trimmed region, and which lie outside. Along with this intersection curve segment, we also maintain the patch number of the other solid that formed this curve. We use this later during topology information updation.

4.3 Intersection curve merging within patch domain

In the previous step, we have obtained the intersection curves on each patch for all patch-patch pairs; we need to merge the new trimming curves for that patch. This is done by matching the endpoints of each of the intersection curves, thereby partitioning the patch domain into closed loops. Notice that the monotonic segments of each intersection curve are already merged. Basically, we must check both endpoints of each intersection curve against the endpoints of all of the other intersection curves. If we find that two curves share a common endpoint, then we

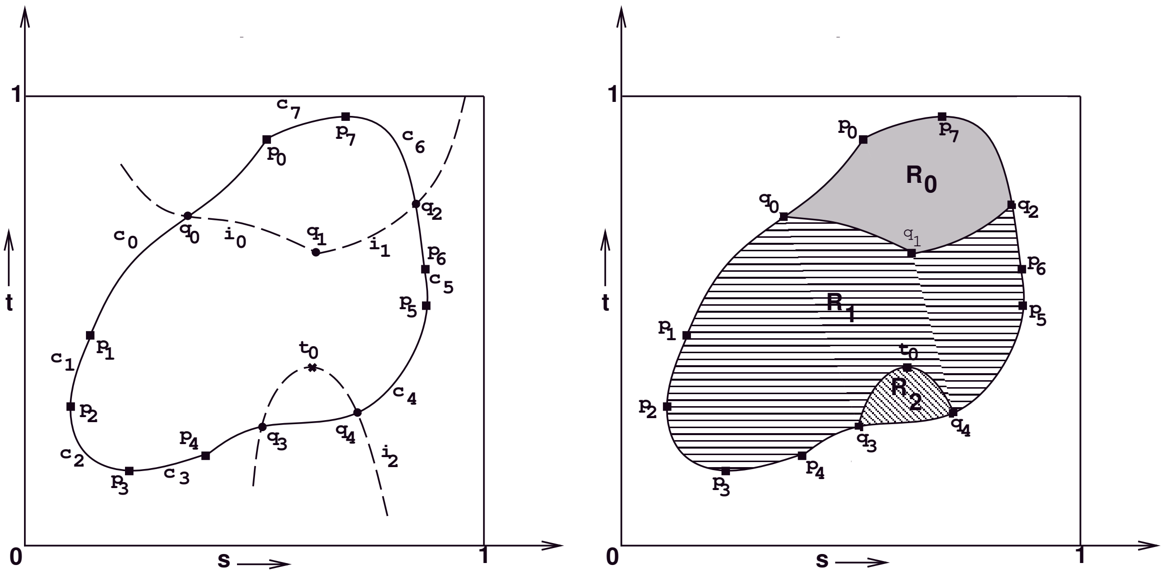

Figure 9: (a) Intersection curves inside trimmed domain (b) Partitions introduced by intersection curves

store this information and consider the curves merged. Point equality is implemented as a kernel routine.

Once the intersection curves have been merged, we need to once again check whether the bounding boxes of the monotonic segments are overlapping and, if so, subdivide the curve appropriately.

4.4 Partitioning Trimming Boundaries

Once all the intersection curves are merged within each patch, they must partition the trimmed domain (if the assumptions that the individual solid boundaries are closed and compact are maintained). Otherwise it is a degenerate intersection (we discuss such cases in Section 6). Fig. 9(a) shows intersection curves inside a trimmed domain. $\mathbf{c_i}$'s (with endpoints $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$) are monotonic curves (in both s and t) that form the trimmed boundary of the patch. $\mathbf{I_0}$, $\mathbf{I_1}$, and $\mathbf{I_2}$ are the intersection curves computed with various patches of the other solid. $\mathbf{t_0}$ is a turning point on the curve $\mathbf{I_2}$. As described earlier, all the turning points are identified before the topology of the algebraic intersection curve can be resolved. $\mathbf{q_i}$'s are points on the intersection curve where the curve intersects the trimmed boundary. Given this information, Fig. 9(b) shows the actual partitions ($\mathbf{R_i}$s). To compute the explicit B-rep of the resulting solid, each of these partitions is generated. We now present an algorithm that computes these partitions provided the intersection curves have no singularity in the trimmed domain.

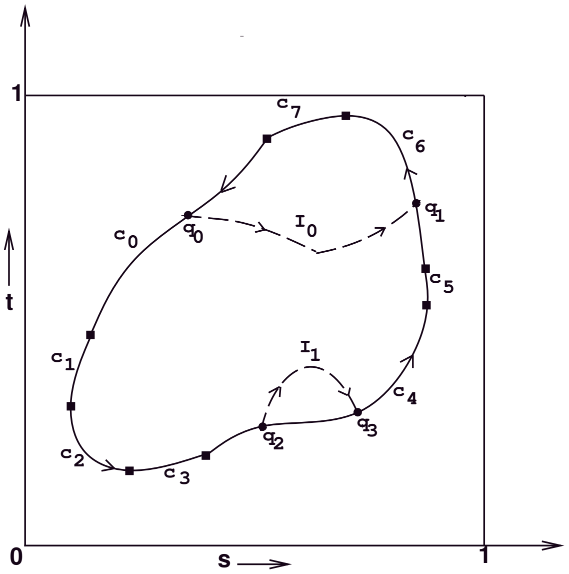

Figure 10: Partitioning a trimmed patch with chains of algebraic curves

The main idea in this algorithm is the fact that since the intersection curve segments (I_0 and I_1 in Fig. 10) are non-intersecting, each resulting partition starts at one endpoint of a curve segment, and ends at the other endpoint of the same curve segment. We shall assume that the trimming curves and the intersection curves are given in a specific order. We number the endpoints of the intersection curve segments such that $\mathbf{q_{2j}}$ and $\mathbf{q_{2j+1}}$ belong to $\mathbf{I_j}$. The algorithm works in three steps.

- Each endpoint of a curve segment (for example, $\mathbf{q_0}$ of $\mathbf{I_0}$) lies on a unique curve (except when it coincides with one of the curve endpoints of the boundary) of the trimming boundary. In fact, points like $\mathbf{q_0}$ are determined as the intersection of $\mathbf{I_0}$ with $\mathbf{c_0}$. Note that even though $\mathbf{c_0}$ and $\mathbf{c_1}$ could be part of the same algebraic curve, the association of $\mathbf{q_0}$ with $\mathbf{c_0}$ is determined because of the monotonicity of the $\mathbf{c_i}$'s. Each boundary curve $\mathbf{c_i}$ is then partitioned into multiple segments depending on the number $\mathbf{q_j}$'s lying on it.

- This is followed by a traversal of the trimming boundary in a consistent order by maintaining a stack. Two types of elements are pushed in the stack - curve segments, and curve endpoints. Initially, we keep pushing in the boundary curve segments until we reach a vertex like $\mathbf{q_0}$. Let the vertex number be k. If the topmost curve endpoint type of the stack (say, $\mathbf{l}$) has a number $(k+1)$ or $(k-1)$, then a partition has to be read out. Otherwise, vertex k is pushed into the stack followed by all the curves that comprise $I_{\lfloor k/2 \rfloor}$. If a decision to read out a region has been reached, all the curve segments until vertex $\mathbf{l}$ are popped. Curves comprising $I_{\lfloor k/2 \rfloor}$ are pushed again because they are required by the next region too. The order in which these curve segments are pushed into the stack has to be monitored carefully so that a region which is read out is oriented consistently.

- Till now, we have considered only intersection curve segments whose endpoints lie on the trimming boundary. However, there may be loops that lie completely inside the boundary. Any loop is present (if at all) inside one of the obtained partitions. Each of the loops (starting from the innermost if the loops are nested) themselves form a partition. The remaining part of the region (it has boundaries with multiple components) is broken into simple regions by introducing a simple cut from the loop to the boundary of the partition or the next loop.

This completes the algorithm to compute the partitions introduced by intersection curves. A feature of this algorithm is that the adjacency structure between the various partitions (which is necessary to avoid redundant, expensive ray-shooting queries during component classification) are obtained by the order in which they are read out.

4.5 Topological Information Updation

It is clear from the previous section that intersection computation introduces new vertices, edges, and faces in the solid. This change needs to be incorporated in our topological structure. Further, information about the adjacency between the various faces significantly reduces the component classification time. At this time, we just concentrate on the face adjacency. Vertex and edge adjacency are updated during final solid generation.

The new graph is a refinement of the original adjacency graph. Each vertex in the original graph is split into a few vertices depending on the partitions obtained due to the intersection curves. We need to figure out the adjacency relationship between the newly created vertices. Consider, for example, that vertices $\mathbf{u}$ and $\mathbf{v}$ were adjacent in the original graph. Due to the intersection curves, let the vertex $\mathbf{u}$ be split into $\mathbf{u_1}$, $\mathbf{u_2}$, …, $\mathbf{u_m}$, and let the vertex $\mathbf{v}$ be split into $\mathbf{v_1}$, $\mathbf{v_2}$, …, $\mathbf{v_n}$. The adjacency between the various $\mathbf{u_i}$'s (similarly $\mathbf{v_i}$'s) has already been determined (during partitioning). These adjacencies (let us call it set S) are purposely left out in the new graph. Let e be the edge along which $\mathbf{u}$ and $\mathbf{v}$ were adjacent in the original graph, and let it be divided into k portions during partitioning. Then all the adjacencies between $\mathbf{u_i}$'s and $\mathbf{v_j}$'s can be obtained in $O(k)$ time. The number of connected components in this graph gives the number of solid components introduced by the intersection curves. Let the solid components be named $CC_0, CC_1, \ldots$. Note that each CC_i has a collection of faces.

To obtain the connectivity between the various CC_is, we introduce some notation. Let R be a mapping which takes a vertex in the new graph to the corresponding vertex in the original graph. For example, if $\mathbf{u}$ was split into $\mathbf{u_1}$, $\mathbf{u_2}$, …, $\mathbf{u_m}$, then $R(\mathbf{u_i}) = \mathbf{u}$. Two components CC_i

and CC_j are connected if

$$\{\exists \mathbf{u_1} \in CC_i, \exists \mathbf{u_2} \in CC_j | R(\mathbf{u_1}) = R(\mathbf{u_2}) \ and \ (\mathbf{u_1}, \mathbf{u_2}) \in S\}$$

Using this, we have obtained the various components and their connectivity of a solid due to intersection curves. Next we resolve each of these components (inside/outside) with respect to the other solid.

4.6 Component Classification

Component classification involves determining whether a given component of one solid is inside or outside the other solid. It is obvious that the entire component (as obtained in the previous section) lies completely inside or outside the other solid. In most polyhedral modelers, component classification is carried out locally [Hof89]. When dealing with sculptured surfaces, though, the same technique cannot be used. The most general method is based on ray-shooting. Ray-shooting is done by firing a semi-infinite ray in an arbitrary direction and checking for intersections with the other solid. If the number of intersections is even, the point (and hence, the entire component) lies outside the solid, if it is odd, it lies inside.

There are three steps involved in our algorithm to perform component classification. The first step involves getting a point that is part of the component. This reduces to finding a point inside the trimming boundary of one patch. This is accomplished by 2D ray-shooting. We initially choose some rational point $p = (s, t)$ in the domain such that t lies between the lower and upper extents of the trimming boundary. A horizontal line passing through p (in both directions) is intersected with the boundary, and all the intersections are determined using the root isolation method of Milne [Mil92]. The number of intersections must be even and are of the form $(s_1, t), (s_2, t), \ldots, (s_{2n}, t)$. The s_i's are algebraic numbers and are represented as small rational intervals. Choosing the midpoint of s_{2i-1} and s_{2i} for $i = 1, 2, \ldots, n$ gives one rational point inside the trimming boundary. Let this point be called $\mathbf{q}$.

The second step involves actual ray-shooting in 3-space. To perform 3D ray-shooting, we raise this point to 3-space using the patch parametrization. Let this point be (x_q, y_q, z_q). This point is rational because of the nature of the patch parametrization. We pick a random direction and fire a semi-infinite ray in that direction. We compute all the intersections of each patch of the other solid with this ray. This is done similar to the curve-surface intersection computation described earlier in this section. However, not all the intersection points computed this way lie inside the trimmed boundary of the patch. Checking if the intersection point lies inside the trimmed boundary forms the third step of our algorithm.

This step is performed similar to the first step. Given a small rational rectangle isolating the actual root, we fire a semi-infinite ray in a random direction and find all the intersections with the trimming boundary. If the number of intersections is odd, the intersection point is part of the trimmed patch, otherwise it is not. We count the number of intersections that lie inside the trimmed patches that make the solid. If that is odd, the component is inside the other solid.

4.7 Final B-rep Generation

The trimmed patches that make up the final solid are determined by the Boolean operation performed. Given two solids $solid_1$ and $solid_2$, we decide on the final B-rep depending on the Boolean operation as follows:

- *Union:* All components of $solid_1$ that lie **outside** $solid_2$, and vice-versa are retained.
- *Intersection:* All components of $solid_1$ that lie **inside** $solid_2$, and vice-versa are retained.
- *Difference:* All components of $solid_1$ that lie **outside** $solid_2$, and all components of $solid_2$ that lie **inside** $solid_1$ are retained.

We also update the topology information. Each connected component that is retained in the final solid has some graph vertices (faces of the solid) whose complete adjacency is not determined. These correspond to edges which are formed by intersection curves. The vertex that should be adjacent to this vertex along these edge in the final graph is part of the other solid, and is the surface that formed the intersection curve. We maintain the patch number of the other solid that resulted in an intersection curve, and use it to complete the adjacency. From this graph, the entire topological information is easily computable.

5 Analysis and Performance

In this section, we shall discuss some of the theoretical and empirical complexity analysis for some important steps of our algorithm. It is clear that the most dominating steps in terms of time are the root isolation of bivariate polynomials, and the topological resolution of intersection curves. We optimize these algorithms, and implement them as part of the kernel routines. The complexity of steps involving partitioning based on the intersection curve, and the face connectivity generation are very small compared to the total cost, and their analysis is omitted here. We now give the theoretical worst-case complexity of the root isolation algorithm.

5.1 Worst case analysis

Root isolation: Most of the results involving real root isolation are based on Sturm sequences, and we quote a result from Davenport [Dav85] for the worst-case time complexity of root isolation algorithm for univariate polynomials.

Theorem 1 *[Dav85] The running time of the root isolation algorithm based on Sturm sequences of univariate polynomials is bounded by $O(n^6(\log n + \log \Sigma a_i^2)^3)$, where n is the degree of the polynomial and a_i are its coefficients.*

But this bound is too pessimistic, and a result based on [Hei71] predicts the average case to be more like $O(n^4)$. We shall now look at the growth of the coefficient size while computing Sylvester resultant of two polynomials. Let

$$f^n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \tag{5}$$

and

$$g^m(x) = b_m x^m + b_{m-1}x^{m-1} + \ldots + b_1 x + b_0 \tag{6}$$

be the two polynomials whose resultant we are computing. In our case, this resultant is used to compute the volume function, and hence a_is and b_js are symbolic coefficients (a polynomial in u, s, and t as given in Section 8). The resultant of these two polynomials is a $(m + n) \times (m + n)$ matrix given by (3). Let

$$c = \max\{\max_{i=0}^{n}(wt(a_i)), \max_{j=0}^{m}(wt(|b_j|))\}$$

be the maximum coefficient size of the polynomials, where $wt(a_i)$ is the sum of the absolute integral coefficients of a_i.

The resultant (which is the determinant of the matrix in (3)) has maximum weight bounded by W_{m+n}, where W_{m+n} is given by the recurrence

$$\begin{aligned} W_{m+n} &= (m + n)\ c\ W_{m+n-1} \\ &\leq [(m + n)\ c]^{m+n} \end{aligned}$$

For the case of quadrics, tori, and other low-degree solids that we deal with m and n equals 4. Thus the bit complexity of the coefficients of the univariate polynomial is roughly 8 times the original bit complexity in the worst case.

Topological resolution of intersection curves: The algorithm described [AF88] computes the sign invariant decomposition of an algebraic curve in a finite domain. The time complexity is given by the following result:

Theorem 2 *[AF88] Given a bivariate polynomial of total degree n and coefficient size d in E^2, one can obtain a sign invariant decomposition of the curve in time $O(n^{12}\ (d + \log n)^2\ \log n)$.*

5.2 Performance improvement of exact arithmetic

As exhibited by the worst case bounds, the arithmetic of these symbolic coefficients is expensive. Actually, the cost of arithmetic operations is quadratic in the size of its coefficients. Even though, in reality, we do not experience such a drastic increase in bit complexity for intermediate computation, it nevertheless grows. To reduce the cost of arithmetic operations, we perform all our computations over finite fields, and use a probabilistic algorithm based on the Chinese Remainder Theorem to recover the actual coefficients [MC93]. The time complexity of the resultant computation (using interpolation algorithm in [MC93]) is directly proportional to the number of primes used in the finite field computation. To reduce this number, we use primes of maximum possible magnitude. Most current implementations of *bignum* libraries use finite fields of order 2^{16} to prevent overflow when taking products. Most of the current machines provide multiplication instructions that give the result out in two registers. Taking advantage of this fact, we use an assembly level subroutine that performs multiplication. This allows us to use finite fields of order as high as 2^{31}. Compared to finite fields of order 2^{16}, we get a significant speedup in this process.

It seems possible to use the mantissa part of floating point numbers (53 bits) to further increase the order of finite fields. However, this needs more investigation before we can apply it to our approach.

Lazy evaluation during root isolation: Another optimization that we perform to improve our speedup is to evaluate rational intervals (in root isolation) as lazily as possible. This is based on the assumption that the worst case bounds that govern the closeness of roots actually occur very rarely in practice. Thus, most of the time, we are able to isolate the roots of polynomials inside the domain of surfaces quickly. However, during later computation of other roots, it might be necessary to make sure that two roots are not the same (because of large overlapping intervals). At this time, we refine the computed intervals further and try to isolate their intervals. This optimistic approach to root isolation behaves almost like an output sensitive algorithm.

5.3 Empirical Performance

It is quite clear that the running time of our algorithm grows quickly as a function of the degree of the solids. We have tried computing B-reps for low degree solids like quadrics and tori. Based on our experience for these solids, *the use of exact arithmetic slows us down by slightly more than one order of magnitude* (as compared to B-rep algorithms based on finite precision arithmetic). The performance varies with the degrees, relative orientation of two solids and the complexity of the output. In cases when the solids are close to being degenerate, the coefficient sizes grows more quickly, and the algorithm slows down. However, we feel that this penalty is justified for improving the robustness of the modeler and performing reliable computation. Our algorithm for root isolation runs on three different platforms - Suns, HPs and SGIs. Most

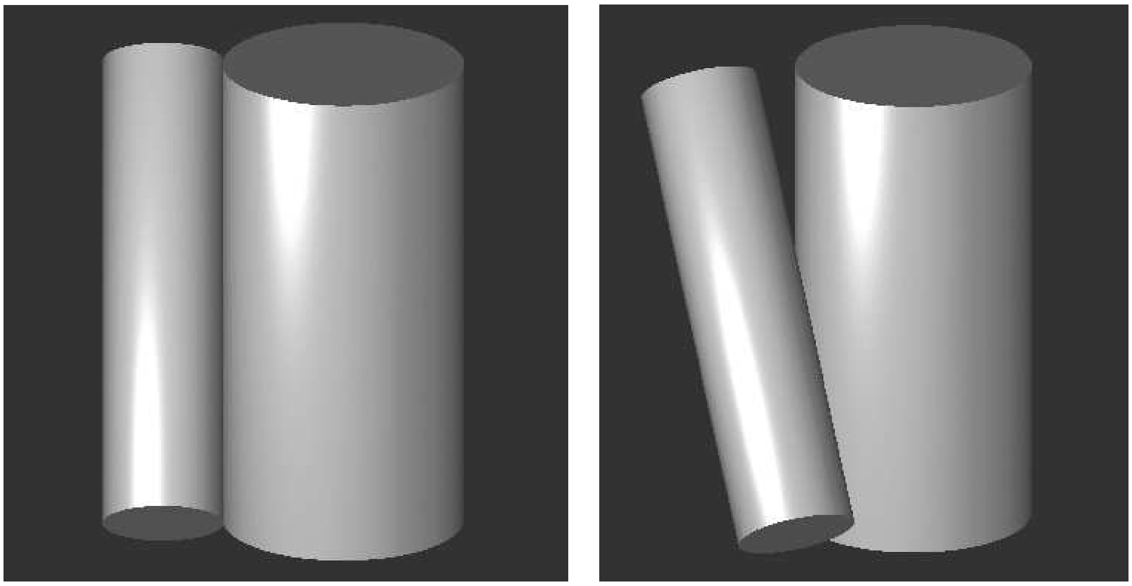

Figure 11: Two views of just interpenetrating cylinders

of our timing measurements and estimates are on an HP 712/100 model workstation with 32 MB of main memory. We would like to mention that though we have implemented various steps of the algorithm independently, they have not been integrated together. Furthermore, our implementations have not been optimized (specially the kernel routines).

For most quadrics, the parameterization is biquadric (leading term s^2t^2). Substituting this into the implicit form of the other patch results in an intersection curve which is biquartic (s^4t^4) in the domain of the patch. While computing the resultants of two such equations for root isolation, the order of the resultant matrix is at most 8×8. For these orders of matrices, the resultant algorithm computes it in about 50 milliseconds. The algorithm uses the special structure of the Sylvester matrix, and evaluates the resultant using interpolation methods. The volume function (see Section 8.2), thus computed, is a univariate polynomial in u (actually it is a trivariate polynomial in u, s , and t, but we treat s and t as constants) of maximum degree 8. Depending on the size of the actual coefficients, the time taken to compute the Sturm sequence ranges from 0.2 to 1 second. The entire root isolation procedure for these cases typically takes less than a second to 2 seconds.

5.4 Example

In this section, we illustrate some key parts of our approach using an example of two cylinders which are just interpenetrating (see Fig. 10). The cylinders are of radius 1 and 0.5, and their centers are spaced 1.49 units apart. The cylinders are rotated with respect to each other. We divide the surface of each cylinder into four equal parts and represent each of them as a rational parametric surface with rational coefficients. The parametric form of a sample patch from each cylinder are given below.

$$\begin{array}{ll} X(s,t) = 1 - s^2 & \overline{X}(u,v) = \frac{199}{100}\, u^2 + \frac{99}{100} \\ Y(s,t) = 2\, s & \overline{Y}(u,v) = \frac{4}{5}\, u - \frac{6}{5}\, v - \frac{6}{5}\, u^2\, v + \frac{3}{5} \\ Z(s,t) = 2\, t + 2\, s^2\, t - 1 & \overline{Z}(u,v) = \frac{3}{5}\, u + \frac{8}{5}\, v + \frac{8}{5}\, u^2\, v - \frac{4}{5} \\ W(s,t) = 1 + s^2 & \overline{W}(u,v) = 1 + u^2 \end{array}$$

After implicitizing using Dixon's formulation, the implicit forms are

$$\begin{aligned} f(x,y,z,w) &= w^2 - x^2 - y^2 \\ g(x,y,z,w) &= 19701\, w^2 - 29800\, x\, w + 10000\, x^2 + 6400\, y^2 + 9600\, y\, z + 3600\, z^2 \end{aligned}$$

To obtain the intersection curve of the two patches in the domain of the first patch, we substitute its parameterization into the implicit form of the second patch ($g(x,y,z,w) = 0$).

$$\begin{aligned} h(s,t) = {} & -3501 + 19200\, s - 45002\, s^2 - 59501\, s^4 + 14400\, t - 38400\, s\, t + \\ & 14400\, s^2\, t - 38400\, s^3\, t - 14400\, t^2 - 28800\, s^2\, t^2 - 14400\, s^4\, t^2 = 0 \end{aligned}$$

Since the patches are untrimmed, we have to compute the starting points of the curve on the boundary of the patch. Substituting $s = 0$ into h and computing the volume function for this univariate case, we get

$$V(u,t) = -3501 + 14400\, t - 14400\, t^2 + 14400\, u - 28800\, t\, u - 14400\, u^2$$

We computed the Sturm sequence of this volume function, and isolated the roots of the original equation between $[0,1]$ to within a precision of $\frac{1}{100}$. The two roots were

$$\left[\frac{226834}{390625}, \frac{1838}{3125}\right], \quad \left[\frac{1298}{3125}, \frac{6634}{15625}\right]$$

These numbers give the ranges in t for which there is an intersection of the intersection curve with the s=0 edge of the patch domain. Of these, only the first one corresponds to a point inside the domain of the second patch. This was obtained by inversion and a bivariate Sturm sequence generation. The point corresponding to $\left(0, \left[\frac{226834}{390625}, \frac{1838}{3125}\right]\right)$ inside the domain of the second patch is $\left(\left[\frac{7352}{78125}, \frac{64}{625}\right], \left[\frac{43682}{78125}, \frac{8866}{15625}\right]\right)$. The s (t) turning points on the intersection curve were obtained by performing bivariate Sturm sequence root isolation on the pairs of polynomials $h(s,t)$ and $h_s(s,t)$ ($h_t(s,t)$). The s and t turning points were found to be $\left(\left[\frac{20464}{390625}, \frac{4352}{78125}\right], \left[\frac{5764}{15625}, \frac{146044}{390625}\right]\right)$ and $\left(\left[\frac{1096}{15625}, \frac{248}{3125}\right], \left[\frac{2}{5}, \frac{6346}{15625}\right]\right)$ respectively.

Now that we have the turning points of the intersection curve, we perform the topological resolution. After separating the turning points and evaluating the signs of slope at various

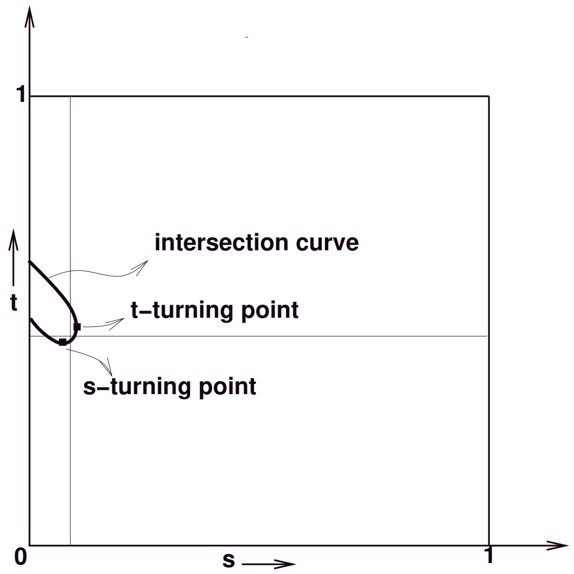

Figure 12: Complete intersection curve in the domain of first patch

grid lines, the connectivity of the curve is obtained unambiguously. The resulting curve in the domain of the first patch is shown in Fig. 11. The intersections of the curve with the s=0 axis, the turning points in s and t, and the grid lines for curve connectivity computation are all shown in the picture. Note that the curve shown was obtained by intersecting this patch with *all* the patches of the second cylinder.

6 Degeneracies

A number of degenerate cases can arise when dealing with curved surfaces. Some of these degeneracies are of the same general type as is found in a polyhedral modeler, while some others arise only with curved surface modelers.

One type of degeneracy which occurs only in the curved surface domain is when two surfaces meet at a point. Such a point will give an intersection curve in the patch domain which is singular. Since we have assumed that we do not have singular curves, we assume that such a case will not occur. If, in fact, such a case does occur, we will find it by noticing that the intersection curve has an s turning point and a t turning point at the same position, which tells us there is an error.

Alternately, the two surfaces could meet along a curve, with neither surface actually in the interior of the other (the surfaces are just tangent along that curve). In this case, we will be able to detect that there is a problem when we try to generate the adjacency graph (we will find that two components which should be adjacent are actually part of the same component).

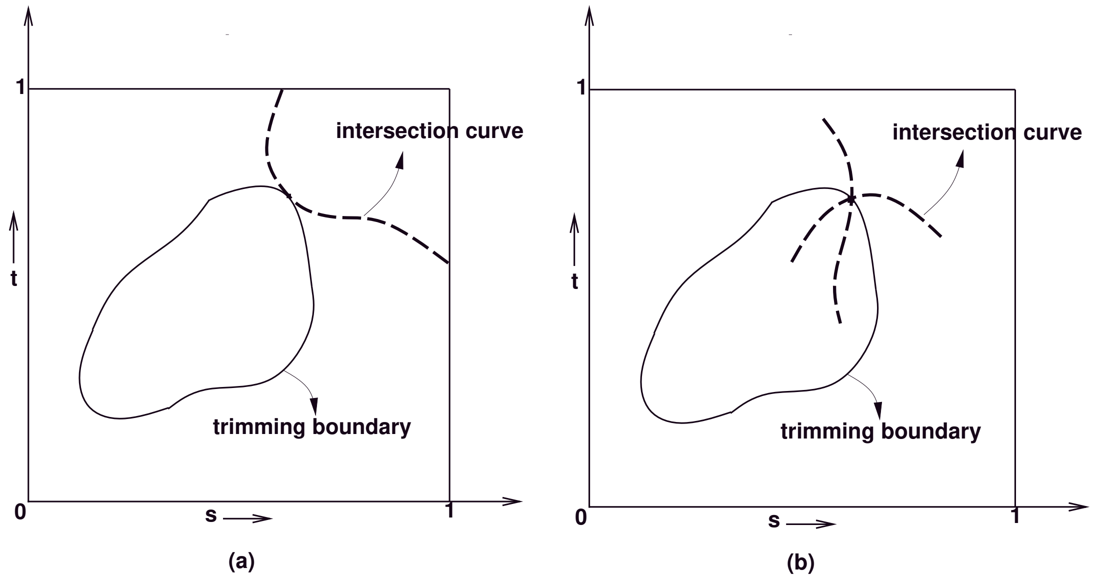

Figure 13: (a) Surface-edge contact degeneracy (b) Four surfaces meeting at a point

Two overlapping surfaces is another degenerate problem we may encounter, however this problem corresponds to a case in the polyhedral domain (face-face overlap). Note that if the surfaces we use have an irreducible implicit form, then those two surfaces will not overlap unless they are identical.

A second degeneracy which has a corresponding case for the polyhedral domain is that of a surface just touching an edge (edge-face contact in polyhedral domain). In our representation, this will appear as an intersection curve which is tangent to a trimming curve (see Fig. 12(a)).Such a case can be automatically eliminated if we check *each* component of the intersection curve to see whether it is in the trimmed region. Notice that checking each component does not allow us to use the speed-up mentioned earlier of propagating the information about one component of the intersection curve to all other components of that curve (which will not detect and remove this case).

There is another type of degeneracy which is analogous to a case in the polyhedral domain. This degeneracy can be viewed as four surfaces intersecting at a point. Examples of this include when a vertex of one solid lies on the surface of another solid, or when the edges of two solids meet. Obviously, the vertex can be thought of as the intersection of three surfaces, and the edges can be thought of as the intersection of two surfaces, thus the cases mentioned would involve the intersection of four surfaces.

Even more degenerate cases, such as two vertices meeting, or a vertex lying on an edge, are possible, but these can be viewed as 5 or 6 surfaces meeting at a point - i.e. at least four surfaces are still meeting at a point.

These cases will manifest themselves in our modeler as three (or more) curves meeting at a common point in the domain of some patch (see Fig. 12(b)). Assume these three curves are $f1$, $f2$, and $f3$. We can find out whether this case has occurred by checking equality of the intersection of $f1$ and $f2$ in some interval with the intersection of $f1$ and $f3$ (or $f2$ and $f3$) in that same interval.

Degeneracies in the polyhedral case can generally be classified into the category of four planes meeting at a point. In the polyhedral domain, a basic geometric predicate exists (plane orientation) which can be used to make the decisions which can be affected by degeneracies. The plane orientation predicate involves finding the sign of a 4x4 matrix whose entries are the coefficients of the four planes. [For95] has shown that symbolic perturbation of the plane equations, coupled with some high-level algorithmic properties, can be used to eliminate the four-planes-meeting-at-a-point degeneracies.

We would like to extend this approach to the curved-surface domain (four planes meeting at a point becomes four surfaces meeting at a point). This would allow us to handle these degeneracies smoothly without having to separately detect and handle them. In order to do so, there are several key elements of Fortune's approach that must be incorporated. These are:

- Ensure that 3 surfaces meeting at a point is satisfied by high-level algorithmic properties.
- Have a basic geometric predicate which can be used to determine how four surfaces are arranged
- Have a way of perturbing surfaces symbolically both "outward" and "inward"

Ensuring that three surfaces meet at a point is relatively simple. The other elements are much more difficult to obtain in the curved surface domain.

As far as a geometric predicate for the curved surface domain, the closest analogy to the plane orientation matrix would be the resultant of the implicit form of four surfaces. The resultant will be zero if and only if there is a common solution to all four equations - i.e. a point lying on all four surfaces. If we limit ourselves to low-degree solids, this computation involves the computation of the sign of the determinant of a matrix with at most a few hundred rows and columns. Whereas the sign of the plane orientation matrix determinant gives which side of a plane a single point lies on, the sign of the resultant will give information about a set of points. Three surfaces will, in general, intersect in a number of points. *Each* of these points will have a sign relative to which side of the fourth surface they lie on. The product of these signs is the sign of the determinant. In order to be able to use this resultant, we need to find a way to bound the surfaces such that the resultant computation only gave information about one of the roots.

It is relatively easy to come up with a scheme for perturbing the implicit equations defining the surfaces. The problem comes with defining how to perturb "inward" and "outward." Offsetting a curved surface is much too complicated to be done with a simple perturbation of one (or a few) coefficients, and a simple translation of a surface might not maintain the structure of the object. Furthermore, it should be remembered that for any perturbation we make, we must be able to calculate with it efficiently in the resultant computation (hopefully separating it from the resultant matrix similar to the way Fortune separates the perturbation from the plane orientation matrix).

7 Extensions and Future Work

In this paper, we have presented representations and algorithms to compute B-reps for boolean combinations of low-degree solids specified with rational parametric surfaces. We use exact arithmetic to perform reliable computations on a number of kernel routines, upon which the rest of the modeler is built. The efficient and accurate implementation of the kernel routines allows us to have an efficient and reliable method for the overall B-rep computation.

There are a number of ways in which the method we have described might be extended. We briefly describe, here, a few areas that might be looked at for future work.

Non-Parametric/Algebraic Surfaces: Currently our surfaces must be defined as rational parametric surfaces. Although this is more general than most previous modelers, which use only planar surfaces, it would be useful to see whether parts of our approach can be extended to deal with non-parametric surfaces or non-algebraic surfaces. Such an extension would involve fundamental changes to the representation (e.g. trimming curves cannot be expressed in the "patch domain" for a non-parametric surface), but it might be that our algorithmic approaches would have straightforward modifications.

Singularities: In our approach, we assume that we do not have singularities in the surfaces or curves. Nothing is done to check for or prevent singularities, and our algorithms can fail in their presence. In order to have a robust modeling system, it will be necessary to handle these situations in some manner. Perturbations or having some extra initial information might help in dealing with this problem.

Non-manifold Cases: We deal only with manifold cases. Others have described methods for handling non-manifold representations and operations ([Wei85]), and we might be able to apply a similar approach in our system.

Handling Degeneracies using Perturbations: It has been shown that perturbation methods can be effective in removing degeneracies ([For95, EM90, GM95]). Perturbations for curved surfaces, however, has not been investigated to any significant degree. Often, perturbation methods will require a symbolic or exact arithmetic as a foundation for their use. Our use of exact representations and exact arithmetic should be helpful in finding a perturbation method which is helpful in the curved surface domain. In Section 6, we have briefly discussed some of the issues which can be considered when trying to develop a perturbation method.

High Degree Surfaces: Our current method seems to work acceptably well for lower degree surfaces (e.g. quadrics, tori). Although this is somewhat of an improvement over modelers which use only planar surfaces, it still limits the surfaces we can represent. Our algorithm can be applied to these higher degree surfaces, however such an implementation would probably be too complex to deal with in a reasonable amount of time. With higher degree surfaces, computations such as finding a resultant will become dramatically more time-consuming. Finding ways of computing efficiently with high degree solids would certainly help to extend our current algorithm.

Using Floating Point for Speed-up: Our current approach uses exact arithmetic to perform all calculations. It has been shown in [BMP94] that the use of a combination of exact and floating point arithmetic can be much faster than exact rational computation, and only slightly slower than floating point computation, for a polyhedral modeler. Exact arithmetic is used only when floating point computations can not guarantee a correct result. Adapting such a scheme to our approach could result in a significant speed-up.

Parallel Implementations: Experience has shown us that parallel implementations of B-rep generators can be significantly faster than sequential versions. It would be interesting to look into how easy it would be to parallelize our current algorithm, and how much speed up we could expect to see as a result.

References

[AB88] S.S. Abhyankar and C. Bajaj. Automatic parametrizations of rational curves and surfaces iii: Algebraic plane curves. *Computer Aided Geometric Design*, 5:309–321, 1988.

[AF88] S. Arnborg and H. Feng. Algebraic decomposition of regular curves. *Journal of Symbolic Computation*, 5:131–140, 1988.

[BMP94] M. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation. *Computer-Aided Design*, 26(6), 1994.

[Bra75] I. Braid. The synthesis of solid bounded by many faces. *Comm. ACM*, 18:209–216, 1975.

[Can88] J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.

[CB89] M. S. Casale and J. E. Bobrow. A set operation algorithm for sculptured solids modeled with trimmed patches. *Computer Aided Geometric Design*, 6:235–247, 1989.

[CK83] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17:289–298, 1983.

[Dav85] J. H. Davenport. Computer algebra for cylindrical algebraic decomposition. Technical Report TRITA-NA-8511, NADA, KTH, Stockholm, 1985.

[Dix08] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[ea95] M. Higashi et al. Face-based data structure and its application to robust geometric modeling. *Proceedings of ACM Solid Modeling*, pages 235–246, 1995.

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[FBZ93] S. Fang, B. Bruderlin, and X. Zhu. Robustness in solid modeling: a tolerance-based intuitionistic approach. *Computer-Aided Design*, 25(9):567–576, 1993.

[FNO89] R.T. Farouki, C.A. Neff, and M. O'Connor. Automatic parsing of degenerate quadric-surface intersections. *ACM Transactions on Graphics*, 8:174–203, 1989.

[For95] S. Fortune. Polyhedral modeling with exact arithmetic. *Proceedings of ACM Solid Modeling*, pages 225–234, 1995.

[For96] S. Fortune. Robustness issues in solid modeling. In M.C. Lin and D. Manocha, editors, *Applied Computational Geometry*, pages 9–14. Springer-Verlag, 1996.

[GM95] Leonidas Guibas and David Marimont. Rounding arrangements dynamically. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 190–199, 1995.

[Hei71] L. E. Heindel. Integer arithmetic algorithm for polynomial real zero determination. *Journal of ACM*, 18(4):535–548, 1971.

[Her80] C. Hermite. Sur l'extension du théoréme de m.sturm a un systéme d'équations simultanées. *tomme III, Mémoire inédit*, 1880.

[HHK89] C. Hoffmann, J. Hopcroft, and M. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, 1989.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.

[Hof96] C.M. Hoffmann. How solid is solid modeling. In M.C. Lin and D. Manocha, editors, *Applied Computational Geometry*, pages 1–8. Springer-Verlag, 1996.

[Jac95] D. Jackson. Boundary representation modeling with local tolerances. *Proceedings of ACM Solid Modeling*, pages 247–253, 1995.

[Joh87] J.K. Johnstone. *The Sorting of points along an algebraic curve.* PhD thesis, Cornell University, Department of Computer Science, 1987.

[KM94] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. Technical Report TR94-062, Department of Computer Science, University of North Carolina, 1994. To appear in ACM Trans. on Computer Graphics.

[KM96] S. Krishnan and D. Manocha. Efficient representations and techniques for computing b-rep's of csg models with nurbs primitives. In *Proceedings of CSG'96*, pages 101–122. Information Geometers Ltd, 1996.

[Mac02] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, 1(33):3–27, May 1902.

[Man88] M. Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Maryland, 1988.

[MB91] M.J. Muuss and L. A. Butler. Combinatorial solid geometry, boundary representations and non-manifold geometry. In D. Rogers and R. Earnshaw, editors, *Advanced Computer Graphics Techniques.* Springer-Verlag, 1991.

[MC93] D. Manocha and J.F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.

[MDA+88] M. Muuss, P. Dykstra, K. Applin, G. Moss, P. Stay, and C. Kennedy. Ballistic research laboratory cad package, release 3.0 - a solid modeling system and ray tracing benchmark. Technical report, BRL Internal Publication, October 1988.

[Men92] J. Menon. *Constructive Shell Representations for Free-form Surfaces and Solids.* PhD thesis, Dept. of Computer Science, Cornell University, 1992.

[MG91] J. Miller and R. Goldman. Combining algebraic rigor with geometric robustness for the detection and calculation of conic sections in the intersection of two quadric surfaces. *Proceedings of ACM Solid Modeling*, pages 221–233, 1991.

[Mil88] V. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Rept. CS88-168, Carnegie-Mellon University, Department of Computer Science, 1988.

[Mil92] P. S. Milne. On the solutions of a set of polynomial equations. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102, 1992.

[RC95] A. Rege and J. Canny. Fast point location for two and three dimensional algebraic geometry. To appear, 1995.

[RR92] A.A.G. Requicha and J.R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, pages 31–44, September 1992.

[RV82] A.A.G. Requicha and H.B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, March 1982.

[RV85] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling: boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1), 1985.

[RV89] J. Rossignac and H.B. Voelcker. Active zones in csg for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithm. *ACM Transactions on Graphics*, 8(1):51–87, 1989.

[Sal85] G. Salmon. *Lessons Introductory to the Modern Higher Algebra.* G.E. Stechert & Co., New York, 1885.

[Sar83] R F Sarraga. Algebraic methods for intersection. *Computer Vision, Graphics and Image Processing*, 22:222–238, 1983.

[Sed83] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces.* PhD thesis, Purdue University, 1983.

[Seg90] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. In *Proceedings of ACM Siggraph*, pages 105–114, 1990.

[Sha91] V. Shapiro. *Representations of Semi-Algebraic Sets in Finite Algebras Gener ated by Space Decompositions.* PhD thesis, 1991.

[SI89] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistencis. *J. Inf. Proc., Inf. Proc. Soc. of Japan*, 12(4):380–393, 1989.

[SJ91] C. Shene and J. Johnstone. On the planar intersection of natural quadrics. *Proceedings of ACM Solid Modeling*, pages 234–244, 1991.

[Wei85] Kevin J. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[Yap90] C. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *Journal of Symbolic Computation*, 40:2–18, 1990.

[Yu92] J. Yu. *Exact arithmetic solid modeling.* PhD thesis, Purdue University, 1992.