Efficient and Accurate B-rep Generation of Low Degree Sculptured Solids Using Exact Arithmetic:I - Representations

John Keyser [a], Shankar Krishnan [b], and Dinesh Manocha [a]

[a] *Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA*

[b] *AT&T Research Labs, Florham Park, NJ 07932, USA*

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. We present the underlying representations necessary in our approach, and provide an overview of the boundary computation algorithm, which is described in more detail in a followup paper.

Key words: Solid modeling, exact arithmetic, robustness, boundary computation

---

1 Introduction

Constructive Solid Geometry (CSG) and Boundary Representations (B-rep) are two major approaches for representing solids [3,29,13,24]. CSG implicitly represents a solid as an algebraic expression, and B-rep explicitly stores an object as a set of surfaces. Both these representations have different inherent strengths and weaknesses, and for many applications both representations are desired.

* Supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract DAAH04-96-1-0257, NSF Career Award CCR-9625217, ONR Young Investigator Award (N00014-97-1-0631), Honda, Intel and NSF/ARPA Center for Computer Graphics and Scientific Visualization

Many of the current solid modeling systems are based on B-reps, and Boolean combinations (union, difference, intersection) are some of the common operations performed. Computing the B-rep of the resulting solid (after performing Boolean operations) is an important operation in these systems. In this paper, objects are sculptured solids, whose boundary can be represented using trimmed rational parametric surfaces. This is a wide family of objects that can exactly represent quadrics, tori and free-form solids.

The first systematic study of CSG to B-rep conversion appeared in [29] and nowadays the algorithms for conversion are relatively well understood [13,24,5,31,22]. However, the problem of *robust* and *accurate* computation of the boundary is considered one of the difficult problems in geometric and solid modeling [6,11,10]. It is important that the computed B-rep be accurate, or at least topologically consistent, and this is jeopardized by even small amounts of error in the representation of the model or in finite-precision computations (e.g. round-off errors). Robustness problems are generally classified as due to either numerical problems or degeneracies.

A number of approaches, mostly restricted to polyhedral modelers, have been proposed for robust and accurate B-rep computation. One of the most common approaches for dealing with numerical error is based on using *tolerances* with floating-point arithmetic [15], however it is hard to decide a global tolerance value for all computations. Other approaches for dealing with numerical error include adaptive tolerances [34] and interval arithmetic [14]. Symbolic reasoning [12] and redundancy elimination [8] are among the other methods used to increase robustness in solid modeling applications.

B-rep computation algorithms involve accurate evaluation of the sign of arithmetic expressions, which can present problems for floating-point arithmetic when the value of the expression is close to zero. If this problem is not properly addressed, the resulting algorithm becomes *unreliable*. Many algorithms based on *exact arithmetic* have been proposed for reliable numeric computation for polyhedra [35,9,2,13]. Often, these algorithms use a fixed upper bound on the bit-length of arithmetic required to evaluate geometric predicates. 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 [9]. Besides reliable computation, exact arithmetic allows the use of symbolic perturbation to handle degeneracies [37]. The perturbation scheme greatly simplifies the implementation of the solid modeler.

There is relatively little work on robust or accurate B-rep computation algorithms for curved primitives. Algorithms to handle degenerate intersections between quadrics have been presented in [25]. 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, for a number of reasons, to be extremely slow and complex. 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. Techniques using bit-length estimates may, in the worst case, require bit-lengths which are exponential with respect to the degree of the algebraic functions [4,39]. Moreover, many representations and predicates that are well-understood in the linear domain become more complex in the curved domain. 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 approach is efficient in the sense that it is significantly faster than other proposed approaches based on exact computation for algebraic primitives, which use worst-case bounds on bit length or are for general algebraic systems (as available in computer algebra systems). The approach presented does not completely address the robustness issue, since it does not effectively deal with all degeneracies, although elimination of numerical errors alleviates some of the robustness problems. Our contributions include:

- **Representation:** We present efficient and exact representations for points, edges, and surfaces using algebraic sets along with a topological representation.
- **B-rep Computation:** We present an overview of the algorithm for B-rep computation over our kernel routines (further details are given in a followup paper [18]).
- **Handling Degeneracies:** We identify most cases where degeneracies can affect our algorithm, and propose ways to identify and resolve some of them.

The resulting algorithm and system works 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 the BRL-CAD system [28], is composed of more than 5000 solids, each defined using 3-12 Boolean operations on such low-degree solids. Finding the boundary representation for complex models, such as that of the Bradley, remains a major challenge. Currently, no solid modeling system is capable of completely evaluating the boundary of the entire Bradley model [27].

Two definitions are in order. When we say that our approach is *efficient*, we say so in comparison to other exact approaches. It does not always compute to worst-case precision, as some approaches do, nor does it rely on a general algebraic system solver (usually based on Grobner bases), which provides more information than is necessary and can take much longer to compute. In

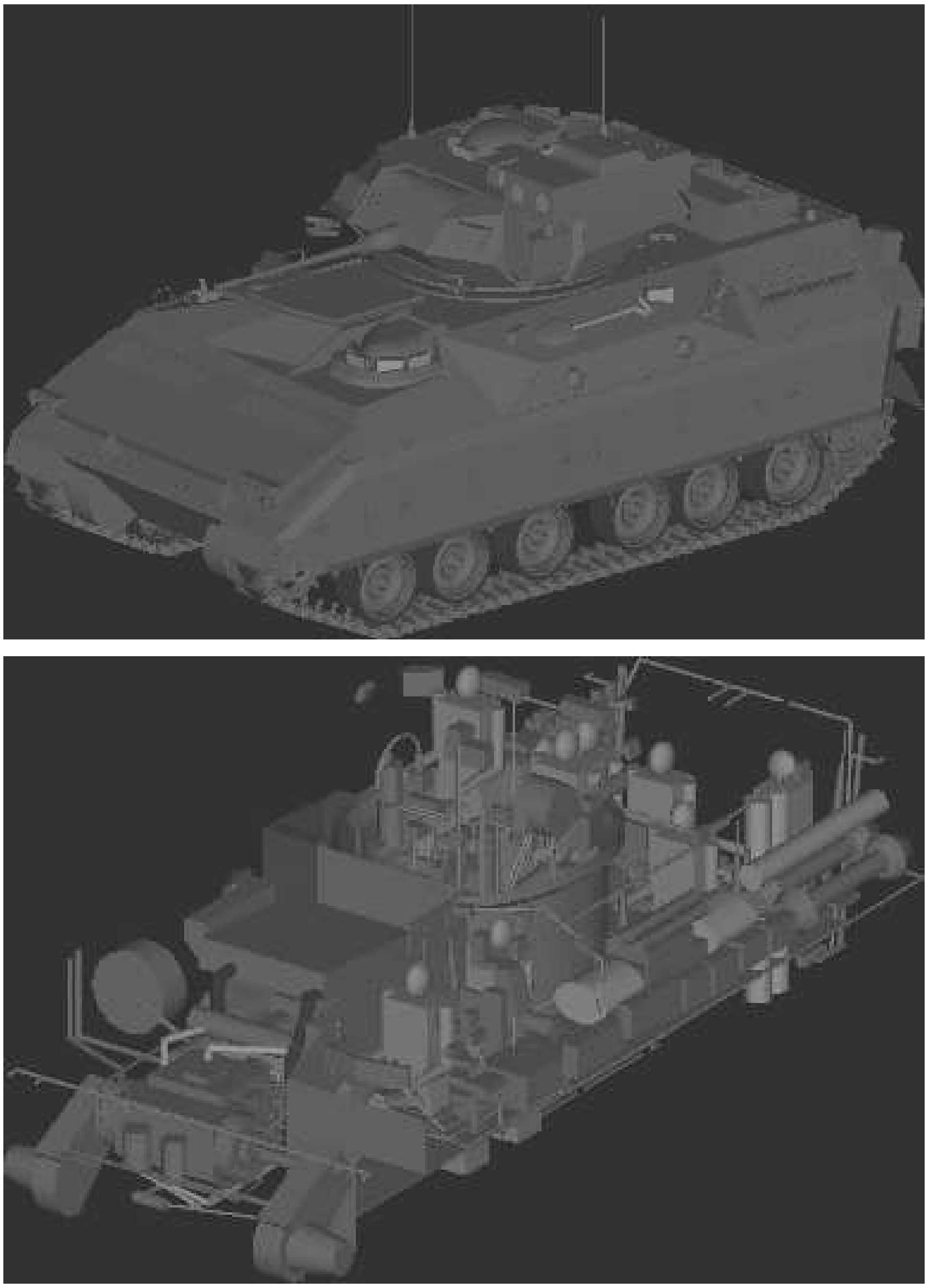

Fig. 1. Exterior and Interior of a Bradley fighting vehicle (Model courtesy of Army Research Lab). The model is composed entirely of ellipsoids, cones, polyhedra, and tori. These images were ray-traced from the CSG representation of the model, which was developed using BRL-CAD. According to the BRL-CAD developers, none of the current commercial systems are capable of finding the boundary of a model of this complexity. Generating the b-rep of a model such as this is a driving problem for our work.

comparison to a floating-point based approach, our approach will seem very inefficient. When we say that our representations and computations are *exact*, we mean that we maintain enough information to uniquely specify any quantity, and that we can make all decisions based on these quantities without error. This allows us to make all geometric queries unambiguously, and fits into the interpretation of exact geometric computation as described by Yap [38].

Organization: The rest of this paper is organized as follows. Section 2 discusses our representation for solids and algebraic numbers. Section 3 gives an overview of our algorithm and describes why we use exact arithmetic. Section 4 discusses degeneracies and Section 5 concludes with a mention of possible areas for extensions and future work. The appendix discusses key background material and parameterizations of specific solids.

A followup paper [18] will provide the details of the various steps in our algorithm, along with a discussion of the kernel routines which form its basis. Also included there are a brief analysis of our approach, preliminary implementation results, and an example.

A preliminary version of this paper was presented at the Solid Modeling '97 conference [19].

2 Representations

In this section, we present our representation and data structures for a solid. We list the assumptions we make about the format of the solid in this section, and discuss the representation of algebraic numbers, which are necessary to specify the vertices of our objects. We also describe some considerations in the type of input which can be handled. Other material (multipolynomial resultants, multivariate Sturm sequences) which is used for our representations is discussed in the appendix.

2.1 Representation of solids

Every solid is represented as a set of *trimmed* parametric surface patches which define the solid boundary. We assume that the solid is closed and compact. We represent each surface patch $\mathbf{F}(s,t)$ as a *rational function* with *rational* coefficients. This kind of parameterization is possible for all quadric surfaces such as spheres and cylinders, surfaces of revolution, and tori (some examples are shown in the appendix). The domain of the patch is the unit square in

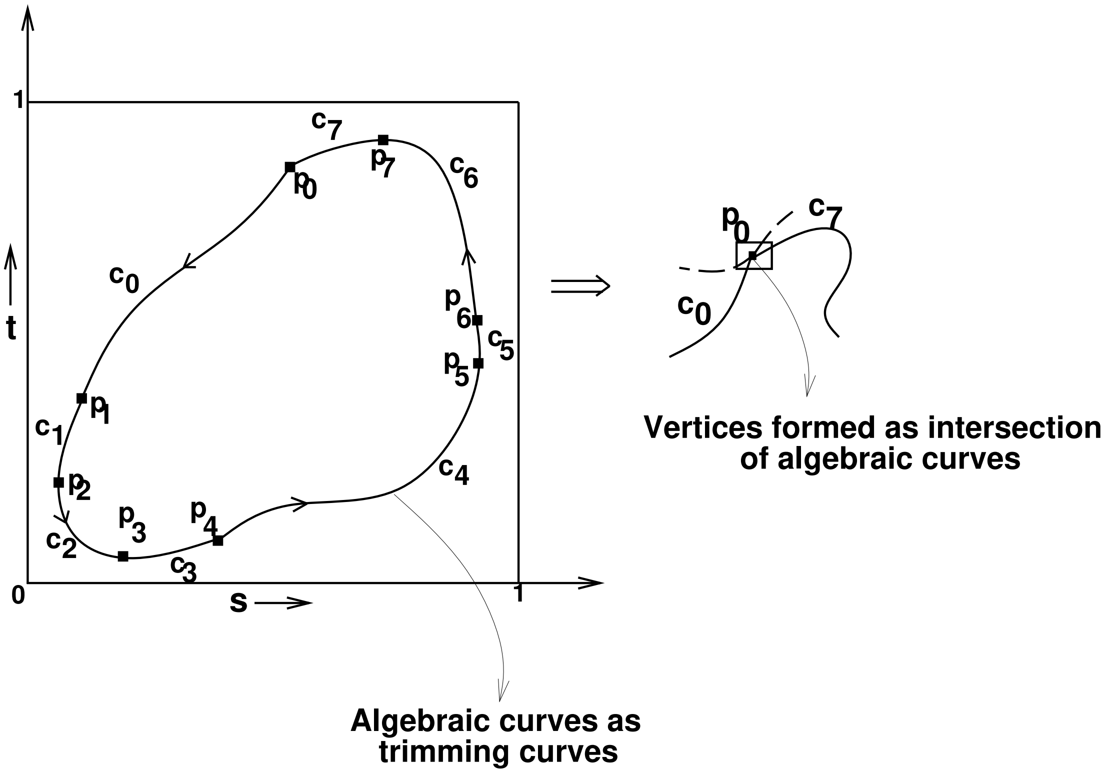

Fig. 2. Representation of a trimmed patch as algebraic curve segments

the (s, t)-plane $(0 \leq s, t \leq 1)$. If we are given a different rectangular domain, we can always reparameterize to $(0 \leq s, t \leq 1)$. We also store an implicit form for the surface, which can be obtained from a parametric form using multipolynomial resultants, as described in the appendix.

Topological information of the solid is maintained in terms of an adjacency graph, giving the connectivity between faces. It is similar to the winged-edge data structure [36]. 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. Such cases would fall under the category of degeneracies, which are addressed in section 6. Given this assumption, it can be shown that an unambiguous topological representation is possible for a solid.

A trimmed patch is described by 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). Each $\mathbf{c_i}$ is a segment of an algebraic curve. 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 parameterization [16]. We also assume that the trimming curves do not contain singularities (self intersections or vanishing tangents) in the domain of the patch. 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 computed by solving a set of polynomial equations, and their coordinates are 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 is formed by 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. Each graph vertex represents a patch, with graph edges expressing the adjacency information (i.e. which patches are next to each other). We also maintain the two faces that are adjacent to each edge, and an anticlockwise order of faces around each vertex.

2.2 Representation of algebraic numbers

Each of the vertices in the solid is defined as a point at the intersection of three surfaces, i.e. a root of a system of three polynomial equations in three variables with rational coefficients. In the domain of one of the surfaces, the vertex is seen as the intersection point of two algebraic curves, with each curve being the intersection curve with one of the other two surfaces. A vertex in the patch domain is therefore the common solution of two equations, $f(s,t) = 0$ and $g(s,t) = 0$. These roots are real algebraic numbers, and usually cannot be represented exactly using finite precision numbers. Notice that a real algebraic number is the unique solution of an equation, $f(s) = 0$, within some interval, $a \leq s < b$. In our system, we represent each algebraic coordinate as a rational rectangle (i.e. an axis-aligned rectangle whose four vertices have rational coordinates), whose size can be reduced to any desired level. The rational rectangle is guaranteed to isolate a single root of $f(s,t)$ and $g(s,t)$ (regardless of its multiplicity). Our root isolation algorithm uses *multivariate Sturm sequences* as proposed by Milne [26], and is described further in the appendix. The overall approach we use will work with *any* method which allows algebraic numbers to be represented as described above, and we are exploring other methods which appear more promising [17]. We use multivariate Sturm sequences here because they offer a direct and relatively simple way of achieving the representation.

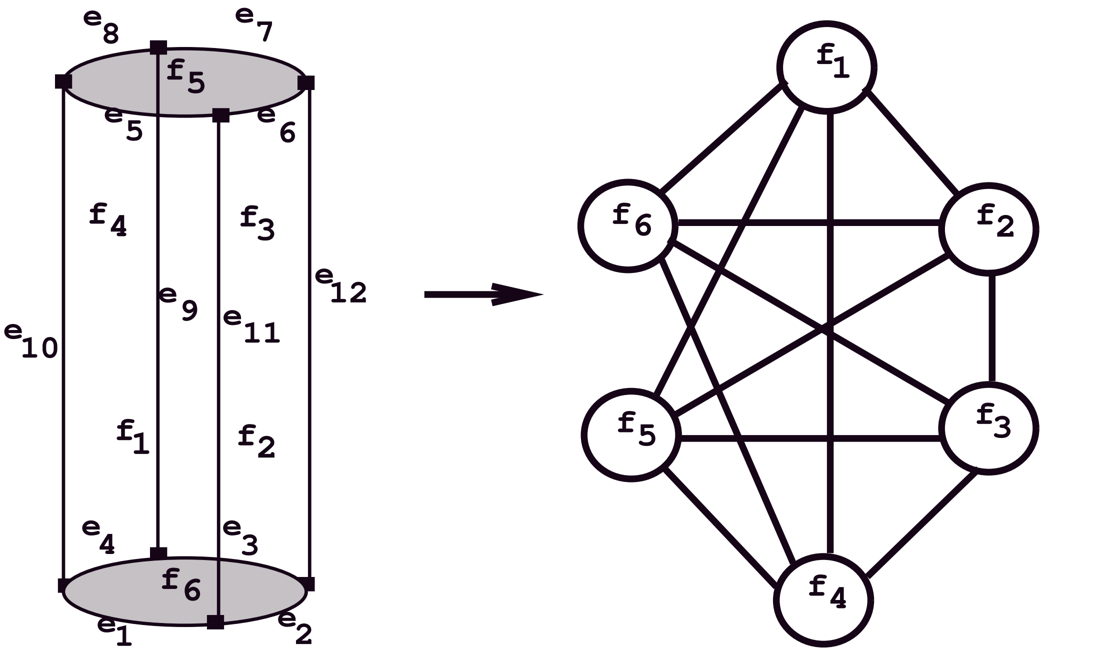

Fig. 3. A cylinder and its face connectivity structure

2.3 Input considerations

The approach we will outline can be used for solids with any degree surface. For high-degree surfaces, however, the computations can become exceedingly slow. This is because high degree surfaces can intersect in curves of very high degree. For example, two bicubic Bezier patches can intersect in a curve of degree 108 in the patch domain.

For this reason, we focus our efforts on optimizing for and testing on surfaces of low degree. If we consider objects with at most quadric surfaces, their parameterizations will be degree two or less, and be acceptable for our approach. This includes a large number of common CSG primitives, such as ellipsoids, cones, polyhedra, and tori. As was mentioned earlier, these primitives are enough to construct the entire Bradley fighting vehicle model pictured in Fig. 1. The intersection curves for such objects will be of degree no greater than four in s and t, the patch parameters. Finding the intersection point for two such curves (the most complicated operation we have to perform) will involve working with matrices of size at most 64×64.

Examples of the way we can parameterize a few of the common shapes are given in the appendix.

A second input consideration is the use of floating-point data. Often, a solid will be created by a system which uses a floating-point representation. If we are to use such solids in our approach, we need to be able to handle floating-point data. Fortunately, each floating-point number can be exactly represented by a rational number. For example, the number $0.537291 = \frac{537291}{1000000}$. A problem can arise when one wishes to represent a fraction such as $\frac{1}{3}$, yet the data has been rounded in floating-point (to 0.33333333, for example). This is really a problem in loss of accuracy because of the original floating-point representation, and we choose to make no assumptions about the source of such data. Even if we did choose to use an approximation, as long as the approximation was made one time only for the *original* data, our exact methods will still guarantee consistent results.

Another area which can cause problems is if one wishes to perform operations which cannot be performed in exact rational arithmetic. A common example of this is rotation. An operation such as "Rotate the object by X degrees" is usually impossible to perform on our structure, since it often will involve the use of irrational numbers. However, if the rotation is given as a transformation matrix with rational elements, it can be applied to our representations. Transformation matrices are already used in some systems, such as the previously mentioned BRL-CAD [28]. Such transformation matrices provide an infinite family of rotations which can be arbitratily close to any desired rotation.

3 Algorithm outline and kernel routines

In this section, we give a brief overview of our algorithm and identify some steps that are susceptible to failure when using finite precision arithmetic.

3.1 Computation of a Boolean operation

Fig. 4 gives a graphical outline of the major steps of our algorithm, the kernel routines (outlined in bold), and the dependencies of the major steps of our algorithm. The details of each step and efficient computation of the kernel routines is given in the followup paper [18].

The overall approach is decomposed into two stages.

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

(1) Evaluate the intersection curve(s) between the two untrimmed patches.
(2) Find the points where the intersection curve meets the patch boundary.
(3) Decompose the intersection curve into a set of monotonic curve segments.
(4) Find the points where the intersection curve meets the trimming boundary, and subdivide the trimming and intersection curves, keeping track

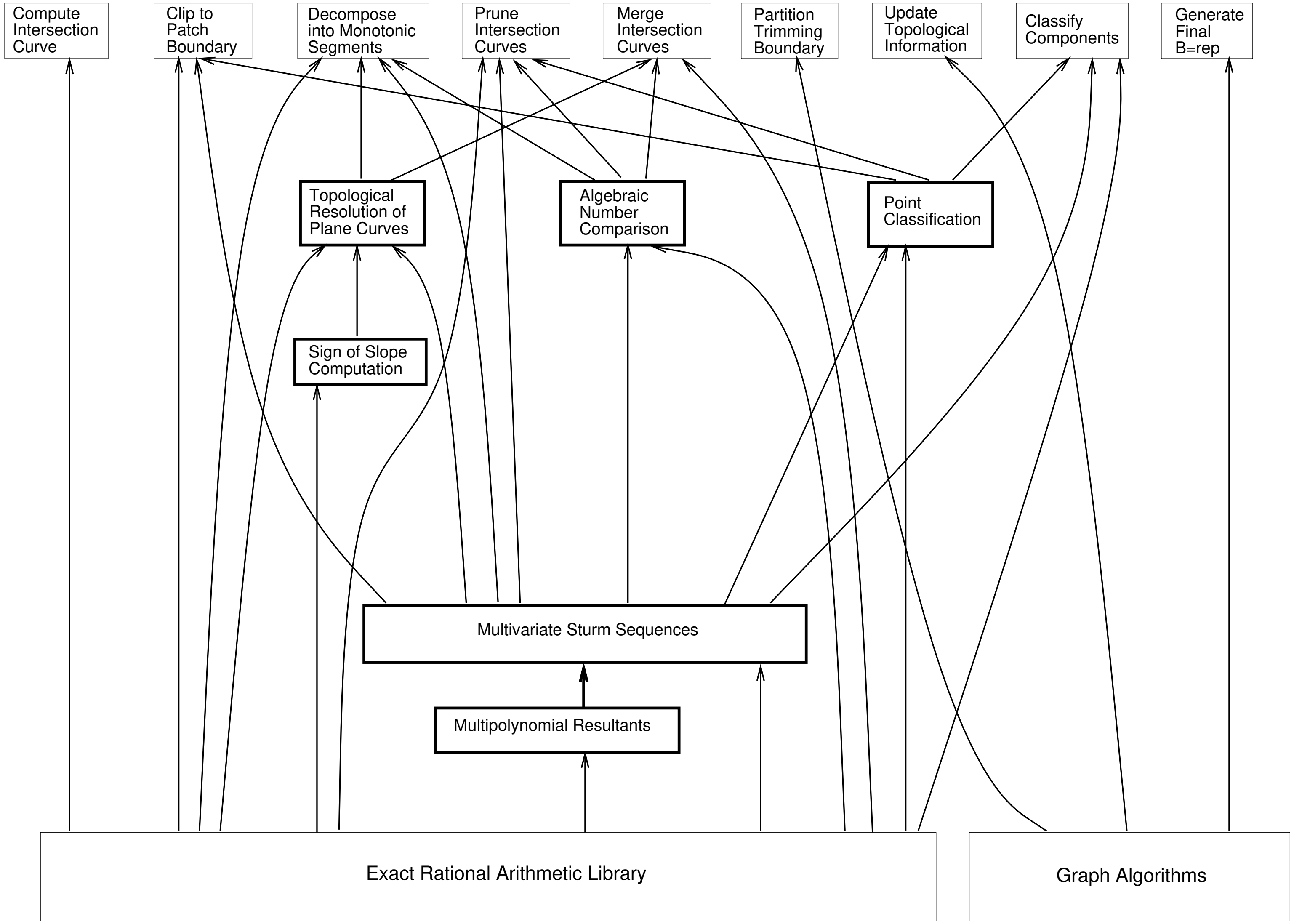

Fig. 4. An overview of the algorithm's components. The kernel routines are those outlined with bold boxes. Arrows indicate dependencies between various components.

of which curves and surfaces are then adjacent to each other.

II . Curve merging and boundary computation:

(1) Merge intersection curves together in each patch.
(2) Partition the patch domain.
(3) Update topological information by storing adjacency information for the new components introduced.
(4) Shoot rays from one solid toward the other solid to classify components as inside or outside.
(5) Propagate the information from step 3 in the adjacency graph to compute the boundary of the resulting solid.

3.2 Need for exact arithmetic

The use of exact arithmetic has been shown to be useful (and probably necessary) for dealing with degeneracies in the polyhedral domain [9], and so it is likely to be needed in the non-linear domain. In addition, the use of finite precision arithmetic can result in numerical problems for non-degenerate cases. Approximating the coefficients of high-degree polynomials by limited precision numbers can lead to severe problems, drastically changing the roots. The well-known Wilkinson polynomial, $f(x) = (x-1)(x-2)\cdots(x-20)$, is an extreme example of this. Our own experience in implementing a floating-point based curved-surface modeler [21] has shown that floating-point algorithms can lead to problems with robustness, even in non-degenerate cases. As stated earlier, there are some models which no current system is capable of handling [27]. This failure is likely due in large part to the inaccuracies of floating-point based arithmetic on curved solids. By using exact arithmetic, we eliminate robustness and accuracy problems due to numerical error, and set the stage for future attempts to deal with degenerate cases.

We shall now identify two areas 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 boil down to incorrect results from either point orientation tests or comparison between two floating-point numbers.

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 (e.g. [13,33]). As a result, the curve is approximated as piecewise linear curves or splines to within a fixed tolerance (which is either too conservative or arbitrarily chosen), or as algebraic curves with floating-point coefficients. In any case, the intersection curve has been evaluated only to a certain precision. The level of

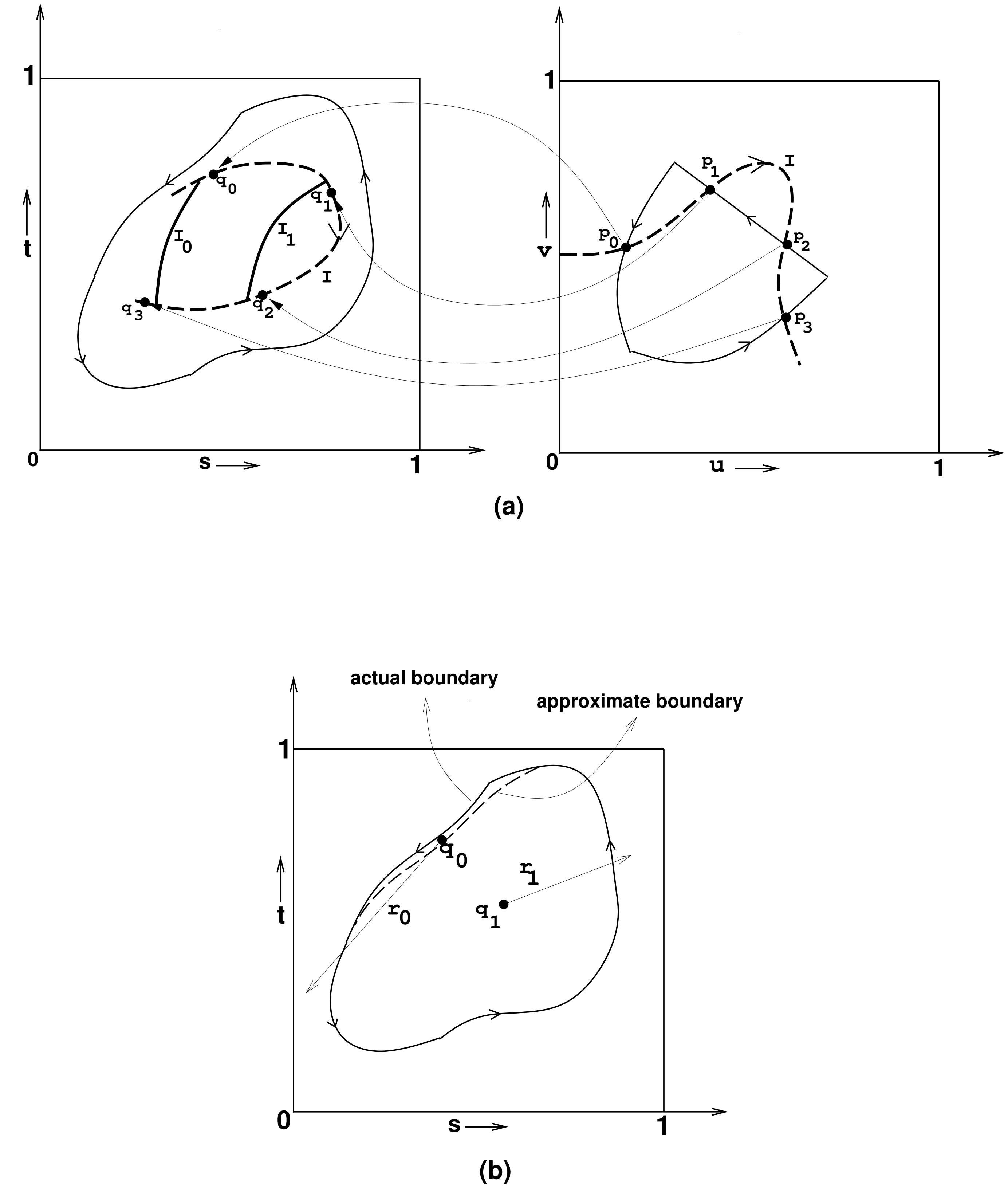

Fig. 5. (a) Inaccurate point inversion for curve merging (b) Inaccurate point classification

precision possible may be limited (due to the use of floating-point arithmetic or representations, for example), and it is usually difficult to determine what precision is desired ahead of time.

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. 5(a) shows 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 intersection curves on the left patch obtained from other surfaces. 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. If the boundary curves or the intersection curve are not accurate, neither are the $\mathbf{p_i}$'s. The curves may be inaccurate due to computation (e.g. error introduced by doing floating-point arithmetic), or due to representation (e.g. approximating a curve by lines or splines). The computed (and thus approximated) $\mathbf{p_i}$'s 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*. Two problems can arise in inversion:

- There may not be any corresponding point on the other patch (because $\mathbf{p_i}$'s do not lie exactly on the intersection curve).
- The $\mathbf{q_i}$'s could be positioned such that the curve segments $\mathbf{q_0 q_1}$ and $\mathbf{q_2 q_3}$ 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.

3.2.2 Component classification

Another area where floating-point errors result in failure of the algorithm is during component classification. As we will describe in a followup paper [18], we use ray shooting for this purpose. The entire computation boils down to classifying whether a point lies inside or outside the trimming region. Fig. 5(b) shows an example. In many cases, classifying points like $\mathbf{q_1}$ is not a problem. One ray-shooting query will determine whether it is inside or outside. However, consider a point like $\mathbf{q_0}$ which lies very close to the boundary. Approximate representations of the trimming boundary makes classifying $\mathbf{q_0}$ a major problem. Depending on the choice of ray directions and the tolerances used we may get different classifications. This error can result in topologically inconsistent answers.

There are a number of similar problems which plague floating-point modelers, and resolving these situations is no different from the ones highlighted. Based on our experience with implementing floating-point modelers, and the other reasons just discussed, we believe that using exact arithmetic and representa-

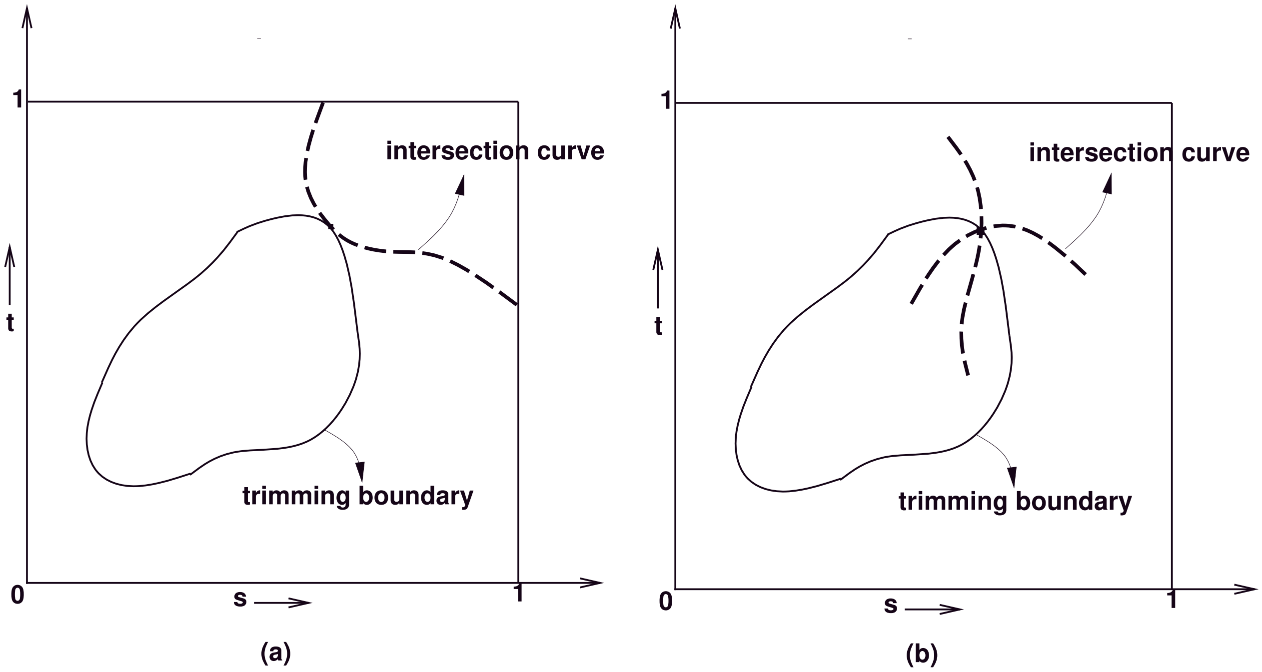

Fig. 6. (a) Surface-edge contact degeneracy (b) Four surfaces meeting at a point

tion is important in achieving reliable B-rep computation.

4 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. The details given here for detecting and handling some of the degenerate cases rely on an understanding of the steps in performing a Boolean operation, as is provided in [18]. Potential degeneracies include:

- **Two surfaces meeting at a point:** This is a singularity (since there is no tangent at the point), which we assume does not occur. If it does occur, we can find it by noticing that the intersection curve has an s turning point and a t turning point at the same position.
- **Two surfaces meeting at a curve:** This is a degenerate case when the surfaces are tangent to each other along that curve. We will be able to detect this when we generate the adjacency graph by finding whether two components which should be adjacent are actually part of the same component.
- **Two surfaces overlapping:** This corresponds to a face-face overlap in the polyhedral domain. Here, though, if the surfaces we use have an irreducible implicit form, then they will not overlap unless this form is identical.

- **Three surfaces meeting in a curve:** This is an edge-face contact in the polyhedral domain, and will appear as two overlapping curves within a patch domain. Again, if the intersection curves have an irreducible implicit form, they will not overlap unless that form is identical.
- **A surface just touching an edge:** In our representation, this will appear as an intersection curve which is tangent to a trimming curve (see Fig. 6(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. This does not allow us to use the speed-up of propagating the information about one component of the intersection curve to all other components of that curve.
- **Four surfaces meeting at a point:** This is the foundation for several types of degeneracies and will be discussed next.

Examples of four surfaces meeting at a point 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. 6(b)). Assume these three curves are f_1, f_2, and f_3. We can find out whether this case has occurred by checking equality of the intersection of f_1 and f_2 in some interval with the intersection of f_1 and f_3 (or f_2 and f_3) in that same interval.

Most degeneracies in the polyhedral case are classified into the category of four planes meeting at a point. It has been shown [9] that a simple perturbation scheme applied to a single basic geometric predicate can eliminate these degeneracies. No obvious extension of this method exists in the curved surface domain, though there is hope that some perturbation method can be developed which would work similarly. Since perturbation methods usually rely on some form of exact computation, our algorithm should be amenable to a perturbation-based approach.

5 Extensions and future work

In this paper, we have presented representations for computing B-reps for Boolean combinations of low-degree solids specified with rational parametric surfaces. A followup paper to this work [18] will provide more specific details of the kernel routines and algorithms used in the Boolean computation. 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, maintaining an exact representation throughout. By eliminating some robustness problems (those due to numerical errors), our method is a step toward a fully robust system. To claim complete robustness, however, there are some remaining areas (such as degeneracies) which must be addressed.

There are a number of ways in which the method we have described might be extended. These are:

- Investigating the use of parallelism or a combination of floating-point and exact arithmetic to get a faster implementation. We have already begun some work in this area [20].
- Handling singularities (such as self intersections) in intersection curves and surfaces. This will be necessary in any robust system.
- Dealing with all degeneracies (possibly using perturbations). We cannot claim robustness until this is done.
- Making the approach more efficient for higher degree surfaces. Currently, it is only efficient for low degree surfaces.
- Extending our method to deal with non-manifold cases or with non-parametric or non-algebraic surfaces. This would involve a major change in the approach described.

5.1 Acknwoledgements

We would like to thank the Army Research Lab for providing access to the BRL-CAD system and the Bradley Fighting Vehicle model. We would also like to thank the reviewers for their many helpful comments and suggestions on the content and style of this paper.

References

[1] S.S. Abhyankar and C. Bajaj. Computations with algebraic curves. In *Lecture Notes in Computer Science*, volume 358, pages 279–284. Springer Verlag, 1988.

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

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

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

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

[6] 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.

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

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

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

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

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

[12] C. Hoffmann, J. Hopcroft, and M. Karasick. Robust set operations on polyhedral solids. *ieeeCGA*, 9(6):50–59, 1989.

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

[14] Chun-Yi Hu, Nicholas M. Patrikalakis, and Xiuzi Ye. Robust interval solid modelling part 1: Representations. *Computer Aided Design*, 28(10):807–817, October 1996.

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

[16] S. Katz and T.W. Sederberg. Genus of the intersection curve of two rational surface patches. *CAGD*, 5, 1988.

[17] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. Mapc: A library for efficient and exact manipulation of algebraic points and curves. Technical Report TR98-038, University of North Carolina, Chapel Hill, 1998.

[18] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic: Ii - computation. *Computer Aided Geometric Design.*

[19] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic. In *ACM/SIGGRAPH Symposium on Solid Modeling*, pages 42–55, 1997.

[20] J. Keyser, S. Krishnan, D. Manocha, and T. Culver. Efficient and reliable computation with algebraic numbers for geometric algorithms. Technical Report TR98-012, Department of Computer Science, University of North Carolina, 1998.

[21] S. Krishnan. *Efficient and Accurate Boundary Evaluation Algorithms for Sculptured Solids.* PhD thesis, Department of Computer Science, University of N. Carolina at Chapel Hill, 1997. Available at http://www.cs.unc.edu/~krishnas/dissertation.html.

[22] 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.

[23] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

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

[25] 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.

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

[27] M. Muuss. Army Research Lab, Personal Communication, 1996.

[28] 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.

[29] 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.

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

[31] R F Sarraga. Algebraic methods for intersection. *CVGIP*, 22:222–238, 1983.

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

[33] T.W. Sederberg and T. Nishita. Curve intersection using bézier clipping. *CAD*, 22:538–549, 1990.

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

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

[36] Kevin J. Weiler. *Topological Structures for Solid Modeling.* PhD thesis, Computer and Systems Engineering, Rensselaer Polytechnic Institute, 1986.

[37] C. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *JSC*, 40:2–18, 1990.

[38] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7:3–23, 1997.

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

6 Appendix

In this portion, we provide some background information about key concepts (i.e. multipolynomial resultants and multivariate Sturm sequences) which are related to the representation we use. We also provide details for representing certain primitive solids in our format.

6.1 Multipolynomial resultants

Elimination theory investigates the conditions under which sets of polynomials have common roots. Usually, it concerns itself with sets of k homogeneous polynomials in k 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 *[30] 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.*

Macaulay [23] provided a general method for eliminating k variables from n homogeneous polynomials. His resultant is expressed as a ratio of two determinants. A single determinant formulation exists for $k = 2$ and 3 [30,7]. For $k = 3$, however, [7] gives a resultant for the case of three equations having the same degree. In our application, it is sufficient to compute resultants for the cases when $k = 2$ and 3.

For the case of $k = 2$, Sylvester's method [30] 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 $n \leq m$, the Sylvester's resultant is

$$\left| \begin{array}{cccccccc} a_n & a_{n-1} & \cdots & a_0 & & & & \\ & a_n & a_{n-1} & \cdots & a_0 & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & a_n & a_{n-1} & \cdots & a_0 & \\ b_m & b_{m-1} & \cdots & \cdots & b_0 & & & \\ & b_m & b_{m-1} & \cdots & \cdots & b_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & b_m & b_{m-1} & \cdots & \cdots & b_0 & \end{array} \right| \tag{3}$$

Sylvester's method is a key component in our method for evaluating multivariate Sturm sequences, as described later in this section.

The case of $k = 3$ arises when computing the implicit representation of a parametric surface. Implicitizing the 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

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

Although usually we will know the implicit form of the surface ahead of time, if we do not, we can use Dixon's resultant [7] to compute it, as described in [32]

6.2 Multivariate Sturm sequences

Here, we describe briefly the algorithm proposed by Milne [26] to compute the number of common real solutions of n polynomials in n variables inside an n-dimensional rectangle. Multivariate Sturm sequences are used to isolate the vertices of our solids. This algorithm is an extension of the univariate case 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. The denominator removes the extraneous u factor which is introduced from the repeated resultant computation. We use the Sylvester resultant [30] 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 $\mathbf{V}$ operator at (a_1,a_2) ($\mathbf{V}(M(a_1,a_2))$) gives the number of sign changes between consecutive terms of the sequence evaluated at (a_1,a_2). Correspondingly, the $\mathbf{P}$ operator is defined as $\mathbf{P}(M(a_1,a_2)) = n-1-\mathbf{V}(M(a_1,a_2))$.*

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

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

The justification for various steps and extension to arbitrary dimensions can be found in [26].

6.3 Parameterizations of solids

In this section we demonstrate the parameterizations which can be used for a few common solids. This is not intended to be a comprehensive list of possible parameterizations or objects we can handle, but rather to demonstrate that our technique is acceptable for use on several common cases. In each case, the parameterization is at most degree 2 in both s and t.

We present only the parameterizations here. For many of the surfaces, the implicit form will already be known. For others, however, we can use the Dixon resultant to generate the implicit form. Such computation is "precomputed" and stored before our approach is used—we assume that we are given solids already in the appropriate format, including both the implicit and parametric forms.

The parameterizations for polyhedra are relatively simple and will not be shown here. Instead, we will give parameterizations for ellipsoids, cones, and tori based upon given parameters.

Ellipsoids: Assume that an ellipsoid is described by the following information:

- The center point: (X_p, Y_p, Z_p)
- Vectors pointing from the center of the ellipsoid to the ends of the three axes of the ellipsoid:

$$[X_a, Y_a, Z_a], [X_b, Y_b, Z_b], [X_c, Y_c, Z_c]$$

We parameterize this into eight trimmed patches. The trimming curves in the patch domain are the s-axis from 0 to 1, the t-axis from 0 to 1, and the curve $s^2 + t^2 = 1$ from $(1, 0)$ to $(0, 1)$. One of these patches would have the following parameterization:

$$\begin{aligned}
X(s,t) &= X_a(1 - s^2 - t^2) + X_b(2s) + X_c(2t) + X_p(1 + s^2 + t^2) \\
Y(s,t) &= Y_a(1 - s^2 - t^2) + Y_b(2s) + Y_c(2t) + Y_p(1 + s^2 + t^2) \\
Z(s,t) &= Z_a(1 - s^2 - t^2) + Z_b(2s) + Z_c(2t) + Z_p(1 + s^2 + t^2) \\
W(s,t) &= 1 + s^2 + t^2
\end{aligned}$$

The other seven patches would have similar formulations.

Cones: Assume that a generalized cone is given by the following information:

- The center of the base ellipse: (X_p, Y_p, Z_p)
- The vector from the center of the base ellipse to the center of the top ellipse: $[X_t, Y_t, Z_t]$
- The two vectors pointing from the center of the base ellipse to the outer edge along the bottom ellipse axes:

$$[X_a, Y_a, Z_a], [X_b, Y_b, Z_b]$$

- The two vectors pointing from the center of the top ellipse to the outer edge along the top ellipse axes:

$$[X_c, Y_c, Z_c], [X_d, Y_d, Z_d]$$

We parameterize this into one trimmed patch each for the top and bottom ellipses, and four patches for the sides of the cones. The top and bottom patches are trimmed by four trimming curves (one for each "side" patch). Since these are planar, their parameterization is relatively simple. The four side patches are formed from the entire $[0,1] \times [0,1]$ patch domain in s and t. The four parameterizations are similar to each other, and one is given here:

$$
\begin{aligned}
X(s,t) &= (X_a + (X_c - X_a)t)(1-s^2) + (X_b + (X_d - X_b)t)(2s)X_t(t)(1+s^2) + X_p(1+s^2)\\
Y(s,t) &= (Y_a + (Y_c - Y_a)t)(1-s^2) + (Y_b + (Y_d - Y_b)t)(2s)Y_t(t)(1+s^2) + Y_p(1+s^2)\\
Z(s,t) &= (Z_a + (Z_c - Z_a)t)(1-s^2) + (Z_b + (Z_d - Z_b)t)(2s)Z_t(t)(1+s^2) + Z_p(1+s^2)\\
W(s,t) &= 1 + s^2
\end{aligned}
$$

Tori: Assume that the information for a torus is given as follows:

- The center of the torus: (X_p, Y_p, Z_p)
- The unit vector pointing in the direction "normal" to the torus—i.e. through the hole in the center: $[X_n, Y_n, Z_n]$
- Two unit vectors perpendicular to the "normal vector"; they point from the center toward the "ring" of the torus:

$$[X_a, Y_a, Z_a], [X_b, Y_b, Z_b]$$

- The radius from the center of the torus to the center of the "ring": A
- The radius of the "ring": B

This can be parameterized by 16 patches, each of which is again formed from the entire $[0,1] \times [0,1]$ patch domain in s and t. The 16 patches have similar parameterizations, and one is given here:

$$
\begin{aligned}
X(s,t) &= (X_a(1-s^2) + X_b(2s))(A(1+t^2) + B(1-t^2)) +\\
&\quad X_n(1+s^2)(2Bt) + X_p(1+t^2)(1+s^2)\\
Y(s,t) &= (Y_a(1-s^2) + Y_b(2s))(A(1+t^2) + B(1-t^2)) +\\
&\quad Y_n(1+s^2)(2Bt) + Y_p(1+t^2)(1+s^2)\\
Z(s,t) &= (Z_a(1-s^2) + Z_b(2s))(A(1+t^2) + B(1-t^2)) +\\
&\quad Z_n(1+s^2)(2Bt) + Z_p(1+t^2)(1+s^2)\\
W(s,t) &= (1+t^2)(1+s^2)
\end{aligned}
$$