

ESOLID – A System for Exact Boundary Evaluation

John Keyser* Tim Culver Mark Foskey
Texas A&M University think3, Inc. University of North Carolina
keyser@cs.tamu.edu culver@acm.org foskey@cs.tamu.edu

Shankar Krishnan Dinesh Manocha
AT&T Research Labs University of North Carolina
krishans@research.att.com dm@cs.unc.edu

September 2, 2003

Abstract

We present a system, ESOLID, that performs exact boundary evaluation of low-degree curved solids in reasonable amounts of time. ESOLID performs accurate Boolean operations using exact representations and exact computations throughout. The demands of exact computation require a different set of algorithms and efficiency improvements than those found in a traditional inexact floating point based modeler. We describe the system architecture, representations, and issues in implementing the algorithms. We also describe a number of techniques that increase the efficiency of the system based on lazy evaluation, use of floating point filters, arbitrary floating point arithmetic with error bounds, and lower dimensional formulation of subproblems.

ESOLID has been used for boundary evaluation of many complex solids. These include both synthetic datasets and parts of a Bradley Fighting Vehicle designed using the BRL-CAD solid modeling system. It is shown that ESOLID can correctly evaluate the boundary of solids that are very hard to compute using a fixed-precision floating point modeler. In terms of performance, it is about an order of magnitude slower as compared to a floating point boundary evaluation system on most cases.

1 Introduction

A key operation in solid modeling systems is boundary evaluation, or computing the boundary of Boolean combinations of two or more solids. A number of algorithms have been proposed in

*Correspondance may be directed to the first author at: Department of Computer Science, 3112 Texas A&M University, College Station, TX 77843-3112 USA, phone:(979)458-0167, fax:(979)847-8578

the literature for boundary evaluation, however these are hard to implement because of accuracy and robustness problems. These problems are particularly significant when dealing with curved primitives. In general, geometric computations on non-linear primitives are more susceptible to inaccuracies in representation and computation. As a result, designing a reliable solid modeling system for graphics and CAD/CAM applications remains a major challenge.

The difficulties in developing a reliable or consistent solid modeler using only fixed-precision arithmetic have been noticed and attacked by many different researchers [17, 21, 22, 25, 26, 32, 45, 51, 52]. Beyond the reliability of individual solid modeling systems, numerical inaccuracy plays a significant role in problems of data transfer, leading to an estimated loss of more than \$1 billion annually in the U.S. automobile industry alone [6]. There are many sources of numerical inaccuracy, including errors in input data, approximation of output, and roundoff error in floating-point computation or intermediate storage. Any complete treatment of the problem would require addressing issues throughout the entire modeling process. In our system, we address the inaccuracies that arise within and are then propagated by the internal system computations.

The numerical inaccuracies cause problems when they result in conflicting geometric and topological information. Such conflicts lead to serious problems, including program crashes and incorrect or impossible to realize output. Many solutions, based on symbolic relationships, tolerances, interval arithmetic, perturbation techniques, etc. have been proposed to increase the accuracy and robustness of boundary evaluation systems. One such approach is the exact computation paradigm [50]. This approach eliminates numerical error in geometric computations entirely. Unfortunately, exact implementations are often far too slow. Naive implementations of even very simple geometric operations can take several orders of magnitude longer than an equivalent floating-point implementation [30]. Various speedup techniques can improve efficiency considerably, but the use of exact computation for curved solids is still widely perceived to be too slow for practical use.

In practice, most exact approaches for boundary evaluation have been applied to polyhedral models only. Exact computations and representations are regarded as extremely slow and impractical for non-linear (curved) models. Polyhedral models are much easier to deal with exactly for

several reasons. Intersections of the planar surfaces making up polyhedra result in points with coordinates that can be expressed as rational numbers. Furthermore, in general, two planes will intersect in a line and three in a single point. In contrast, with curved surfaces two surfaces will intersect in (possibly several) curves, and three surfaces can intersect in a large number of points, each of which may have irrational algebraic coordinates. While additional bits of precision are sufficient to handle rational numbers, a different type of representation is needed to handle algebraic numbers exactly. In addition, the polynomial calculations that are required for these algebraic numbers are significantly more complex and time-consuming. While the algebraic issues involved in curved surface intersections are well-understood (e.g. as described by Hoffmann [23]), the practical difficulties and inefficiencies have prevented most work on exact solid modeling with curved surfaces. Worst-case analysis of exact computation for curved objects (e.g. [52]) has further fueled the perception that exact computation on curved solids is completely impractical.

Our work addresses this by demonstrating, for the first time, that an exact computation-based approach can achieve reasonable efficiency for boundary evaluation on curved solids, while eliminating numerical error.

1.1 Main Results

We present a system, ESOLID, that performs exact boundary evaluation of low-degree algebraic curved solids described by parametric patches. We show that ESOLID can perform these operations on real-world data in “reasonable” amounts of time. ESOLID uses exact representations throughout in order to compute accurate Boolean combinations. We present the issues and challenges involved in implementing such a system. We describe techniques for taking advantage of a sequence of low-dimension operations in order to avoid operations in higher dimensions, thus saving time. We also describe a number of other techniques that improve the efficiency of the system, including lazy representations and evaluation, floating point filters, and the use of arbitrary precision floating-point arithmetic with tight error bounds. ESOLID has been applied to a number of complex solid models, including both synthetic models and models designed using the

BRL-CAD solid modeling system. We have compared its performance with a boundary evaluation system based on floating-point computation. In terms of performance, ESOLID is less than one order of magnitude slower in most cases and no more than two orders of magnitude slower in the worst case. However, ESOLID can easily handle cases that are very hard to handle by fixed precision boundary evaluation systems. The work presented here describes the system and practical implementation aspects of an exact boundary evaluation approach proposed in a different form in our earlier work [33, 34], and builds upon the exact two dimensional operations implemented previously [35]. A shortened form of this paper has appeared in Solid Modeling '02 [31]. To the best of our knowledge, there are no previous exact implementations of boundary evaluation that achieve comparable speeds on real-world examples.

1.2 Exact Computation

The primary reason for using exact computation has been to ensure *consistency* in operations by eliminating numerical error accumulation in intermediate computations. Although input data might not be exact (e.g. positions may be inaccurate or “noisy,” and certain desirable rotations can not be represented exactly by rational numbers), exact computation is still very useful. Without exact computation, errors build up in intermediate computation, resulting in inconsistent intermediate data that can cause program crashes and incorrect or invalid output. ESOLID makes a particular interpretation of the given data, then uses exact computation for all operations on the data and exact representations for intermediate data. This eliminates problems due to intermediate error buildup. As long as a consistent interpretation of the input data can be made (not necessarily a trivial task), all further computation is guaranteed not to suffer numerical problems.

1.3 Paper Outline

In section 2 we describe previous work that has led to the development of ESOLID. In section 3 we give an overview of ESOLID, breaking it into its major components. In section 4, we describe some of the major challenges encountered in implementing ESOLID, along with the solutions that we

used. We discuss the various techniques used to increase efficiency in section 5. Section 6 presents the results of ESOLID applied to both synthetic datasets and “real world” examples. Finally, section 7 concludes with a summary of important lessons learned in the process of implementing ESOLID.

2 Previous Work

2.1 Boundary Evaluation

As a key part of conversion from a constructive solid geometry (CSG) representation to a boundary representation (B-rep), boundary evaluation is a well-studied problem in solid modeling. Specifically, boundary evaluation refers to determining the boundary of Boolean combinations of solids. Braid [4] provided one of the earliest treatments, and Requicha and Voelcker [43] gave a comprehensive description of basic boundary evaluation. Casale and Bobrow presented one of the first detailed descriptions for boundary evaluation for curved solids [7]. Today, the basic approaches for boundary evaluation are well-understood, and have been incorporated into textbooks [23, 40].

More recently, robustness in boundary evaluation has gained greater attention. Some researchers have focused on the use of exact computation for polyhedral solids. This work includes that of Sugihara and Iri [48], Yu [52], Benouamer et al. [2], Sugihara [47], and Fortune [17]. Others have proposed methods for increasing robustness that do not rely on exact computation. Hoffmann et al. [24] have described methods for increasing robustness by eliminating redundancy and checking for consistency. For eliminating numerical errors in boundary evaluation on curved solids, the work has been much more limited. Yu has explored some theoretical bounds of exact computation [52], Fang et al. have explored tolerance methods for boundary evaluation [16], Hu et al. have explored interval computations and representations [25, 26], and Desaulniers and Stewart have given limited results on the interpretation of (possibly inconsistent) output [12].

Boundary evaluation is a common part of many solid modeling systems, and there are far too many such systems to attempt to list them here. Requicha and Voelcker have listed and summarized

many of the earliest solid modeling systems [42]. Solid modeling systems have continued to be developed in recent years, such as the Recent modeling systems relevant to this work include the CSG-based BRL-CAD system from the Army Research Lab [14, 13], the IRIT system [15], and a number of research systems created to demonstrate new robustness techniques. Examples of these latter systems are ones by Fortune [17], Jackson [27], Benouamer et al. [2], Fang et al. [16], and Hu et al. [25, 26].

2.2 Exact Computation

A significant amount of work has been done on exact computation in computational geometry, solid modeling, and symbolic computation. The primary focus of this work has been on making exact computations more efficient. Usually, this involves trying to replace high bit-length numbers with less precise numbers that provide faster computation. Error bounds of various types are used to *guarantee* that the computed result is accurate enough that a particular decision based on that number is correct. Thus, the decision is made as if the numerical calculation were exact, but at a faster rate. This guarantee that all decisions are made as if the input and subsequent computation were exact (i.e. guaranteed correct decisions) is known as *exact computation*. Note that exact computation eliminates the topology vs. geometry inconsistencies that cause robustness problems.

Among the methods used to increase the efficiency of exact computations are those based on interval arithmetic [30, 28], floating-point filters [18, 19], lazy arithmetic [2], tuned computations [18, 19], precision-driven computation [50], minimized intermediate computation [8, 5], fast hardware computation [46], and modular arithmetic [18, 5]. Libraries supporting basic exact computation have been developed, with LEDA [41] and CORE [29] being notable examples. These libraries, however, have supported only linear computations and a limited set of algebraic computations, and are not sufficient for general boundary evaluation problems. While some exact methods have been applied to polyhedral solids, we are not aware of any previous practical implementations for curved solids.

2.3 Exact Boundary Evaluation on Non-linear Primitives

Keyser et al. previously presented [32, 33, 34] the outline of an approach for exact boundary evaluation. While this approach has guided our later work, the work presented here builds on this previous work and significantly extends the results presented in those papers. Specifically, the architecture specifics, data transfer algorithms, efficiency considerations, implementation issues, and performance details are presented here for the first time. Other relevant previous work by Keyser et al. includes the MAPC library [35]. MAPC provides data structures and routines for polynomials, algebraic plane curves, and two-dimensional points with algebraic coordinates. The MAPC data structures and routines, which were developed in the process of implementing ESOLID, form a primary building block for ESOLID.

3 ESOLID Overview

ESOLID is a system for performing exact boundary evaluation. Input is supported for several primitives (including the “CSG standard primitives” [23]), stored in a CSG tree that allows union, intersection, and difference operations, as well as transformations by a 4×4 matrix (see section 3.3 for a more detailed discussion of input). The internal representation supports manifold objects made up of trimmed patches with surfaces expressed as rational functions of polynomials with rational coefficients.

Note that ESOLID is designed to work correctly for parametric surfaces of arbitrary degree and complexity. For efficiency reasons, however, only low-degree (algebraic degree four or less) surfaces are practical—higher degree surfaces (such as bicubic patches) tend to take unreasonable amounts of time and memory. With each operation, ESOLID determines the boundary of the result, updating all geometry and topology to store the resulting object. Boolean operations are not supported for degenerate configurations of objects. Thus input, intermediate representations, and output must all be manifold solids, intersection curves may not have singularities, surfaces of different objects should not overlap, etc. Currently, output is provided in one of two forms: either a

human-readable exact output suitable mainly for testing and debugging, or an approximate output of trimmed Bezier patches, useful mainly for visualization (e.g. the pictures shown here).

Below, we briefly discuss the architecture and representations in ESOLID, the process of boundary evaluation, and input considerations.

3.1 Architecture

ESOLID consists of approximately 45,000 lines of C++ code, implemented on top of the LiDIA library [3]. LiDIA provides data structures and routines for exact arithmetic on rational numbers. Other libraries (such as LEDA [41]) for exact rational arithmetic could easily be used instead.

Solids in ESOLID are represented as B-reps broken up into trimmed parametric patches. Each patch is described by a surface with both a parametric and implicit form. The trimming curves are described in the same way as intersection curves between two patches, since this is how trimming curves in the output of a Boolean operation are formed. The intersections of such surfaces are stored as algebraic plane curves in the parametric patch domain. Note that these intersection curves are typically not parameterizable. For example, representing the intersection of the patches by a rational parametric curve with rational coefficients (such as a B-spline) could not be done without introducing some error. So, intersection curves (and trimming curves) are stored in implicit form with endpoints. These endpoints, since they can be the intersection of two algebraic plane curves, can have irrational algebraic coordinates. ESOLID uses the MAPC representation for points, which involves representing points as 2D intervals that are guaranteed to contain a unique intersection of two algebraic plane curves. The interval bounds are rational numbers, and the interval size can be reduced on demand.

A diagram showing the organizational structure of ESOLID is given in figure 1. The portions of ESOLID that are incorporated in MAPC are represented as an external library in the figure.

- **MAPC** provides routines for handling polynomials (`K_POLYs`), algebraic plane curves (`K_CURVEs`), and both 1D points (`K_POINT1Ds`) and 2D points (`K_POINT2Ds`) with algebraic coordinates [35]. It includes routines for determining the topology of algebraic plane curves over

a limited domain and intersecting algebraic plane curves. MAPC was developed while implementing ESOLID.

- A **K_SURF** is the ESOLID representation for a surface. It includes K_POLYs that describe the rational parametric form of the surface, as well as a K_POLY giving the implicit form.
- A **K_PATCH** describes a single patch in the B-rep. A K_PATCH includes a K_SURF defining the surface, LiDIA birationals defining the domain boundaries, and arrays of K_CURVEs defining trimming and intersection curves. Trimming curves define the boundary of the patch, and intersection curves indicate where the patch intersects patches of another solid. Each curve, of either type, is associated with a K_SURF that intersects the patch. To illustrate, let patch P be part of a surface S . If C is a curve in P 's domain, then C will be associated with a surface, S' , such that C is a subset of the intersection of S and S' . The associated K_SURFs are kept in an array parallel to the array of K_CURVEs. In some cases, the associated K_SURF is the K_SURF of a different patch from the same solid. In other cases it only exists to determine a boundary between adjacent patches, so that, e.g., a sphere may be parameterized using multiple patches.
- A **K_PARTITION** describes one subpatch formed during boundary evaluation. During boundary evaluation, each patch is subdivided into one or more subpatches based on the intersection of that patch with all the patches of the other solid. A K_PARTITION includes data denoting the particular curves in an associated (parent) K_PATCH structure that define the K_PARTITION.
- A **K_SOLID** describes the overall solid, and is made from a group of K_PATCHs. K_SOLIDs are the input and output for boundary evaluation. They can also be formed from collections of K_PATCHs, groups of K_PARTITIONs coupled with topological information, and conversion of input CSG data (from BRL-CAD).
- Topological connectivity information is kept in the individual classes. Each face stores the

list of trimming curves and the adjacent face along each curve. Each curve (edge) stores the adjacent vertices. The same 3D curve or point is often stored in more than one 2D patch domain. These “associations” (pointers to an equivalent point or curve in another domain) are also stored and are important in boundary evaluation. An overall topological graph (**K_GRAPH**) is constructed as necessary during later stages of boundary evaluation. Details of the topological information stored and proof of its sufficiency are given by Keyser [36]. Note that because of the use of exact computation, storing redundant topological information does not lead to robustness problems.

- Not shown in the figure, the **PRECISE** library [39] can be optionally included as a part of MAPC to speed up calculations involving algebraic numbers. PRECISE is an extension of the range arithmetic techniques developed by Aberth and Schaefer, and implemented in their *range* library [1].

3.2 Boundary Evaluation

Boundary evaluation is defined within the **K_SOLID** class. The traditional two-stage approach to boundary evaluation is followed in **ESOLID**. In the first stage, the patches are intersected pairwise, partitioning them into separate components. In the second stage, the partitions are identified and selectively stitched together to form the final solid. Although this traditional approach is well-understood and straightforward, a number of individual steps must be modified considerably in order to allow it to be used in an exact computation scheme. Previous papers have described some of these issues in detail [33, 34], but actual implementation highlighted the importance of other issues (e.g. curve correspondence) that had not been considered. Only a brief overview will be given here, although some steps are treated in more detail in section 4.

The procedure is as follows:

- For each pair of patches:

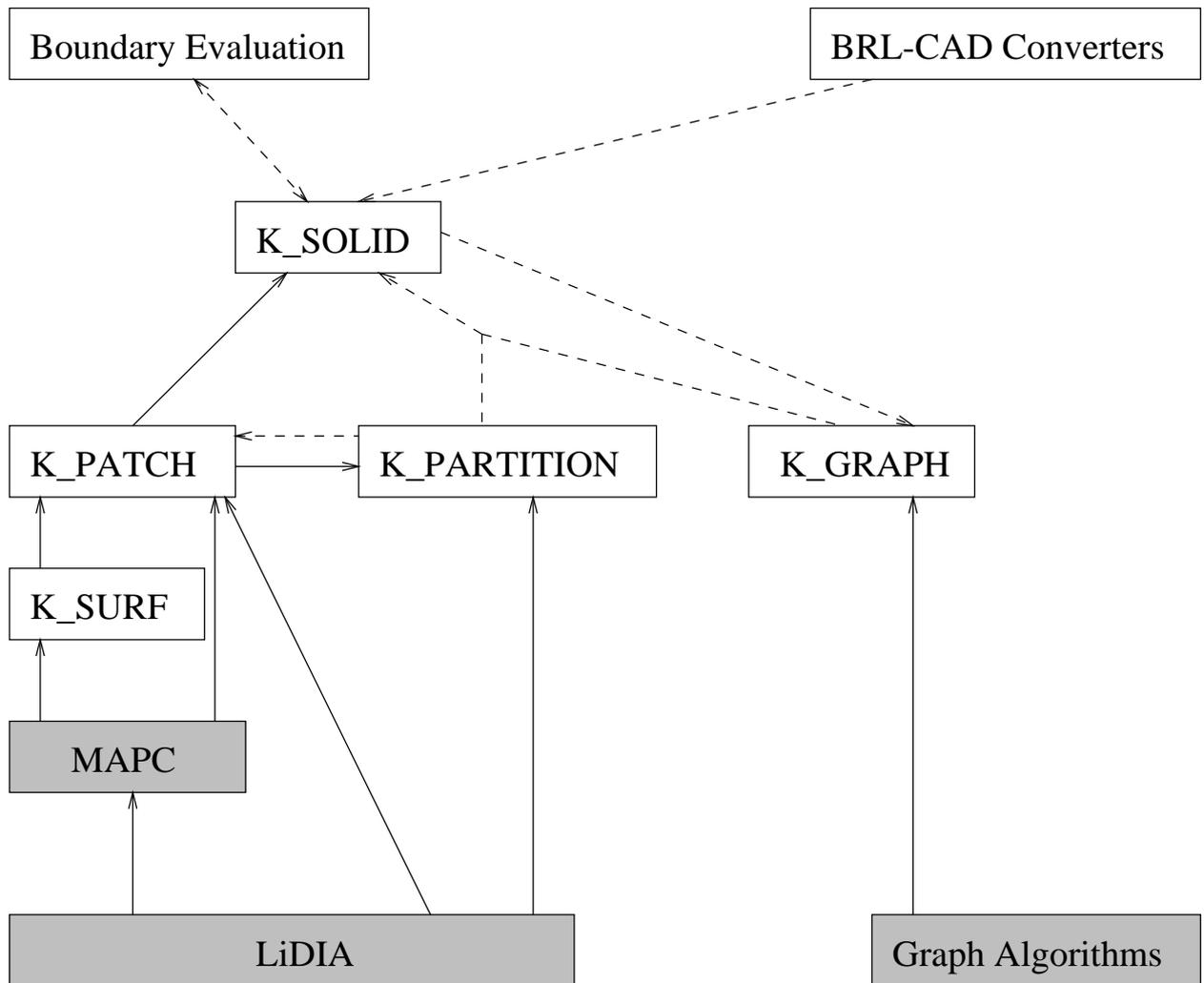


Figure 1: The major parts of ESOLID. Shaded boxes indicate external libraries used in ESOLID (including MAPC). A solid arrow indicates that one library or structure is a necessary part of another. A dashed arrow from one structure to another indicates that the source structure can be used to create the destination structure.

- **Generate an intersection curve** (in implicit form) in the domains of the patches by substituting the parametric representation of each patch into the implicit representation of the other patch. Each intersection curve is represented as the zero set of a bivariate polynomial.
 - **Resolve the topology** of the intersection curves (i.e. determine their structure in the patch domain). This involves finding intersections with the domain boundary, closed loops of the curve, etc.
 - **Intersect the intersection curve with the trimming boundary.** This involves intersecting curves in a 2D patch domain, then determining the position of each such intersection point in the domain of the other patch (*point inversion*).
 - **Determine the curve correspondence**, that is, how the individual portions of the algebraic plane curve in one patch domain relate to those in the other domain.
 - **Clip** the intersection curves in each domain so that only the portions inside the trimmed regions of both patches are maintained.
- For each patch:
 - **Merge** intersection curves from the separate patch/patch intersections to form patch/solid intersection curves.
 - **Partition the patch** into different components based on the trimming curves and merged intersection curves.
 - **Classify partitions** as to whether they are inside or outside of the other solid. This is done by classifying whether a point contained in one partition from each solid is inside or outside of the other solid. From there, inside/outside decisions can be made on the basis of topological relationships.
 - Based on the Boolean operation, choose the correct components from each solid to **build the final solid**, updating all topological information.

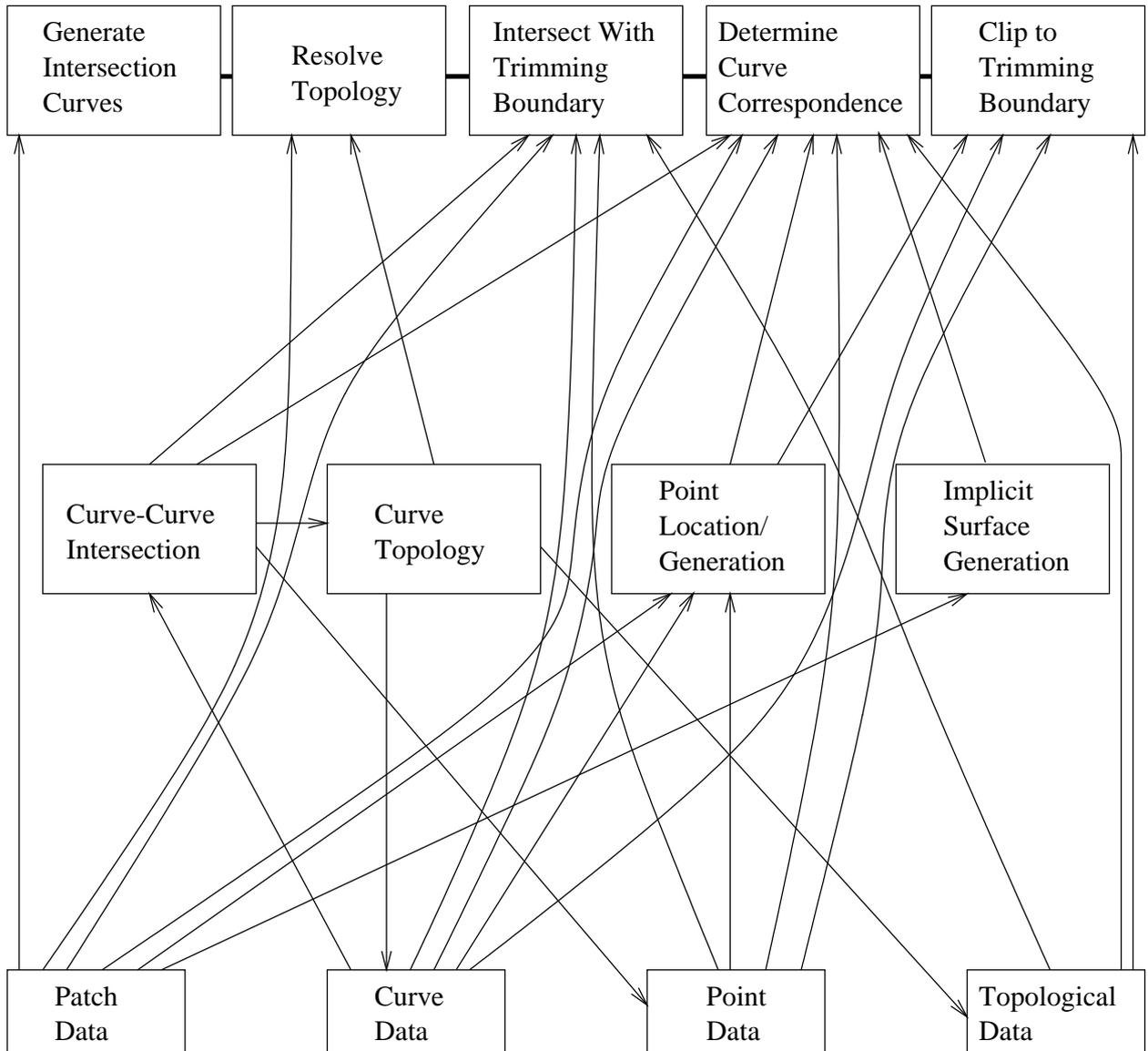


Figure 2: A summary of the five main steps in the first stage of the boundary evaluation algorithm. These operations are done for each pair of patches (one from the first solid, one from the second solid). Arrows show how the basic data and kernel operations are used in the various steps. At top are the steps in boundary evaluation, in the middle are the kernel operations, and at bottom are the data structures for the input solids.

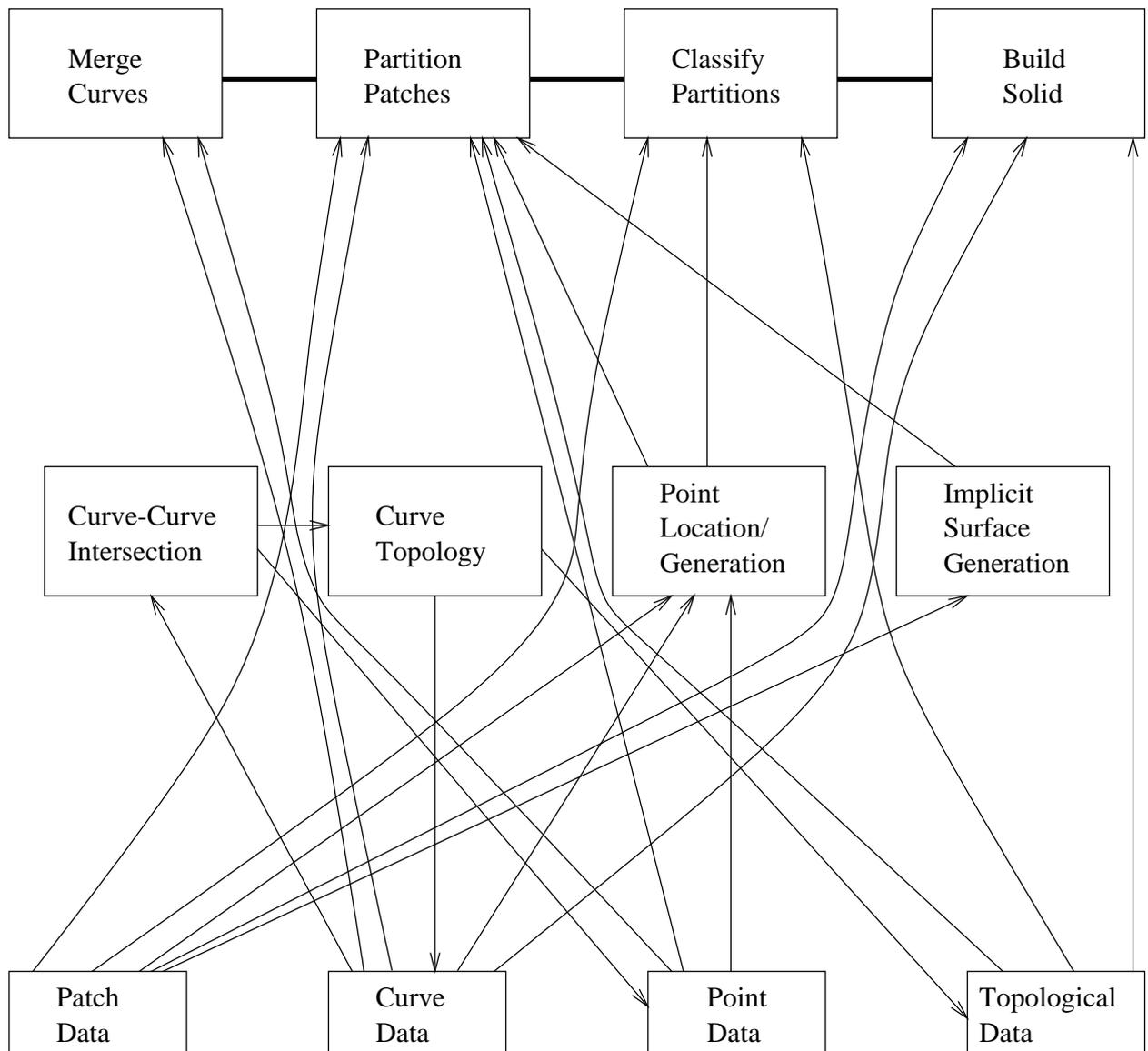


Figure 3: A summary of the four main steps in the second stage of the boundary evaluation algorithm. Arrows show how the basic data and kernel operations are used in the various steps. At top are the steps in boundary evaluation, in the middle are kernel operations, and at bottom are the data structures for the input solids.

These operations are built on a set of “kernel operations.” The efficiency of these kernel operations has a tremendous effect on the efficiency of the entire system. **Curve-curve intersection** and **curve topology** are a part of MAPC, and the new algorithms developed for them have been highlighted elsewhere [35]. **Point generation** refers to quickly generating a point with rational coordinates that lies on the surface of an object. **Point location** refers to classifying whether a 2D point lies inside or outside the trimmed region of a patch, or whether a 3D point lies inside or outside of another solid, and is accomplished through ray-shooting tests. **Implicit surface generation** refers to creating an implicit surface that meets certain criteria related to a specific parametric curve and/or patch. Figures 2 and 3 show the relationships between the kernel operations and the steps in boundary evaluation. Also shown in the figures is the way that the point (K_POINT), curve (K_CURVE), patch (K_PATCH), and topological data are used in the various kernel routines and steps of boundary evaluation.

3.3 Input Considerations

ESOLID was designed to handle data from real-world examples, meaning data not developed specifically to test exact boundary evaluation. The BRL-CAD [14, 13] data format was used as the model for ESOLID input. BRL-CAD is a CSG-based solid modeling system developed at the Army Research Lab and used for a variety of defense applications. Specifically, we focused on the Bradley Fighting Vehicle model provided to us courtesy of the Army Research Lab. It provided a large, complex, real-world example on which previous boundary evaluation attempts had proven difficult. While BRL-CAD supports a number of primitive CSG solids, most of them, including all those primitives used in the Bradley, are of low degree (surfaces no more than degree four), so we focused our efforts on handling such low-degree cases efficiently.

BRL-CAD represents all transformations as transformation matrices. Transformation matrices are the only method currently supported by ESOLID for specifying translations, rotations, etc. Note that transformation matrices can be input exactly, while other transformation descriptions, such as “rotation by X degrees,” might not have an exact representation using rational numbers.

While we view the use of transformation matrices as a restriction on the input allowed, we do not view it as an overly important problem because:

- The system we are using for our real-world input already uses this format for specifying transformations.
- Any other common transformation description (e.g. rotate by X degrees) can be expressed to within any desired tolerance as a transformation matrix with rational entries. We are currently exploring ways of automatically generating such approximations, thus allowing them to be used in an exact computation paradigm.

Although routines have been developed to convert BRL-CAD data files into the ESOLID input format, ESOLID is not limited to BRL-CAD data. Any data that can be expressed by the structures given in section 3.1 can be used in ESOLID. Specifically, solids must have a boundary representable as a set of parametric patches described as rational functions of polynomials with rational coefficients. Note, however, that only low-degree surfaces will yield reasonable running times. Surfaces such as bicubic Bezier patches and most NURBS surfaces have an implicit form that is of too high an algebraic degree to be practical in our current implementation. There are some other minor restrictions (e.g. the surfaces must be one-to-one mappings over the patch domain), however these are not significant for the most common CSG primitives. See [36] for conversion of several common CSG primitives to the ESOLID format.

By default, ESOLID will treat input as exact. However, as mentioned in section 1.2, the purpose of exactness is mainly to ensure consistency. Routines are included in ESOLID that allow a user the option of perturbing input data to achieve a particular interpretation. For example, the four vertices of a face of an input rectangular parallelepiped might not be coplanar, due to roundoff error in the input file. Options are provided to either treat the face as a bilinear patch (an “exact” interpretation), or to fit planes to each face then form new vertices at the intersections of the faces (the “perturbed” interpretation). As long as such interpretation is made *only* at the original input stage, and not in intermediate computations, the consistency provided by exact computation is

maintained.

Finally, ESOLID input is restricted to non-degenerate configurations. Although certain degeneracies are accounted for and handled within ESOLID, other degeneracies can cause ESOLID to fail. Many real-world examples (including several from the Bradley Fighting Vehicle) contain numerous degeneracies. ESOLID cannot be considered a robust system, in terms of handling all possible input configurations. However, ESOLID's elimination of numerical error increases robustness (over an inexact system), and since treating numerical error is an important prerequisite to fully handling degeneracies, ESOLID supports future treatment of degeneracies.

4 Challenges

A number of challenges were faced in the development of ESOLID. Among these were creating the necessary data structures and algorithms for exact computation and propagating information between the patches. A primary concern was efficiency, and this is discussed further in section 5. Other challenges, such as parts of algorithm design and development of certain new algorithms, have been presented previously [33, 34, 35].

4.1 Exact Data Structures and Algorithms

One major obstacle encountered in implementing ESOLID was the lack of existing library support for exact computation. While libraries exist for exact rational number computation, none were found for algebraic number computation, except general computer algebra systems. Because of their generality, these computer algebra systems do not provide the level of efficiency needed for boundary evaluation. Also, libraries providing geometric data structures tend to focus on linear structures (and occasionally circles). A more general library for representing curves exactly was not found.

We developed the MAPC library to meet this need. Although geared specifically to the boundary evaluation problem, the data structures and routines for polynomials, points, and 2D curves

that MAPC provides have been applied to other problems, as well [10, 49, 20]. As libraries are developed that support exact computation, one of the major hurdles to exact implementations (lack of library/compiler/hardware support) will gradually be lowered.

4.2 Transferring Data Between Patches

Many sub-algorithms used in boundary evaluation involve transferring the data from the patch of one solid to the other. Two major examples of this are point inversion (part of the intersecting curve step) and the curve correspondence step (see section 3.2).

In these cases, the most obvious and direct approach would be to treat the problem in 3 or more dimensions. This proves to be problematic, however. First, exact operations in higher dimensions are generally extremely slow. Second, and perhaps more fundamental, new data structures might be necessary to perform such computations. For example, the intersection curve between two patches is represented as a 2D curve in each patch domain—the algebraic space curve is not explicitly represented. We present algorithms for point inversion and curve correspondence based on the lower-dimensional representation.

While these algorithms are specific to the boundary evaluation problem, we believe that they are illustrative of the type of reasoning that can be used to improve efficiency in other exact geometric calculations. In each case, we take advantage of problem-specific information (such as known surface relationships) to reduce the complexity of the overall calculation significantly.

4.2.1 Point Inversion

A point in the domain of a patch P_1 determines, via the parameterization, a point \mathcal{P} in 3-space. If \mathcal{P} is in the intersection of P_1 with another patch P_2 , it may be necessary to find the inverse image of \mathcal{P} under the parameterization of P_2 . This process is *point inversion* and the inverse image is called the *inverted point*. Point inversion can be viewed as a problem in as many as seven dimensions (the two dimensions, s and t of P_1 , the two dimensions, u and v , of P_2 , and the three spatial dimensions

(x, y, z)). The problem is easily reduced to four dimensions:

$$F_x(s, t) = G_x(u, v)$$

$$F_y(s, t) = G_y(u, v)$$

$$F_z(s, t) = G_z(u, v)$$

where one wants to find a particular (u, v) interval (the *inverted point*) corresponding to a given (s, t) interval. This four dimensional operation is still too time-consuming to yield an efficient implementation. Fortunately, we can reduce the computation to a series of 2D and simple 3D calculations, as presented here:

In the domain of P_1 , \mathcal{P} is described as a particular intersection of two curves, $f(s, t) = 0$ and $g(s, t) = 0$. In boundary evaluation, f and g will always be either intersection or trimming curves. Thus, f is the intersection of a surface, $S_1(x, y, z) = 0$ with P_1 , and g is the intersection of a surface, $S_2(x, y, z) = 0$ with P_1 . In all cases where point inversion is necessary, either S_1 or S_2 (without loss of generality, say S_1) is the surface corresponding to the patch P_2 .

The intersection of P_1 with P_2 is already computed as a curve, $\tilde{f}(u, v) = 0$ in P_2 's domain. The intersection of S_2 with P_2 is now computed in the form $\tilde{g}(u, v) = 0$. Next, the intersections of the curves $\tilde{f} = 0$ and $\tilde{g} = 0$ are computed. This yields a set of points, p_1, p_2, \dots, p_n , in the domain of P_2 , one of which must be the inverted point.

Note that if S_1 or S_2 is self-intersecting (i.e. it does not have a one-to-one correspondence between the domain and the surface), then there may be more than one possible inverted point. We avoid such cases by always dividing primitive input solids into patches such that each patch has a one-to-one mapping over the patch domain. By decomposing solids appropriately, this is always possible for the patches of the common CSG primitives [36].

To this point, only 2D operations have been required. Very basic 3D operations are now used to determine which of the p_i is the inverted point. We find a 3D interval (in x, y, z space) bounding each p_i . This can be done by substituting the 2D interval bounding p_i into the parametric form

of the P_2 's surface, S_1 . Interval arithmetic operations determine the bounds for a 3D interval guaranteed to bound p_i . These intervals are compared to an interval bounding \mathcal{P} (generated from the 2D interval in P_1 's domain). Typically, only one p_i has an overlapping interval and this will be the inverted point. If two or more intervals overlap \mathcal{P} 's interval, the involved intervals can be reduced. This is done by reducing the interval surrounding each point in the 2D patch domain (a function provided in MAPC), constructing a new 3D interval from that 2D interval, and iterating until the ambiguity is resolved.

In this way, point inversion has been converted from a higher-dimensional problem into a series of 2D computations (curve-curve intersections), along with some simple 3D interval matching.

4.2.2 Curve Correspondence

Curve correspondence refers to finding the orientation of a curve in one patch domain relative to the same curve represented in the domain of another patch. Each algebraic plane curve has a direction induced on it in the domain of the patch. The algebraic plane curve (in the parameter space) is part of a curve in three spatial dimensions. This 3D curve (or a portion of it), represented in the domain of a different patch, may have either the same or an opposite orientation from the original algebraic plane curve. A common way to compute curve correspondence is to trace the curve in three dimensions. However, most tracing methods are approximate, and subject to numerical error. Furthermore, it is preferable to rely on only 2D operations, for efficiency reasons.

Note (see figure 2) that Curve correspondence occurs after intersection with trimming curves. The intersection with trimming curves step determines points on each curve, and inverts those points to the other patch domain. The curves that have either a point or inverted point on them are potentially corresponding curves (i.e. they may be part of the same 3D curve). Curves that do not have such points, with one easily-handled exception involving loops, are not relevant to the boundary evaluation problem, and do not need to be considered further.

Each of the remaining curves is guaranteed to contain at least 2 points for which a corresponding point is known on another curve in another domain. If there are three or more points (i.e.

the curve hits the trimming boundary in at least 3 places), then these three points can be used to determine a direction for the curve in both patches, and to verify that a particular portion of the curve in one patch domain (bounded by two of the points) corresponds to a given portion in the other. If there are only two intersections, an additional point on both curves can be generated to determine this information. While requiring several steps, this point generation and inversion is actually a simpler process in practice than finding the original intersection points and inversions. Thus, previously generated inverted point information (and possibly generating and inverting one more point) is used instead of curve tracing to transmit orientation from one patch domain to the other. There are a number of non-obvious details in this process, as well as a number of other simplifying assumptions that can be made. For space reasons, these are omitted here, but more details are given by Keyser [36].

5 Efficiency Considerations

Efficiency was a major concern in the implementation of ESOLID. The goal was to create an implementation that was one to two orders of magnitude slower than an inexact implementation (i.e. taking no more than 10–100 times as long) on real-world examples. In order to achieve this, a number of different speedup techniques had to be combined.

In order to understand certain speedups, we must mention Sturm sequences. Sturm sequences are used to count the number of real roots of a polynomial in an interval. Sturm sequences are obtained by performing a computation similar to a polynomial greatest common divisor to obtain a sequence of polynomials. These polynomials are then evaluated at various points to count the number of roots in an interval (see elsewhere for a more complete description [11]). Along with resultant calculations, they form the basis for the curve-curve intersection tests in MAPC, and play a major role in ESOLID's efficiency.

5.1 Speedups

Numerous speedup techniques were employed in ESOLID, and space permits only a brief mention of each type here. References are provided to prior uses of the techniques, though not necessarily in the way used in ESOLID. Note that all of these techniques are used while maintaining exactness. That is, even when approximate calculations are performed, guaranteed error bounds are used to ensure that any decisions made are correct.

- **Lazy evaluation** attempts to postpone high-precision (i.e. time-consuming) computations as long as possible in hope that they will not be necessary [2]. Lazy evaluation is applied to both point representations (reducing intervals surrounding algebraic coordinates only as needed) and curve representations (subdividing curves into segments) in the MAPC portion of ESOLID.
- **Quick rejection** techniques quickly identify cases where computation can be avoided entirely. *Interval arithmetic* [30, 28] can be used to avoid more complicated algebraic calculations involving curves and patches (e.g. determining whether a curve can intersect a particular patch boundary). *Affine arithmetic* [9], closely related to interval arithmetic, can speed up polynomial sign tests by providing tight error bounds and an efficient implementation for evaluating an interval in a polynomial. ESOLID uses interval arithmetic based on both exact rational interval bounds and IEEE floating-point bounds. The use of *bounding boxes* is another well-known technique for quick rejection, and is part of the point, curve, and patch representations in ESOLID. For example, patch bounding boxes are compared to eliminate cases where patches clearly do not intersect.
- **Simplified computation** refers to substituting fast, simple computations for more complex ones. As an example in ESOLID, *qualitative information* can be maintained with points to allow nearly instantaneous equality checking in certain cases, as opposed to the rather time-consuming exact algebraic number comparisons used otherwise. Algorithms can make use of *problem-specific information* to avoid more general, and thus more time-consuming,

computation. For example, curve-curve intersection is greatly simplified when the curves are horizontal or vertical. Algorithms developed for MAPC [35] use knowledge about the limited domain to perform more efficient curve-curve intersection tests and to determine curve topology. *Interval reduction* is used in ESOLID to speed computation of the bounds on polynomial roots. If an algebraic number is a simple root, its defining polynomial is negative on one side of the root and positive on the other (and which side is which is already known). This allows one to use a simple polynomial sign test (rather than a full Sturm sequence evaluation) to reduce the width of the interval. The polynomial sign tests themselves can be made faster through other speedup techniques.

- **Lower-dimensional formulation** of several parts of the computation also leads to great efficiency improvements. With exact computation especially, the higher the dimension of the problem, the longer the computation takes. It is often much faster to replace a single higher-dimensional computation by several lower-dimensional computations. Examples in ESOLID are point inversion and curve correspondence (see section 4.2), the overall boundary evaluation algorithm (all points, curves, and computations are in only two dimensions), and curve-curve intersection (2D computation replaced by a resultant and a series of 1D computations [35]).
- **Floating-point evaluations** are still very useful as a speedup technique, even though they might not be exact. *Floating-point filters* [18, 19] are a way of avoiding certain expensive exact computations by computing in floating-point hardware, but maintaining an error bound. If the error is small enough, a decision is made based on the result of the computation, but without exact arithmetic. This is used to avoid certain Sturm sequence calculations in ESOLID. Another technique, *floating-point guided computation* has proven even more useful for dealing with algebraic numbers. This refers to making a “guess” of a root using floating-point techniques (with no error bound), then using exact methods to verify that the guess was close enough to the real answer. For example, roots of a polynomial can be approximated us-

ing an imprecise method (e.g. Newton's method), then Sturm sequences can be used to verify that the right number of roots was found, and that a small interval around each approximate root contains the true root. Thus a tight, *guaranteed*, exact bound is generated faster than by standard interval bisection techniques. *Arbitrary-precision floating-point* computations can also be used. This means that floating-point numbers are represented using any number of bits. Although slower than standard, hardware supported, IEEE floating-point computations, such floating-point numbers can provide precision from the IEEE level (53 bits) all the way up to exact floating-point calculations. This allows floating-point filters with varying levels of accuracy. The PRECISE library [39] implements arbitrary-precision floating-point computation. It is an extension of the range arithmetic presented by Aberth and Schaefer and implemented in their range library [1]. ESOLID optionally includes PRECISE within MAPC as part of a filter for speeding up Sturm sequence computations.

5.2 Layering Speedups

ESOLID applies all of the speedups listed above into a multilayered approach. As an example, the process for reducing the size of an interval surrounding an algebraic root (in one dimension) will be described. This interval reduction computation is in turn part of more complex computations that incorporate more of the speedups listed above.

- The midpoint of the interval is determined (or another point is provided).
- If the root is simple, the defining polynomial will be positive on one side, negative on the other. The signs at the upper and lower interval bounds are known beforehand. Thus, only the sign at the midpoint needs to be determined:
 1. Apply a floating-point filter to try to determine the sign of the polynomial at the point.
 2. If that fails, evaluate the polynomial using exact computation.
- Otherwise, a Sturm sequence calculation at the midpoint must be performed:

1. Use floating-point filtered computation to attempt to evaluate the Sturm sequence.
2. If that is unsuccessful, use arbitrary-precision floating-point computation (PRECISE library) to determine and evaluate an approximate Sturm sequence.
3. If that also fails, determine polynomials for Sturm sequence exactly, and evaluate signs using a multilevel approach as described above.

5.3 Effectiveness of Combined Techniques

Many of the speedup techniques we use have been used individually in previous applications with good success. Combining techniques, however, does not necessarily combine the effectiveness of individual techniques. There are two reasons for this.

First, certain techniques tend to speed up the same cases. For example, the cases where exact affine arithmetic [9] is most effective are often the same as where floating-point filters are most effective. In most cases we have found, there is still a benefit to be realized by using both techniques, but in other cases, the increased overhead from applying a second method can actually reduce overall efficiency.

Second, some speedup techniques have different goals and tend to conflict. For example, lazy evaluation of point coordinates encourages intervals surrounding the point to be maintained as large as possible, shrinking them only as necessary. Floating-point guided computation, on the other hand, encourages intervals surrounding points to be reduced to the precision of the floating-point estimate. To find the best technique or (as in the case of shrinking intervals) a good balance between techniques takes experimentation and results may vary based on a particular type of input.

The important things to realize are that:

- Blindly adding speedup techniques does not provide the product of the improvement that the methods would provide on their own. In fact, some speedup techniques might not help at all.
- It might be necessary to balance the use of different methods. That is, using one technique to a point, then moving to a different one.

- An “ideal” balance and combination will depend on the particular data it is being tested on. It might be difficult to measure the exact tradeoffs and impossible to arrive at an optimum.
- Nevertheless, by combining these techniques and carefully choosing when to use each one, we have observed that they can provide *several orders of magnitude* improvement in speed over a straightforward exact approach.

6 Results

ESOLID has been applied to several test cases, both “synthetic” and “real-world.” Synthetic cases were created specifically to test or demonstrate the capabilities of ESOLID. Real-world cases were taken from a model developed in another solid modeling system (BRL-CAD [14, 13]) in order to determine the effectiveness of ESOLID on cases not specifically designed for ESOLID.

ESOLID provides the option of including the PRECISE library [39]. PRECISE is an extension of Aberth and Schaefer’s range arithmetic [1], based on arbitrary precision floating-point computation. It is used in ESOLID as a filter to speed up calculation of Sturm sequences, a key part of curve-curve intersection calculations as well as other calculations involving algebraic numbers. Except where noted, timings below do *not* include PRECISE.

All timings are in CPU seconds on a 300 MHz R12000 processor.

6.1 Synthetic Data

Figure 4 shows examples of simple Boolean combinations on basic primitives supported in ESOLID. This demonstrates some of the objects that ESOLID allows. Note that ESOLID can handle cases where objects have multiple components and genus greater than zero. Table 1 gives performance data for these basic cases. As can be seen, even for these basic examples, several curve-curve intersection tests may be performed, and the results may need to be found to high precisions.

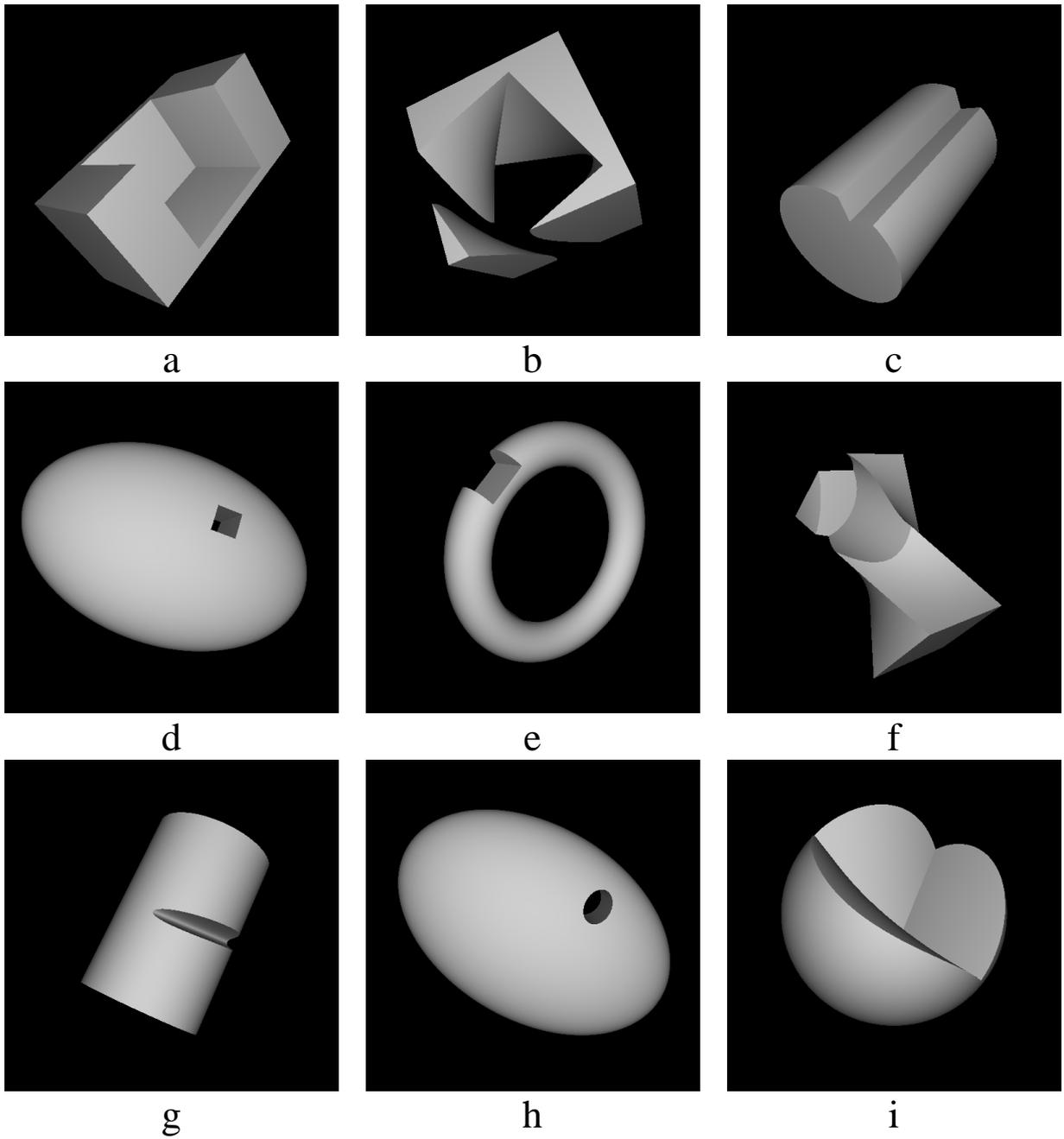


Figure 4: The result of Boolean operations on pairs of primitives in ESOLID. Details of the various operations are given in table 1.

<i>Example</i>	a	b	c	d	e	f	g	h	i
<i>Object 1</i>	box	box	cyl.	ell.	torus	twist	cyl.	ell.	ell.
<i>Object 2</i>	box	twist	box	box	box	cyl.	cyl.	cyl.	twist
<i>Degree of Object Surfaces</i>	1,1	1,2	2,1	2,1	4,1	2,2	2,2	2,2	2,2
<i>Number of Intersecting Patches</i>	6	12	6	8	8	8	4	8	9
<i>Maximum Degree of Intersection Curves</i>	1	2	3	2	4	6	6	6	4
<i>Number of Curve-Curve Intersections</i>	90	551	394	990	447	562	407	881	744
<i>Number of Univariate Roots Found</i>	0	240	353	1268	900	2004	607	2248	1753
<i>Bits of Precision in Algebraic Numbers</i>	-	10	13	59	87	52	31	87	25
<i>Total Time</i>	0.39	1.35	0.90	4.62	8.25	22.05	5.50	49.41	28.83

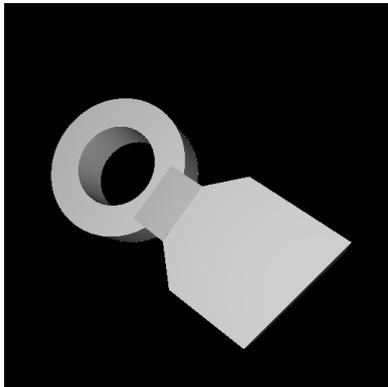
Table 1: Details of the difference operations illustrated in Figure 4. Object 1 describes the base primitive, while Object 2 describes the primitive being subtracted. The primitives shown are a box (polyhedron), twist (a box twisted so that some faces are bilinear patches), cylinder, ellipsoid, and torus. The degree of the surfaces in the two objects is given, followed by the number of pairs of patches that actually intersect. The maximum degree (in the parametric domain) of the intersection curves is also shown. The total number of curve-curve intersection operations performed is given, along with the total number of univariate roots found (i.e. the number of algebraic numbers found as a root of a univariate polynomial). The maximum number of bits of precision used to represent these algebraic numbers is given, followed by the total time taken to perform the Boolean operation.

6.2 Real-World Data

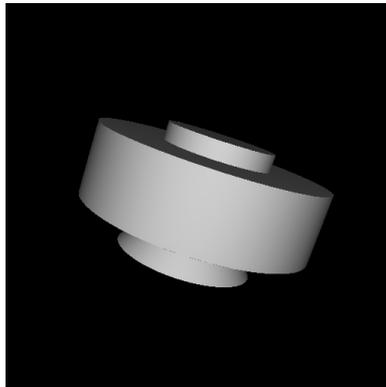
Real-world sample input was taken from the Bradley Fighting Vehicle model, provided courtesy of the Army Research Lab. This is a model created in the BRL-CAD system [14, 13], a CSG-based solid modeling system. The Bradley is composed of over 5000 solids. Primitives used are polyhedra (53%), generalized cones including cylinders (44%), ellipsoids including spheres (2%), and tori (1%). Although these primitives are low-degree, they have been combined to create a complex model. ESOLID was created specifically to handle the inputs provided by BRL-CAD, and includes conversion routines that were used to read in the BRL-CAD format and convert the primitives to the ESOLID format.

ESOLID was applied to several parts of the Bradley, some of which are shown in figure 5. These are meant as a representative sample of objects from the Bradley model. Timing data was taken both with and without inclusion of the PRECISE library. The same parts were also processed by the BOOLE system [38, 37]. The BOOLE system performs (inexact) boundary evaluation based on IEEE double-precision floating-point arithmetic. BOOLE uses tolerances to attempt to reduce the problems associated with numerical error. Table 2 gives the number of Booleans involved in each part, along with timing data for each system. Note that these parts sometimes include a grouping operation, which may appear as a union operation but does not require any arithmetic computation (i.e. solids are merged without concern for potential intersections). As is shown, for cases that BOOLE also worked on, ESOLID performs within two orders of magnitude in time. With the inclusion of PRECISE, these times are within about one order of magnitude. Note also that BOOLE is unable to handle several cases (see section 6.3).

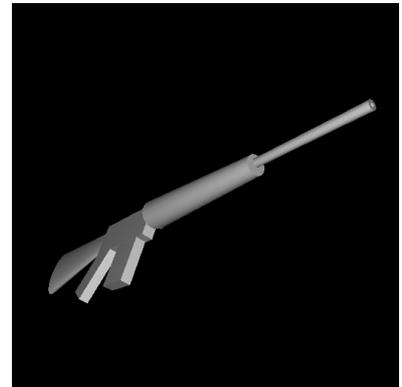
A breakdown of the individual timings under ESOLID (without PRECISE) are shown in table 3. Notice that curve-curve intersection computations are the dominant factor in the overall time. The two major parts of curve-curve intersection (as implemented in MAPC) are resultant computations and (univariate) Sturm sequence computations. As the table shows, the portion of curve-curve intersection time spent in each of these varies greatly. In general, it appears that for longer-running cases, the curve-curve intersections (specifically Sturm sequence calculations) take



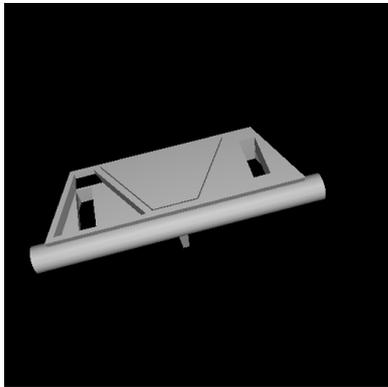
a



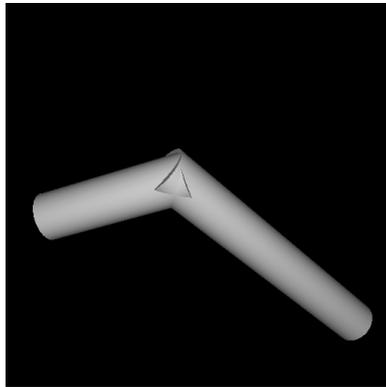
b



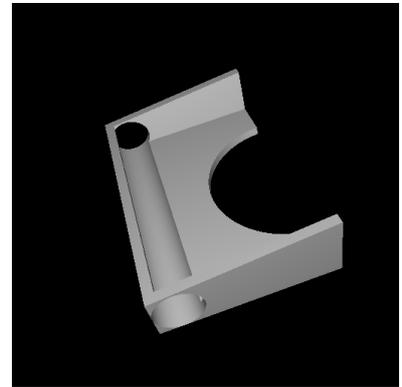
c



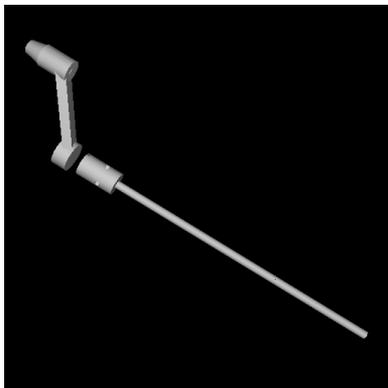
d



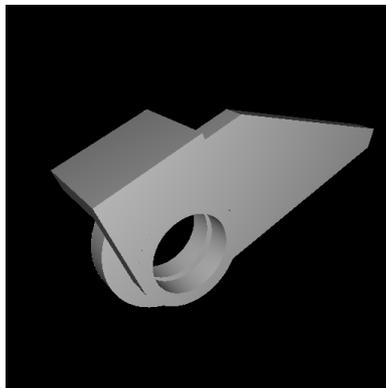
e



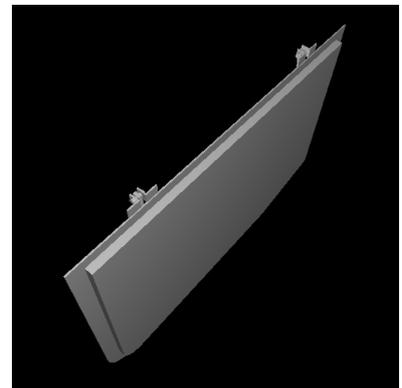
f



g



h



i

Figure 5: Example parts from the Bradley Fighting Vehicle model. Details of the models and timings are given in tables 2 and 3.

Example Number	Name	Number of Booleans	ESOLID Time w/out PRECISE	ESOLID Time w/ PRECISE	BOOLE Time
a	Tow Hook	2	10.23	10.95	2.23
b	Wheel Assembly	4	12.57	12.69	2.81
c	M16 Rifle	6	633.42	42.99	6.68
d	Track Link	11	132.48	137.64	27.74
e	Relay Mechanism	1	250.74	73.86	-
f	Launcher Mount Part	3	63.15	61.26	-
g	Support Assembly Part	6	213.72	105.99	-
h	Rear Hatch Hinge	7	58.92	63.48	-
i	Engine Access Hatch	16	54.78	58.44	-

Table 2: Overall timings for the examples in figure 5. The number of Boolean operations performed is shown, along with the times taken in ESOLID (both without and with the PRECISE library for arbitrary-precision floating-point filters of Sturm sequences) and in BOOLE (a boundary evaluation system based on double precision IEEE floating-point arithmetic and tolerances). A ‘-’ indicates that boundary evaluation failed on that object.

Example Number	Number of Curve-Curve	Number of Univariate Roots	Maximum Bits of Precision	Total Time	% of Time in Curve-Curve	% of Total Time in Resultant	% of Total Time in Sturm
a	425	1831	42	10.23	68.0	54.3	5.0
b	637	1106	59	12.57	54.2	46.8	1.9
c	1003	3834	57	633.42	98.2	3.6	94.3
d	4444	13511	75	132.48	74.9	64.6	3.7
e	320	6311	41	250.74	95.1	15.6	76.0
f	974	5227	65	63.15	81.7	63.6	13.2
g	1162	7116	66	213.72	92.5	35.8	54.7
h	1266	8191	87	58.92	69.1	57.4	5.3
i	1799	5334	69	54.78	64.2	55.0	3.8

Table 3: Timing breakdown under ESOLID, without PRECISE, for the examples in figure 5. The number of curve-curve intersections is given. The number of algebraic numbers found as roots of univariate polynomials is shown, along with the maximum number of bits of precision used to represent these algebraic numbers. The total time is shown, along with the percentage of time spent in curve-curve intersection, the major component of the boundary evaluation algorithm. The percentage of total time spent in the two major components of curve-curve intersection, resultant computations and Sturm computations (generation and evaluation of Sturm sequences), is also shown.

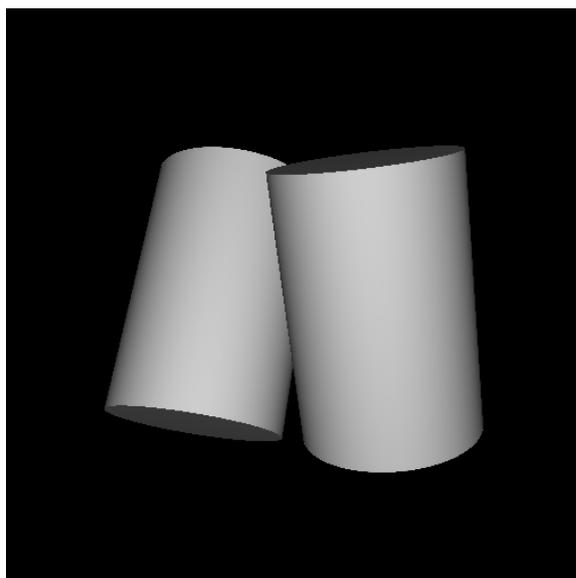
a higher percentage of the overall time. The range arithmetic (following Aberth and Schaefer's development [1]) included in the PRECISE library primarily speeds up Sturm sequence calculations, and achieves its best effects on cases where Sturm computations dominate the running time. Also notice that all cases use a high level of precision to isolate algebraic numbers. Due to the lazy evaluation procedures used in ESOLID, it is likely that levels of precision close to this would be required in order to guarantee accuracy. While a system that does not provide this level of precision may still work (e.g. BOOLE on cases a–d), it will be prone to failure.

6.3 Importance of Precision

The effectiveness of exact arithmetic in dealing with numerical errors can easily be demonstrated. Consider the synthetic example in table 1. Two cylinders barely interpenetrate. The table gives performance data for this case at varying levels of interpenetration. As can be seen in the table, depending on the depth of penetration, high levels of accuracy may be required in order to guarantee correctness. For some depths, it is impossible for standard floating-point data to provide the appropriate level of precision, since IEEE double-precision arithmetic provides at most 53 bits of precision, under the most ideal circumstances. In fact, IEEE floating point data could not even represent the input in most cases shown, as the depth of interpenetration is too small. While this is a synthetic case, and it is unlikely that any real-world example would be arranged like this, this case illustrates that ESOLID can correctly operate at these high levels of accuracy.

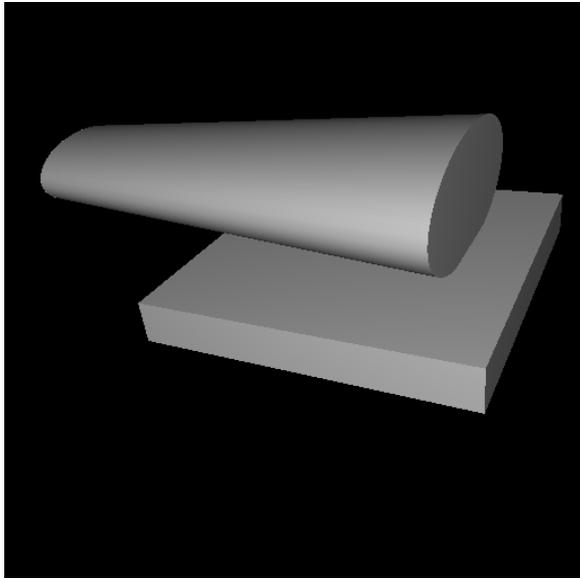
Accuracy in numerical calculations is also very important in real-world situations, as seen in the failure of the BOOLE system on certain example inputs. It is not always clear why BOOLE fails on specific examples. While numerical error is certainly a contributing factor [38], other failures may be due to more general programming bugs. Nevertheless, there are cases where the cause of failure is more clearly due to numerical problems.

Figure 6 shows two real-world examples where a fixed precision arithmetic based modeler can have problems. Example 6(a) shows one Boolean operation that is part of a larger model in the Bradley. A difference operation is performed, resulting in the solid shown in figure 6(b). ESOLID

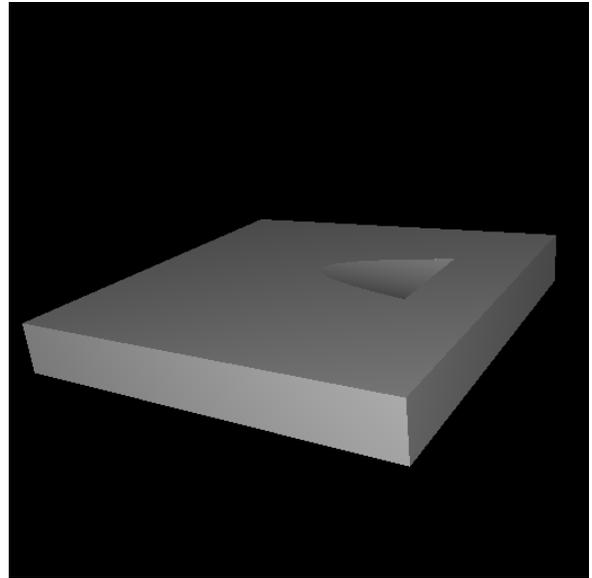


Depth of Penetration (10^{-x})	Precision Required (bits)	Total Time (s)	Sturm Time (s)	Resultant Time (s)
3	20	8.64	2.19	4.17
6	20	12.45	4.14	5.61
9	25	17.25	7.23	7.47
12	30	22.98	11.13	9.15
15	40	33.21	17.07	11.88
18	52	47.46	24.66	14.46
21	58	60.15	32.64	18.15
24	62	86.76	47.64	22.80
27	68	147.66	99.75	26.37
30	71	120.36	74.79	29.64
33	77	164.01	108.03	34.41
36	117	205.17	143.40	38.34
39	88	446.28	357.63	46.89
42	141	317.55	237.15	49.11
45	96	385.80	296.19	55.80

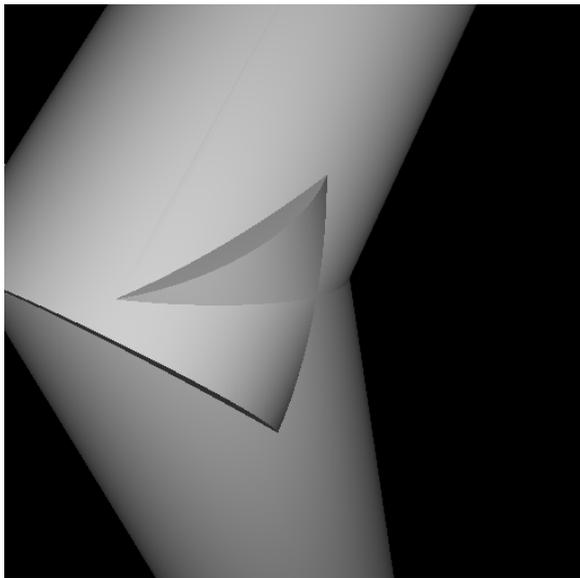
Table 4: Timing results for the example shown in the picture. The depth of penetration of the two cylinders is given in the first column. Following that is the maximum precision required in the boundary evaluation algorithm to represent the algebraic numbers exactly. n bits of precision required means that algebraic numbers were determined to an interval of width no smaller than 2^{-n} . The total time to perform boundary evaluation is listed, followed by the time spent in Sturm computations (both generation and evaluation of Sturm sequences), and in resultant computations.



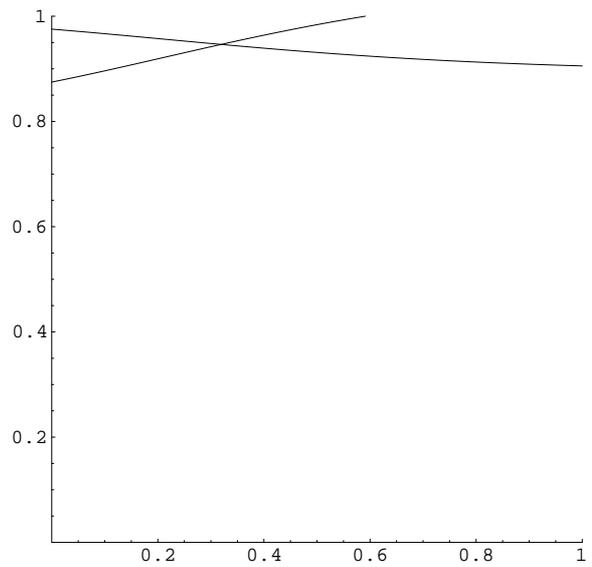
a



b



c



d

Figure 6: Close-up views of two operations where BOOLE fails.

correctly handles the case, while BOOLE fails. BOOLE’s failure in this case is reported as a “curves did not close” error, indicating a significant problem with the intersection curve computation. Although there are many possible reasons this could occur, it is clear that the two solids meet nearly tangentially. Near-tangential intersections are highly prone to numerical error, since a slight modification in either solid can have a major impact on the intersection curves between them. It can be surmised that such a problem led to BOOLE’s failure.

A more direct example is shown in figure 6(c). This close-up view of the Relay Mechanism (5(e)) shows two cylinders meeting in a nearly degenerate configuration. The intersection curve, shown in the domain of one patch in figure 6(d), even appears singular. In fact, this curve is not singular (it has two separate components) and ESOLID correctly resolves the topology of the curve. BOOLE, however, exits with an error that a singularity has been found. Clearly the exact computation of ESOLID allows an operation to be easily performed that would otherwise cause problems.

7 Conclusion

We have presented a description of the ESOLID system for performing exact boundary evaluation of curved solids. It has been applied to real-world examples, achieving times within one order of magnitude of the time spent by an inexact system on those cases. We have demonstrated that ESOLID can accurately evaluate a boundary in cases that are prone to numerical error in inexact systems. To our knowledge, no other exact system has achieved such results.

7.1 Implications for Further Development

ESOLID has demonstrated that exact boundary evaluation is possible with reasonable efficiency for low-degree curved solids. ESOLID was designed both as a proof-of-concept and as a system to allow various speedups and algorithms to be compared. We hope that showing that such an implementation is possible will spark further work in exact computation with curved solids, and

that knowledge gained from the implementation of ESOLID can be transferred to aid future systems. Although exact computation may still be too slow for many applications, it is reasonable to expect that with further research and development, the efficiency of exact computations can far exceed the level presented here. The speedups described in this paper have allowed an improvement of many orders of magnitude, and further research should yield significantly more improvement. For example, recent work by Rouillier and Zimmerman [44] might yield significant speedup in the curve-curve intersection routines, far beyond what has been obtained via MAPC's implementation. We have attempted to show that exact computation is a feasible approach for eliminating numerical error, is not as hopeless as common perception would believe, and should be a useful direction for further research.

7.2 Lessons Learned

The implementation of ESOLID led us to several observations about the implementation of exact systems. Among these were:

- **Designing Algorithms for Exact Computation:** It is tempting to just take an existing algorithm and substitute exact computations for the inexact ones. This implementation may be exact, but it is likely to be far less efficient than an algorithm for which exact arithmetic was assumed during design. When building an (efficient) exact system, exactness should be considered at *all* levels of algorithm design.

For example, in an inexact system, point coordinates may be represented as IEEE floating-point numbers, while an exact system would use some exact representation of algebraic numbers. Algorithms designed assuming inexact arithmetic would likely assume that comparing two points for equality would be very fast, as would be the case if their coordinates were just floating-point numbers. However, comparing two points represented exactly can potentially take a great deal of time. An algorithm designed with this consideration in mind may be able to reduce the number of point comparisons, and perform better on exact point representations.

- **Layering Speedups:** As mentioned in section 5, for efficiency, a wide variety of layered speedups must be used. Different speedup techniques may be appropriate at different algorithmic and conceptual levels of the program. Finding the appropriate places to use various techniques requires a significant amount of testing.
- **Testing:** Although exact computation is meant to eliminate numerical errors, exact computations are just as prone (if not more so) to programming errors as inexact ones. Since each exact routine is assumed to be reliable and no tolerances are used at later stages, thorough testing is important. While general-purpose computer algebra systems, such as Mathematica, might not be appropriate for the basis of a program, they can be extremely helpful in the testing and analysis stage.
- **Space Requirements:** Exact computations and representations tend to use tremendous amounts of memory. Although not a focus of the ESOLID work, the importance of memory management became apparent at later stages of development. ESOLID has the potential to take up very large amounts of memory (several megabytes on simple problems). Decisions made at earlier development stages, such as a lack of reference counting and storing intermediate results, made it difficult to implement more memory-efficient approaches at later stages. While this was not a problem for our development, memory consumption could easily become an issue for exact implementations on other systems. This is likely to be a problem for any exact implementation, since exact calculations tend to produce results that use far more memory (e.g. require additional data structures and arbitrary numbers of bits) than standard, hardware supported calculations. Such considerations need to be taken into account at the original stages of system design.
- **Redundant Information:** In a non-exact system, redundant topological or geometric information is a potential source for serious robustness problems. For example, storing both vertex positions and plane equations for polyhedra can yield inconsistencies (since the vertex might not lie exactly on the plane of an adjacent face). Thus, in inexact computations, it

is often wise to avoid redundant data. With exact computation, however, no inconsistencies will arise, so it is not necessary to constantly ensure that only a consistent set of data is used. This can result in much simpler programming and allows certain operations to be performed much more efficiently.

7.3 Future Work

There are several avenues open for future work extending from ESOLID. Among these are:

- **Higher-degree Surfaces:** Although ESOLID does not limit the degrees of input surfaces, higher-degree parametric surfaces, including most spline patches (e.g. bicubic Beziers), are still far too slow (more than 1–2 orders of magnitude difference) in our current implementation. While low-degree surfaces are sufficient for the standard CSG primitives, handling higher-degree surfaces would certainly be useful.
- **Degeneracies:** ESOLID is restricted in that it assumes that input will not be degenerate, preventing ESOLID from being considered fully robust. Degeneracies are a part of many real-world examples, including several from the Bradley Fighting Vehicle, and it would be useful to address them directly. Because numerical error can both create and remove degeneracies, exact computation is an important prerequisite to fully handling degeneracies. Thus, ESOLID can serve as a base for exploring new approaches to degeneracies.
- **Speedups:** Besides those listed, numerous other speedup techniques may be applied.
- **Memory Issues:** Besides time-efficiency, memory-efficient exact computations are a worthwhile subject for further study.
- **Extended I/O and Integration:** The current input and output capabilities of ESOLID, while useful for research purposes, could be expanded significantly. Of particular interest would be to identify ways to output or store intermediate data in such a way that the exact representation could be easily recovered.

- **Implicit Surfaces:** Our approach relies on the assumption that models are described by parametric patches. It would be useful to see whether a similar exact computation approach could be applied to solids described by implicit surfaces.

7.4 Acknowledgements

We would like to thank the Army Research Lab for access to the BRL-CAD system and the Bradley Fighting Vehicle model. We also thank the reviewers of the earlier Solid Modeling paper [31] for their helpful comments.

References

- [1] O. Aberth and M. J. Schaefer. Precise computation using range arithmetic, via c++. *ACM Transactions on Mathematical Software*, 18(4):481–491, December 1992.
- [2] M. O. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic: A detailed implementation. *Computer Aided Design*, 26(6):403–416, June 1994.
- [3] I. Biehl, J. Buchmann, and T. Papanikolaou. LiDIA: A library for computational number theory. Technical Report SFB 124-C1, Fachbereich Informatik, Universität des Saarlandes, 1995.
- [4] I. C. Braid. The synthesis of solids bounded by many faces. *Communications of the ACM*, 18(4), April 1975.
- [5] Hervé Brönnimann, Ioannis Emiris, Victor Pan, and Sylvain Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 174–182, 1997.

- [6] Smita Brunnermeier and Sheila Martin. Interoperability cost analysis of the u.s. automotive supply chain. Technical Report RTI Project Number 7007-03, Research Triangle Institute, 1999. <http://www.rti.org/publications>.
- [7] 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.
- [8] Kenneth L. Clarkson. Safe and effective determinant evaluation. Manuscript, February 1995.
- [9] Joao Luiz Dibl Comba and Jorge Stolfi. Affine arithmetic and its applications to computer graphics. *Anais do VII Sibgraphi*, 1993.
- [10] T. Culver, J. Keyser, and D. Manocha. Accurate computation of the medial axis of a polyhedron. In *Proceedings of the Symposium on Solid Modeling and Applications*, pages 179–190, 1999.
- [11] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra*. Academic Press, London, 1993. 2nd edition.
- [12] Helene Desaulniers and Neil F. Stewart. Semantic interpretation of inconsistent curve and surface data for the definition of solids. In Joe Warren, editor, *Curves and Surfaces in Computer Vision and Graphics III*, volume 1830 of *SPIE Proceedings*, pages 81–90. SPIE, Bellingham, WA, 1992.
- [13] Paul H. Dietz, Jr. William H. Mermagen, and Paul R. Stay. An integrated environment for army, navy and air force target description support. Technical Report Memorandum Report BRL-MR-3754, Ballistics Research Laboratory, Aberdeen Proving Ground, MD, 1989.
- [14] Phillip C. Dykstra and Michael John Muuss. The BRL-CAD package an overview. Technical report, Advanced Computer Systems Team, Ballistics Research Laboratory, Aberdeen Proving Ground, MD, 1989.
- [15] Gershon Elber. IRIT solid modeling system. <http://www.cs.technion.ac.il/~gershon/irit/>.

- [16] Shiaofen Fang, Beat Bruderlin, and Xiaohong Zhu. Robustness in solid modeling: A tolerance-based intuitionistic approach. *Computer Aided Design*, 25(9):567–576, September 1993.
- [17] Steven Fortune. Polyhedral modelling with multiprecision integer arithmetic. *Computer-Aided Design*, 29(2):123–133, 1997.
- [18] Steven Fortune and Christopher J. van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the 9th ACM Conference on Computational Geometry*, pages 163–171, 1993.
- [19] Steven Fortune and Christopher J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15(3):223–248, July 1996.
- [20] Nicola Geismann, Michael Hemmer, and Elmar Schömer. Computing a 3-dimensional cell in an arrangement of quadrics exactly and actually! In *Proceedings of the 17th ACM Conference on Computational Geometry*, pages 264–273, 2001.
- [21] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31–41, March 1989.
- [22] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 106–117, 1988.
- [23] Christoph M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.
- [24] Christoph M. Hoffmann, John E. Hopcroft, and Michael S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, November 1989.

- [25] C.-Y. Hu, N. M. Patrikalakis, and X. Ye. Robust interval solid modeling, Part I: representations. *CAD*, 28:807–818, 1996.
- [26] C.-Y. Hu, N. M. Patrikalakis, and X. Ye. Robust interval solid modeling, Part II: boundary evaluation. *CAD*, 28:819–830, 1996.
- [27] David J. Jackson. Boundary representation modelling with local tolerancing. In *Proceedings of the Symposium on Solid Modeling and Applications*, pages 247–253, 1995.
- [28] J. R. Johnson. Real algebraic number computation using interval arithmetic. In *Proceedings of ISSAC '92*, pages 195–205, 1992.
- [29] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proceedings of the 15th ACM Conference on Computational Geometry*, pages 351–359, 1999.
- [30] M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulations using rational arithmetic. *ACM Transactions on Graphics*, 10:71–91, 1991.
- [31] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. ESOLID: A system for exact boundary evaluation. In *Proceedings of the Symposium on Solid Modeling and Applications*, pages 23–34, 2002.
- [32] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. In *Proceedings of the Symposium on Solid Modeling and Applications*, pages 42–55, 1997.
- [33] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic I: Representations. *Computer Aided Geometric Design*, 16(9):841–859, October 1999.

- [34] 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*, 16(9):861–882, October 1999.
- [35] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer Aided Design*, 32(12):649–662, 2000. Special Issue on Robustness.
- [36] John C. Keyser. *Exact Boundary Evaluation for Curved Solids*. Ph.D. thesis, Department of Computer Science, University of North Carolina, Chapel Hill, 2000.
- [37] S. Krishnan, D. Manocha, M. Gopi, T. Culver, and J. Keyser. BOOLE: A boundary evaluation system for boolean combinations of sculptured solids. *International Journal of Computational Geometry and Applications*, 11(1):105–144, 2001.
- [38] Shankar Krishnan. *Efficient and Accurate Boundary Evaluation Algorithms for Boolean Combinations of Sculptured Solids*. PhD thesis, University of North Carolina, 1997.
- [39] Shankar Krishnan, Mark Foskey, Tim Culver, John Keyser, and Dinesh Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. UNC-CH technical report, Department of Computer Science, University of North Carolina, Chapel Hill, NC. A version of this paper has appeared in the 2001 Proceedings of the Symposium on Computational Geometry. The technical report has more complete citation, however, and can be downloaded from <http://www.cs.unc.edu/~geom/papers/techreport.shtml>.
- [40] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.
- [41] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 1999.

- [42] Aristides A. G. Requicha and Herbert B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, March 1982.
- [43] Aristides A. G. Requicha and Herbert B. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1):30–44, January 1985.
- [44] F. Rouillier and P. Zimmerman. Efficient isolation of polynomial roots. Tech Report RR-4113, INRIA, 2001.
- [45] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Comput. Graph.*, 24(4):105–114, August 1990.
- [46] Jonathan Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the 12th ACM Conference on Computational Geometry*, pages 141–150, 1996.
- [47] Kokichi Sugihara. A robust and consistent algorithm for intersecting convex polyhedra. In M. Dæhlen and L. Kjelldahl, editors, *Proceedings of EUROGRAPHICS '94*, volume 13, pages C–45–C–54. Blackwell Association, 1994.
- [48] Kokichi Sugihara and Masao Iri. A solid modeling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380–393, 1989.
- [49] Irina Voiculescu. Personal Communication, 1999.
- [50] Chee Yap and Thomas Dube. The exact computation paradigm. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 452–492. World Scientific Press, Singapore, 1995.
- [51] Norimasa Yoshida, Masato Shyokawa, and Fujio Yamaguchi. Solid modeling based on a new paradigm. *Comput. Graph. Forum*, 13(3):55–64, 1994. Proc. EUROGRAPHICS '94.

[52] J. Yu. *Exact Arithmetic Solid Modeling*. PhD thesis, Purdue University, CS Dept., West Lafayette, IN 47907, USA, December 1991.