International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

BOOLE: A Boundary Evaluation System for Boolean Combinations of Sculptured Solids*

S. Krishnan
AT&T Research Labs
180 Park Avenue, Room E-201
Florham Park, NJ 07932
email: krishnas@research.att.com

and

D. Manocha, M. Gopi, T. Culver, J. Keyser
Dept. of Computer Science
CB# 3175 Sitterson Hall
Univ. of North Carolina
Chapel Hill, NC 27599-3175
email: {dm,gopi,culver,keyser}@cs.unc.edu

ABSTRACT

In this paper we describe a system, BOOLE, that generates the boundary representations (B-reps) of solids given as a CSG expression in the form of *trimmed* Bézier patches. The system makes use of techniques from computational geometry, numerical linear algebra and symbolic computation to generate the B-reps. Given two solids, the system first computes the intersection curve between the two solids using our surface intersection algorithm. Using the topological information of each solid, it computes various components within each solid generated by the intersection curve and their connectivity. The *component classification* step is performed by ray-shooting. Depending on the Boolean operation performed, appropriate components are put together to obtain the final solid. We also present techniques to parallelize this system on shared memory multiprocessor machines. The system has been successfully used to generate B-reps for a number of large industrial models including parts of a notional submarine storage and handling room (courtesy - Electric Boat Inc.) and Bradley fighting vehicle (courtesy - Army Research Labs). Each of these models is composed of over 8000 Boolean operations and is represented using over 50,000 trimmed Bézier patches. Our exact representation of the intersection curve and use of stable numerical algorithms facilitate an accurate boundary evaluation at every Boolean set operation and generation of topologically consistent solids.

Keywords: solid modeling, constructive solid geometry (CSG), boundary evaluation, surface intersection, trimmed parametric surface, ray shooting

*Supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract DAAH04-96-1-0257, NSF Grant CCR-9319957, NSF Career Award CCR-9625217, ONR Young Investigator Award (N00014-97-1-0631), Intel Corp., DARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization

1. Introduction

The field of solid modeling deals with the design and representation of physical objects. One of its main emphases has been on the consistency of models generated. Boolean operations, such as regularized unions, intersections and differences, on solids play a fundamental role in solid modeling. They are used in various applications in mechanical engineering, computer graphics, robotics and computer vision. The two major representation schemata used in solid modeling are constructive solid geometry (CSG) and boundary representations (B-rep). B-reps describe solids as a set of vertices, edges, and faces with topological relations among them. In contrast, CSG considers solids as expressions of Boolean operations and rigid motion transformations of primitive solids which typically include polyhedra, spheres, cylinders, cones, tori and surfaces of revolution. Both these representations have different inherent strengths and weaknesses, and for most applications both are desired. For instance, a CSG object is always valid in the sense that its surface is closed, orientable and encloses a volume, provided the primitives are valid in this sense. A B-rep object, on the other hand, is easily rendered on a graphic display system and is useful for visual feedback in solid design. Figure 1 shows the model of a notional submarine storage and handling room that we obtained from Electric Boat, a division of General Dynamics. This model consists of more than 5000 solids, each designed using Boolean operations. The primitives used to generate these models vary from simple polyhedral objects, spheres and cylinders to fairly complex ones like generalized prisms, surfaces of revolution and offset surfaces. Figure 2 is a model of a real Bradley fighting vehicle from Army Research Labs. This model has over 8500 solids generated entirely using Boolean operations as well. Generating the B-reps of such large CAD models is necessary for applications like interactive visualization and model verification. Another application where B-reps are required is in collision detection for dynamic simulation of machine parts. For example, consider the track of the Bradley shown in Figure 21. The toothed circular structure shown in the left hand side of the image is the drivewheel. It is placed in the track in such a way that when it rotates without slippage, the Bradley vehicle moves forward. Placement of the drivewheel is very critical to obtain this effect. Dynamic simulations are performed to study the model placement. To simulate these realistically, we require algorithms that can perform interference detection. B-reps are necessary for this purpose.

Earlier, most B-rep modelers were able to support solids composed of polyhedral models and quadric surfaces (like spheres, cylinders etc.) and their Boolean combinations only. Over the last few years, modeling using free-form surfaces (*sculptured models*) has become very useful throughout the commercial CAD/CAM/CAE industry. On the research front, there has been considerable effort in integrating geometric and solid modeling[37,35,13,76,40,22,21]. In particular, there is a lot of interest in building complete solid representations from spline surfaces and their Boolean combinations[31,63,11,9,79,64,12,57]. However, the major bottleneck is in performing robust, efficient and accurate Boolean operations on the sculptured models. The topology of a surface patch becomes quite complicated when a number of Boolean

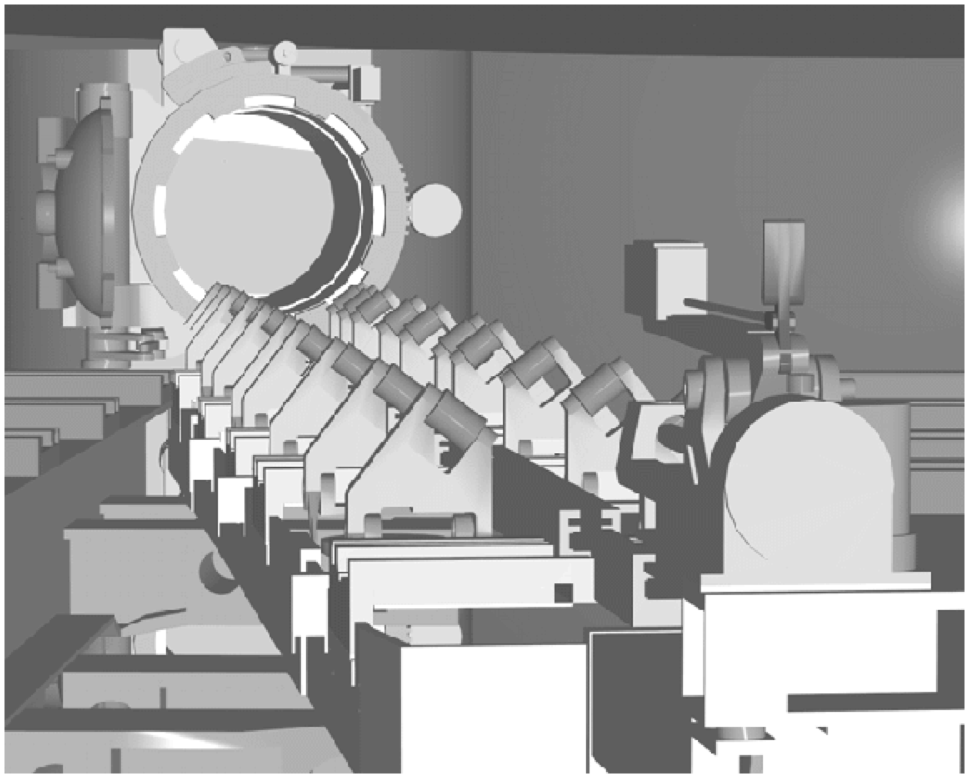

Fig. 1. Submarine storage and handling room

operations are performed and finding a convenient representation for these topologies has been a major challenge.

In many applications involving CAD/CAM, solids are designed in terms of tensor product trimmed Non-Uniform Rational B-Spline (NURBS) surfaces. This class includes a number of rational parametric surfaces like tensor-product and triangular Bézier patches. The representation capability of these surfaces is quite large, and is sufficient to represent all primitive solids encountered in boundary evaluation systems. Due to the difficulty in performing free-form surface intersection, many earlier B-rep modelers used high-resolution polyhedral approximations to these surfaces and apply existing algorithms to design and manipulate these polyhedral objects. Apart from the fact that the resulting solids are inaccurate, there is an additional cost in terms of increased memory usage due to data proliferation. The system presented in this paper provides effective strategies to perform Boolean operations on sculptured solids without resorting to polyhedralization.

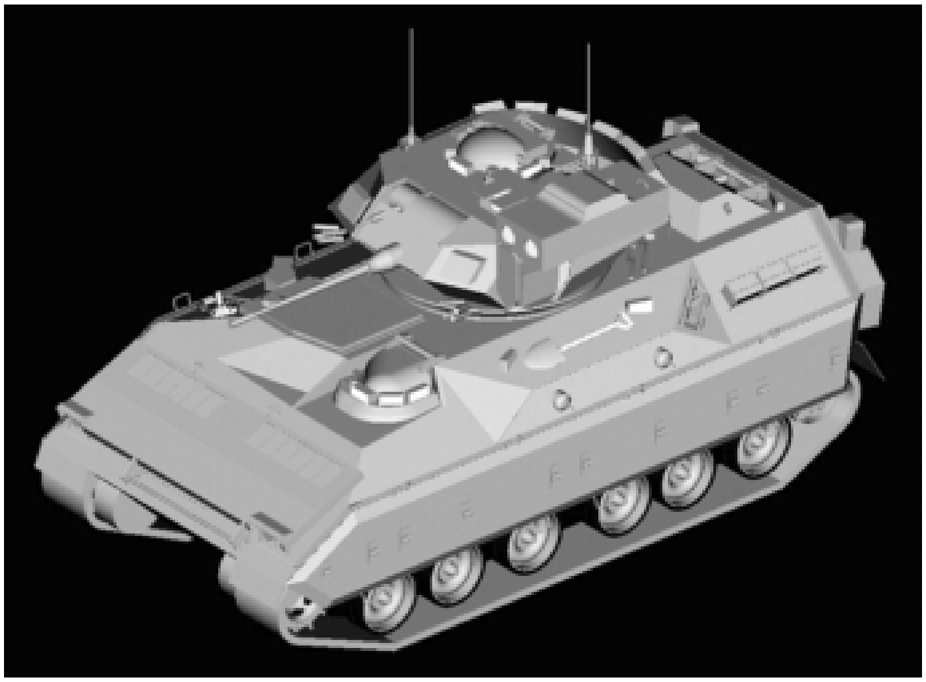

Fig. 2. Exterior of a Bradley fighting vehicle

1.1. Main Contributions

We present a system for computing the boundary of boolean combinations of sculptured solids. The resulting boundary is represented in terms of trimmed Bézier patches. Given two primitives, our system performs surface-surface intersections, curve-merging and component classification. To speed up these computations, we also parallelize and distribute them among multiple processors. The main contributions in this paper are:

- **Complete system:** BOOLE is a B-rep modeling system that converts solids represented in CSG form to its boundary representation. BOOLE provides data structures to represent a variety of primitive solids and an implementation of a number of geometric algorithms to manipulate them efficiently.

- **Integration of numeric, symbolic and geometric algorithms:** Our algorithm for boundary evaluation uses a number of symbolic, numeric and geometric techniques for efficient and accurate computation. The integration of these algorithms into one big system is another major contribution.

- **Accuracy:** Each intermediate primitive and the resulting B-rep solid is represented as a collection of trimmed Bézier patches and an adjacency graph. The trimming curves are the result of accurate surface intersection computation. The resulting B-reps are guaranteed to be manifold.

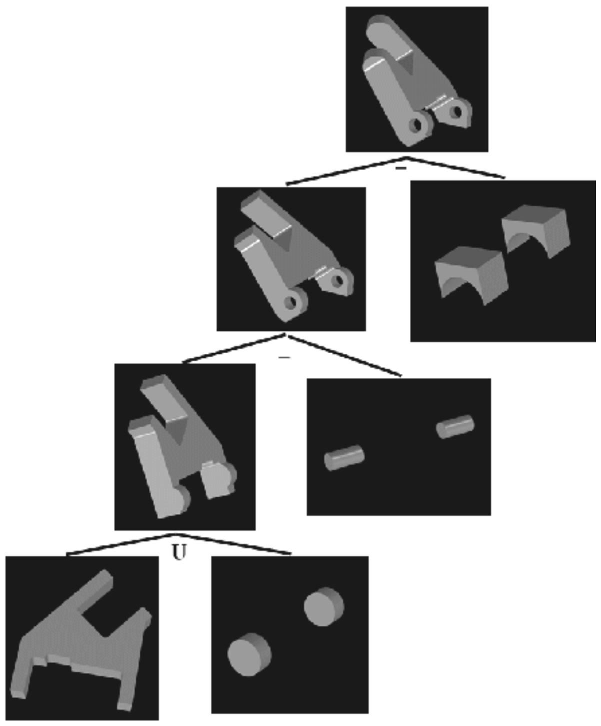

Fig. 3. Part of a CSG tree of the roller from the submarine model

- **Parallel Implementation with Load Balancing:** Our algorithm exploits parallelism at all stages of boundary computation. Our system can work on any shared memory parallel system. We also present algorithms to perform load balancing with minimum use of locking while parallelizing the boundary computation algorithm among various processors.

Organization: The rest of the paper is organized in the following manner. A brief discussion of previous work in the area of boundary computation and B-rep solid modeling systems is given in Section 2. Some basic definitions and terminology used in the solid modeling literature are described in Section 3. The representation of each solid in our system is explained in Section 4. Section 5 briefly describes our B-rep generation algorithm. Implementation issues that went into the design of BOOLE are described in Section 6. Section 7 discusses the architecture of BOOLE. We present some techniques to improve the robustness and accuracy of our system in Section 8. Section 9 talks about the degeneracies that we encountered and how they are handled by BOOLE. Section 10 describes our load balancing scheme for the parallel implementation. Section 11 shows the performance of our system on some models and the speed-up achieved due to parallelism. Section 12 talks about the public domain release and system interface of BOOLE. We conclude in Section 13.

2. Previous Work

The need to generate accurate boundary representations of solid objects in many applications involving design and manufacturing has generated significant interest in the research community. Over the years, the body of literature addressing these problems has grown to be quite extensive. Some of the earliest work in generating B-reps was done on polyhedral solids. The need to use free-form surfaces to represent solids has led to research in the problems of curve-surface and surface-surface intersection and loop detection which are important for B-rep generation.

Polyhedral solids: Algorithms for performing Boolean operations on polyhedra in B-rep have been proposed by a number of researchers[7,29,52,60,77,80]. Most of these techniques rely heavily on the algebraic formulation of the problem. Cameron[8] considers several strategies and redundancy tests to propagate approximations of CSG primitives from the root of the CSG tree down to the leaves, and possibly refining them on the way. Rossignac and Voelcker[67] consider redundancy determination without approximating the primitives. They define certain *active zones* on solids and show how knowledge of active zones can be used to improve conversion from CSG to B-rep, detection of redundancy and other operations on CSG trees.

The use of topological structures of solids has been very popular in B-rep solid modeling. The winged-edge style of boundary representation is due to Baumgart[5]. Many variants of the method, and other alternatives, have been proposed and used in B-rep modeling systems since then. A complete survey of topological structures in solid modeling is given in Ref. [78]. The use of non-manifold boundary representations was first proposed by Wesley[80]. Weiler[79,78] observed that a number of geometric operations on polyhedra simplify when non-manifold structures are permitted. Paoluzzi et. al.[61] implement Boolean operations on B-rep solids by using only triangular faces for their polyhedra. Laidlaw et. al.[48] describes another method in which all faces must be convex polygons, and suggest random perturbations to eliminate complex vertex intersection cases.

A number of approaches have been proposed for robust and accurate B-rep computation in polyhedral modelers. One of the most common approaches is based on using *tolerances* with floating-point arithmetic[34]. However, it is hard to decide a global tolerance value for all computations. To circumvent these problems, combinations of symbolic reasoning[30] and adaptive tolerances[71] have been proposed. Other algorithms include those based on redundancy elimination[18]. Many algorithms based on *exact arithmetic* have been proposed for reliable numeric computation for polyhedra[74,23,6,31].

Sculptured Solids: The idea of using free-form surfaces in solid modeling was introduced by Chiyokura et. al[13]. It describes the implementation of a system called *Designbase* with some curved-surface capabilities. In this system, curved solids are designed and modified by local operations such as altering the shape of certain edges and faces. However, Boolean operations require that one of the intersecting objects be polyhedral. Geisow[25] maps surface intersection curves to the plane and uses subdivision methods to solve surface interrogation problems. Requicha and Voelcker[65] describes the PADL system developed at University of

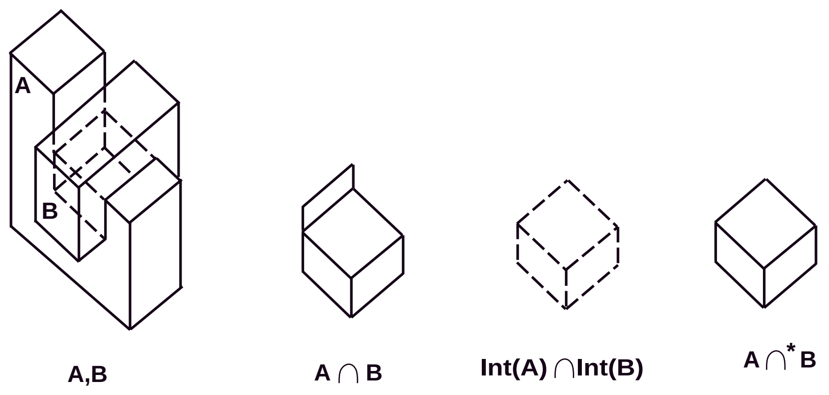

Figure 4: An example of regularized intersection operation [Hof89]

Rochester. This system supports Boolean operations on polyhedral solids and a few curved primitives. Casale et. al.[11,9,10] use trimmed parametric surfaces to generate B-reps of sculptured solids. The algorithm uses subdivision methods to evaluate surface intersections, and represents the trimming boundary with piecewise linear segments. Chan[12] uses special properties of quadric surfaces and other free-form surface to design industrial parts. A number of techniques like interval arithmetic and shell representations[76,40,75,69,57,16] have been developed to perform solid design with free-form geometries. Sorting points along intersection curves[36] was used to classify components with respect to solids.

The Alpha_1 CAD system developed at the University of Utah has many features to combine solids composed of sculptured surfaces. A systematic approach for design, analysis and illustration of assemblies has been presented in Ref. [15,66]. Ray representations along with specialized parallel architectures[56,17,57] like the Ray-Casting engine and 'Solids engine' were used to achieve interactive solid modeling on low-degree primitives like quadrics. Mantyla and Ranta[54] describe methods to perform solid modeling using HutDesign. Rossignac et. al.[66] present algorithms for inspection of cross-sections and interference between solids with bounded degree and limited height of CSG trees.

Most of the recent work in the literature on Boolean combinations of curved models has focussed on computing the intersection curve between a pair of B-spline surfaces[38,70,59,33,50,41,3,4,46].

3. Background Material and Definitions

In this section, we will briefly describe some terms used in the solid modeling literature and also give a mathematical introduction to *trimmed* parametric surfaces.

3.1. Regularized Boolean Operations

Boolean operations, such as *regularized* unions, intersections and differences, on solids play a fundamental role in solid modeling. Algorithms for determining the

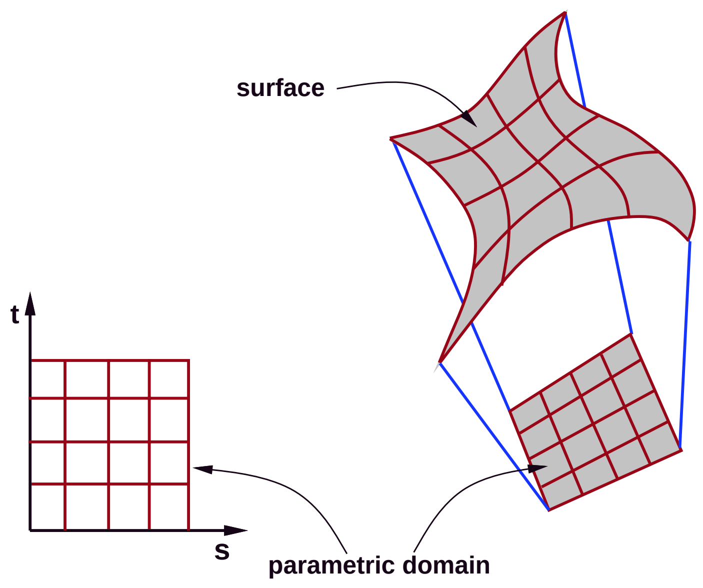

Figure 5: A surface patch and its parametric domain

regularized union, intersection, or set difference of two solids is a useful component of most B-rep modelers. The *regularized* operations differ from their corresponding set-theoretic counterparts in that the result is the *closure* of the operation on the interior (mathematically speaking, this refers to all of the solid except its boundary) of the two solids, and are used for eliminating "dangling" lower-dimensional structures (see Figure 4). If $Int(A)$ represents the interior of solid A and op corresponds to one of the set operations, we define op^*, the regularized version of the Boolean operation as

$$A \; op^* \; B \;=\; cl(Int(A) \; op \; Int(B))$$

where $cl(A)$ denotes the closure (generates boundary) of A. These operations can be used to convert solids represented by CSG trees (see Figure 3) to an equivalent B-rep. These processes for performing Boolean operations on B-reps are called *boundary evaluation* algorithms.

3.2. Curve and Surface Representation

Most algebraic curves and surfaces in 3D space can be represented using their implicit form, $f(x, y, z) = 0$. Geometric modeling applications frequently involve computing a set of points on a given curve or surface. But the process of computing points on surfaces with implicit representation is computationally intensive. An alternative representation is the parametric form. For example, a parametric space curve is a mapping from the real line to $\mathcal{R}^3$. The domain of these functions is also called the *parameter* of the curve. By substituting different values for the parameter, we obtain different points on the curve.

A NURBS curve [20] is a special kind of parametric curve. This curve is completely

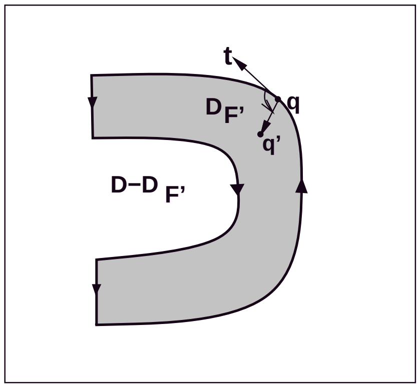

Figure 6: Trimming rule

specified by a set of points in space and a few smooth functions. These points are called the *control points* of the NURBS curve. The pre-specified functions are called the *basis or blending functions*. The control points and the blending functions are combined mathematically to give rise to a single curve.

The NURBS curve is composed of a number of segments or *spans*. In the parametric domain, these spans are described by a *knot vector*, which is basically a non-decreasing sequence of parameter values. The knot vector determines the region of influence a particular basis function has on the curve. A NURBS polynomial is defined as a linear combination of basis functions. When the coefficients of the linear combination expression are 4-tuples, the set of four implied polynomials form a curve. Each 4-tuple is a homogeneous representation of a control point in projective 3-space, and the homogenizing variable (4^{th} coordinate) is called a *weight*. We assume that the weights are non-negative. Essentially, this assumption ensures that the curve or surface is completely contained within the convex hull of the control points. This is not a major restriction because most curves and surfaces occurring in CAD applications can be represented using non-negative weights.

We shall represent control points in homogeneous coordinates $(\mathbf{v_i}, w_i)$, where $\mathbf{v_i} = (w_i x_i, w_i y_i, w_i z_i)$ and w_i is the weight. Therefore the parametric curve $f(t)$ of degree $k-1$ with n control points and the standard basis functions $\mathcal{N}_{i,k}$ is given by

$$f(t) = \frac{\sum_{i=0}^{n} \mathbf{v_i} \mathcal{N}_{i,k}(t)}{\sum_{i=0}^{n} w_i \mathcal{N}_{i,k}(t)}$$

$\mathcal{N}_{i,k}(t)$ is defined recursively over the knot interval $[t_i, t_{i+k}]$ as follows:

$$\mathcal{N}_{i,1}(t) = \begin{cases} 1, & if\ t_i \le t \le t_{i+1} \\ 0, & otherwise \end{cases}$$
$$\mathcal{N}_{i,k}(t) = \frac{(t_{i+k} - t)\mathcal{N}_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}} + \frac{(t - t_i)\mathcal{N}_{i,k-1}(t)}{t_{i+k-1} - t_i}$$

Based on the above formulation of the parametric curve, it is clear that the control points determine the shape of the curve. Further, since each control point has only a limited range of influence, it is very easy to shape the curve (or surface) by local modification of the control points. In most parametric specifications, the

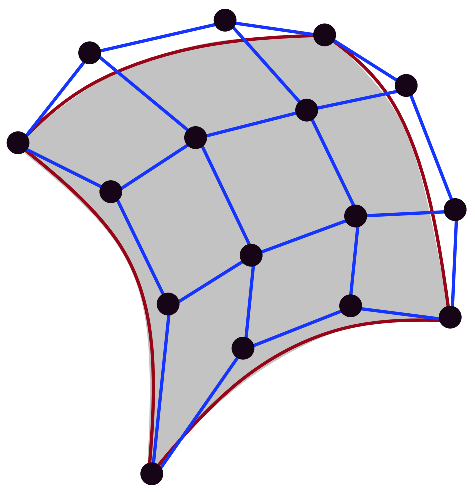

Figure 7: Sixteen control points of a bicubic Bezier patch

domain is normalized to lie in the unit interval, $t \in [0,1]$, without loss of generality.

A *tensor product* NURBS surface is defined over a two dimensional parametric domain over the parameters $0 \leq s,t \leq 1$. The shape of the surface is determined by two array of knot vectors (one for each parameter) and a two dimensional array of control points [20]. Figure 5 shows the relationship between a surface patch and its parametric domain. The weighted sum formulation of a NURBS surface is:

$$\mathbf{F}(s,t) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{v_{ij}}\mathcal{N}_{i,k}(s)\mathcal{N}_{j,l}(t)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}\mathcal{N}_{i,k}(s)\mathcal{N}_{j,l}(t)}$$

In this equation, the surface is of degree $k-1$ in s and $l-1$ in t (degree $(k-1)\times(l-1)$, for short). A *trimmed* NURBS surface, $\mathbf{F}^{'}(s,t)$, is a subset of $\mathbf{F}(s,t)$ defined by a set of trimming curves. A trimming curve is a simple, closed, piecewise sequence of curves (linear, NURBS or algebraic) defined in the domain, $\mathcal{D} = [0,1]\times[0,1]$, of $\mathbf{F}(s,t)$. The subset of the domain that is part of the trimmed surface is usually given by an unambiguous rule. For consistency, we shall define a rule that we follow for algorithmic description and implementation purposes.

- The trimming curve is oriented counterclockwise when looking into the plane of the paper from above (see Figure 6). More precisely, the simply closed trimming curve is homeomorphic to a circle which is oriented counterclockwise.

- The curve retains the part of the surface domain immediately to the left of it. Consider a point q on the curve and a domain point $q^{'}$ arbitrarily close to q (see Figure 6). Let the tangent at q be $\vec{\mathbf{t}}$. Then $q^{'} \in \mathcal{D}_{F'}$ is a part of the trimmed surface if the counterclockwise angle between $\vec{\mathbf{t}}$ and $\vec{qq'}$ is less than π.

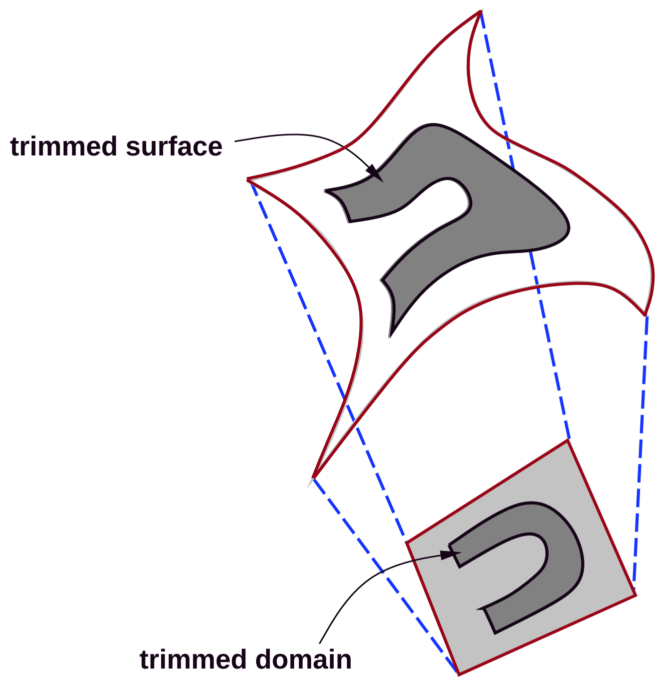

Figure 8: A trimmed surface patch

- Two points q_1 and q_2 belong to the same trimmed region ($\mathcal{D}_{F'}$ or $\mathcal{D} - \mathcal{D}_{F'}$) if and only if the line segment q_1q_2 intersects the trimming curve even number of times.

Therefore,

$$\mathbf{F}^{'}(s,t) = \{\mathbf{F}(s,t) \mid (s,t) \in \mathcal{D}_{F'}\}$$

Bézier surfaces are special types of NURBS surfaces, that do not have any knots except at the corner points (i.e., $(s,t) = (0,0), (0,1), (1,0), (1,1)$). The multiplicity of s and t knots is one more than s and t degrees, respectively, of the surface. The main advantage of the Bézier representation is that they are more easy to evaluate than general NURBS. Using *knot insertion* algorithms [20], it is possible to decompose each NURBS surface into a series of rational Bézier patches. We use Bézier patches to represent boundaries of the solid primitives in our algorithms.

A rational Bézier patch, $\mathbf{F}(s,t)$, of degree $m \times n$, defined in the domain $(s,t) \in [0,1] \times [0,1]$ and specified by a two dimensional array of control points $(\mathbf{v_{ij}}, w_{ij})$ (see Figure 7) is given by:

$$\mathbf{F}(s,t) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{v_{ij}} \mathcal{B}_i^m(s)\mathcal{B}_j^n(t)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij} \mathcal{B}_i^m(s)\mathcal{B}_j^n(t)}. \qquad (1)$$

$\mathcal{B}$ is the Bernstein basis function defined as

$$\mathcal{B}_i^m(s) = \begin{pmatrix} m \\ i \end{pmatrix} s^i (1-s)^{m-i}$$

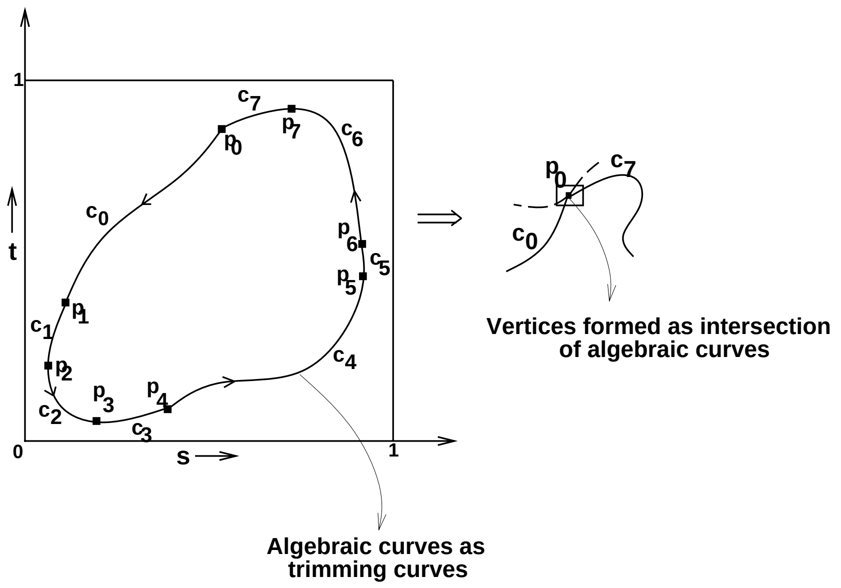
1
t
0
s
1
c_0
c_1
c_2
c_3
c_4
c_5
c_6
c_7
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
Vertices formed as intersection of algebraic curves
Algebraic curves as trimming curves

Figure 9: Representation of a trimmed patch as algebraic curve segments

A trimmed Bézier patch, as shown in Figure 8, has trimming curves in the domain of the patch and trims out the domain similar to its NURBS counterpart.

4. Representation of Solids

In this section, we describe our representation for a solid. Our algorithms assume that all B-rep solids are specified in this format. Every solid is represented as a set of *trimmed* parametric surface (tensor-product Bézier) patches which define the solid boundary.

Topological information of the solid is maintained in terms of an adjacency graph. It is similar to the winged-edge data structure[31,55]. To start with, we assume that each of the input objects has *manifold* boundaries, and the Boolean operation is *regularized*[53]. While it is possible to generate non-manifold objects from regularized Booleans on manifold solids, we assume that such cases do not occur. Given this assumption, it has been shown that an unambiguous topological representation is possible for a solid[31].

A trimmed patch consists of a sequence of curves defined in the domain of the patch such that they form a closed curve ($\mathbf{c_i}$'s in Figure 9). In the figure, the $\mathbf{c_i}$ refer to the algebraic curve segments forming the trimming boundary. The portion of the patch that lies in the interior of this closed curve is retained. Most of these trimming curves correspond to intersection curves between two surfaces. Therefore, these curves are typically algebraic curves that do not admit a rational parameterization[1]. We represent these curve segments ($\mathbf{c_i}$) by their algebraic equation (for accuracy), and a piecewise linear approximation (for efficient computation) and the two endpoints ($\mathbf{p_i}$ and $\mathbf{p_{i+1}}$).

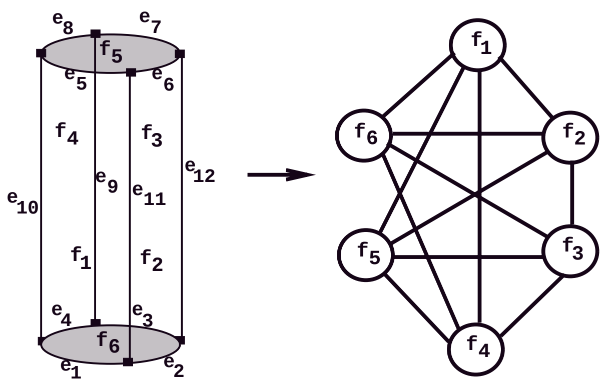

Figure 10: A cylinder and its face connectivity structure

This representation of a solid lends itself to a description in terms of *faces, edges, and vertices* analogous to the polyhedral case. Each *face* is a trimmed patch. Each of the trimming curves form an *edge*, and are formed as an intersection of two surfaces (faces). Finally, endpoints of edges form the *vertices.* They are represented as the intersection of three surfaces. Figure 10 shows an example solid and the face connectivity structure that we maintain. We also maintain the two faces that are adjacent to each edge, and an anticlockwise order of faces (when viewed from the exterior of the solid) around each vertex.

5. Algorithm Overview

We present a system, BOOLE, to effectively compute boundary representations of Boolean combinations of sculptured primitives and perform associated surface interrogations. It employs a combination of symbolic and numeric methods to compute the B-reps accurately and efficiently. The input to BOOLE is a CSG tree that describes the solid as a Boolean expression of primitive solids. We assume that the surface boundaries of all the primitives can be represented as a piecewise collection of parametric surface patches. However, our algorithms apply equally well on solids composed of algebraic surfaces. We use trimmed tensor-product rational Bézier patches to represent the surfaces. In order to compute the B-rep of the final solid, our algorithm computes the Boolean combination of the solids at the leaves of the CSG tree and propagates the results up the tree.

Given two such solids, our algorithm identifies pairs of surface patches from the two solids that intersect. The intersection curve between each such pair is computed using a new *surface intersection*[46] algorithm. The surface intersection algorithm ensures accurate evaluation of the intersection curve using algorithms for *curve-surface intersection*, *loop detection* and *curve tracing*. It makes use of a *matrix representation* of the intersection curve to accurately compute intersections between *trimmed surfaces* and to *classify* the various topological features generated by the intersection curve.

We now briefly describe our algorithm to evaluate the B-rep when two solids enter into a Boolean operation. Let the number of patches in one solid be m and those in the second solid be n and let the degree of each patch be $d_s \times d_t$. The algorithm to evaluate the Boolean operation between the two solids runs in six stages.

Stage 1: The main part of the algorithm is to compute the intersection curve between the two solids. Hence each patch of one solid has to be checked for intersection with each patch of the other solid. However, not all the mn pairs would intersect typically. We prune out most of the non-intersecting pairs using a two step process.

Initially, we compute the $3D$ bounding box for each patch (this is actually the axis-aligned bounding box of the control points of the patch). This is done in parallel as the construction of bounding boxes for each of the patches can be done independently. If a pair of bounding boxes do not intersect, the corresponding patches are also non-intersecting (*convex hull property* of Bézier patches[19]). All the redundant pairs are removed using a simple sort on all the bounding boxes. The next step of pruning uses linear programming. Linear programming is used to prune out pairs of patches whose convex hulls (as defined by their control points) do not intersect. This is a much stricter test, but is also more expensive, and hence we use the two-step pruning process. We formulate the linear programming problem as follows. Two patches do not intersect if there exists a separating plane between them. Thus we eliminate the patch pairs whose bounding boxes have a separating plane between them. We use Mike Hohmeyer's implementation of the linear programming algorithm by Seidel[72]. By applying these two methods on the two solids, we are left with few pairs of patches that are most likely to intersect.

Stage 2: The evaluation of the intersection curves between the remaining pairs of patches is performed next. We use a recently developed algorithm[46] (described briefly below) to compute the intersection curve between two parametric patches.

Stage 3: B-rep evaluation involves merging of the intersection curves computed in the previous stage. It can be shown that for closed C^0 continuous solids, the intersection curve between them must form a collection of closed curves in space for regularized Boolean operations. Merging is the process of collecting different pieces of the intersection curve and ordering them in sequence to form closed curves in space. The first step of merging is to merge the curves within a patch. After that curves between patches in each solid are merged.

Stage 4: The merged intersection curve partitions the boundary of the solid into various *components*. The components are generated by a simple graph traversal algorithm using the existing topological information in each solid.

Stage 5: Each component has the property that all the patches corresponding to it is either completely *inside* or *outside* the other solid. Therefore, it suffices to compute the inside-outside information of exactly one point in each component. If a solid is closed and not self-intersecting, then this query is answered by computing the number of intersections of a ray, emanating from that point, with the solid. If the number is odd, then the point is inside, otherwise it is outside the solid.

Stage 6: The particular set operation performed on the solids, and the inside/outside classification of the component, determine if a component is part of the new solid. The algorithm to generate the new solid forms the last stage of our algorithm. The connectivity information between various trimmed patches of the new solid is found using the topological information of the original solids and the intersection curves.

A brief description of the various algorithms used in BOOLE are given next.

5.1. Surface-Surface Intersection

Computing intersections of surfaces forms a critical part of any boundary evaluation algorithm. Modelers that perform Boolean operations on polyhedral solids have to deal only with plane-plane intersections. The essential difference between intersecting two planes and two free-form surfaces is that while the former generates a single line, the latter results in a high degree algebraic space curve with a number of components including open components, closed loops and singularities.

The main theme of our approach is to combine well known symbolic and numeric techniques for accurate and efficient computation. Our algorithm borrows a basic theorem of space curves from algebraic geometry. The crux of the theorem is that any algebraic space curve can be projected into an equivalent plane curve after a suitable linear transformation of the coordinates. Using this idea, we obtain a new representation of the intersection curve in a plane in the form of a matrix polynomial. We then evaluate the curve using numeric matrix computations and tracing algorithms. The algorithm guarantees determination of all components of the intersection curve for well-conditioned input cases by employing newly developed algorithms for curve-surface intersection and loop detection. Since all the computation is performed in floating point arithmetic, we evaluate the intersection curve to a user-specified tolerance[a]. The details of the surface intersection algorithm are given in Ref. [46]. The main steps of the algorithm are

- Given the two parametric surfaces, eliminate two of the variables using Dixon's resultant[14] and obtain the intersection curve in the plane as a bivariate matrix polynomial (implicit function of two variables). We represent the intersection curve as the singular set (values of the variables that make the matrix singular) of this matrix polynomial.
- Compute a starting point on each component of the intersection curve using curve-surface intersection and loop detection algorithms (described below).
- Subdivide the domain of the surface into regions such that each sub-region has at most one curve component. This process is called *domain decomposition.*
- If the separability condition is not satisfied due to singularities in the intersection curve, use local optimization techniques to isolate singular points within small portions of the domain.

[a]we use 10^{-5} in our implementation

- For each starting point, follow that component of the intersection curve using tracing methods.

Of all these steps, the elimination step dominates the computational cost. However, most of the computation involved in this stage can be performed off-line, and its cost amortized over a large number of surface intersection operations. This is particularly advantageous in boundary evaluation algorithms where the surface intersection routine is called hundreds of times for each solid. We have used this algorithm to generate surface boundaries of models like the submarine storage and handling room (Figure 1) and the Bradley fighting vehicle (Figure 2). On an average, our algorithm takes a fraction of a second (0.2–0.5 seconds) to perform one surface intersection.

5.2. Curve-Surface Intersection

We use curve-surface intersections to evaluate starting points on intersection curves of two surfaces and to perform ray-shooting tests (see section on *component classification*) to classify surface features with respect to solids. In these applications, we are interested in finding intersections only in a small subset of the real domain.

In BOOLE, we use a technique called *algebraic pruning* which uses matrix computations effectively to prune out regions of the domain with no intersections quickly. The basic idea of the algorithm is: Assume that we have an algorithm A which given a guess ρ to an intersection point generates the closest intersection point α. Let the separation between ρ and α be $\delta = |\rho - \alpha|$. Then, we know that there is no intersection point in the region $(\rho - \delta) < t < (\rho + \delta)$. We can safely prune out this region.

We use inverse power iterations (an iterative matrix computation algorithm) to converge to the closest intersection point. To the best of our knowledge, our algorithm performs faster than previously known curve-surface intersection algorithms when the number of intersections is fairly sparse. It performs competitively even when the intersection set is not sparse. This algorithm can be used without significant modification for finding zero-dimensional intersection sets like planar curve-curve intersection as well. Details of algebraic pruning can be found in Ref. [51].

5.3. Loop Detection

Loop detection in algebraic curves is an important part of any curve evaluation algorithm, and is traditionally considered hard. The reason for this is because searching for such curve features in higher dimensions is difficult. Any discretized search strategy suffers from the possibility of missing small loops. Our algorithm for loop detection[47] is based on a simple algebraic characterization. We use the fact that any real algebraic plane curve is continuous in the complex projective plane. Put simply, it means that while curve components appear disjoint when restricted to the real plane, they are actually connected into one single component in the complex plane. Therefore, by following the curve in complex space, we can

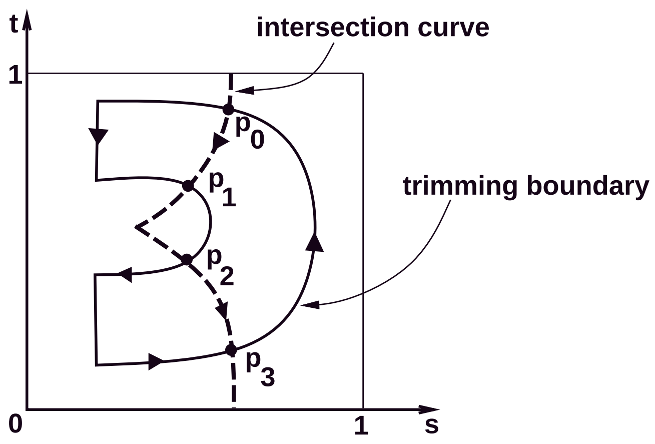

Fig. 11. Obtaining intersections between trimmed surfaces

reach at least one point on every loop component. The overview of the algorithm is described below.

- Evaluate all the starting points of the curve (in complex space) at the boundary of the domain.
- Follow each starting point by tracing out the curve in complex space.
- Few of these paths meet the real plane. These form candidates for loop components of the curve.

Compared to some of the traditional algebraic approaches which exhibit quadratic complexity in terms of the degree of the curve, our method traces out only a linear number of paths (our algorithm takes about 10-20 milliseconds, depending on the length, to trace out a single complex path completely). However, the number of complex paths to be traced could be high depending on the degree of the algebraic curve. This method offers the flexibility of being combined with other heuristics that would limit the number of complex paths traced.

5.4. Trimmed Surface Intersection

Our algorithm for boundary evaluation generates surface boundaries in the form of trimmed Bézier patches. Along with the parameterization of the surface, a trimmed Bézier patch also has an *oriented* closed curve called the *trimming curve* in the domain. This trimming curve determines the portion of the patch that is valid. For example, in Figure 11, the trimming curve is generated in a counterclockwise sense and the portion of the patch that is on the left of the curve is valid.

Our surface representation requires trimming so that they can maintain their closure under Boolean operations. When we perform a Boolean operation (union, intersection or difference) between two solids, their intersection curve determines which part of the original surface belongs to the final solid. If we look in the domain of one of these surfaces, the intersection curve partitions it. Only a few of the partitions are retained in the final solid. For the kind of operations we perform on solids, it is therefore, natural to represent their surface boundaries using trimmed parametric patches. Moreover, the trimming curves are portions of intersection curves themselves.

Our stand-alone surface intersection algorithm[46] deals with untrimmed parametric surfaces only. Applying this algorithm, only some parts of the intersection curve generated are valid for trimmed surfaces. For example, in Figure 11, the valid intersection curve is only between (p_0, p_1) and (p_2, p_3). Generating the p_i's accurately is not an easy problem because it involves intersections of two fairly high degree algebraic curves. The accuracy of these points is crucial because they determine important surface features of the new solid.

BOOLE uses an efficient and accurate algorithm to generate these intersection points. The algorithm uses the piecewise linear representation (generated by curve tracing from the surface intersection algorithm) of the intersection and trimming curves to compute approximations for these points. We then use the patch parameterizations of the surfaces involved and the analytic representation of the intersection curve to refine the approximations using iterative minimization techniques. A detailed explanation of this technique can be found in Ref. [42].

5.5. Component Classification

When two solids enter into a Boolean operation, only portions of the surfaces of each solid remain in the final solid. The portions to be retained are determined by the intersection curve between the two solids. For example, consider a union operation between two solids A and B. After computing the intersection curve, only portions of A that lie outside B and those of B that lie outside A are retained in the solid $A \cup B$. Similar characterizations exist for other operations as well. Component classification refers to algorithms that generate maximally connected portions of the boundary π of a solid that have the property that π either lies completely inside or outside (*orientation-invariant component*) the other solid. Furthermore, it also deals with the resolution of the inside/outside nature of each orientation-invariant component.

We use the topological information (connectivity between the various features) of each solid and the intersection curve between them to generate the various orientation-invariant components. Our algorithm creates an associated undirected graph and computes its connected components for this purpose. It also generates another graph, , , whose vertices are the various orientation-invariant components. An edge exists between two such vertices if and only if orientations are opposite with respect to the other solid. This connectivity information turns out to be very useful in classifying the various components efficiently.

When two polyhedral solids intersect, it is fairly easy to classify the inside/outside nature of the various components by performing simple local tests based on the orientation of the intersection curve[31]. However, for solid boundaries composed of curved surfaces, local tests cannot be performed. The main reason for this is the

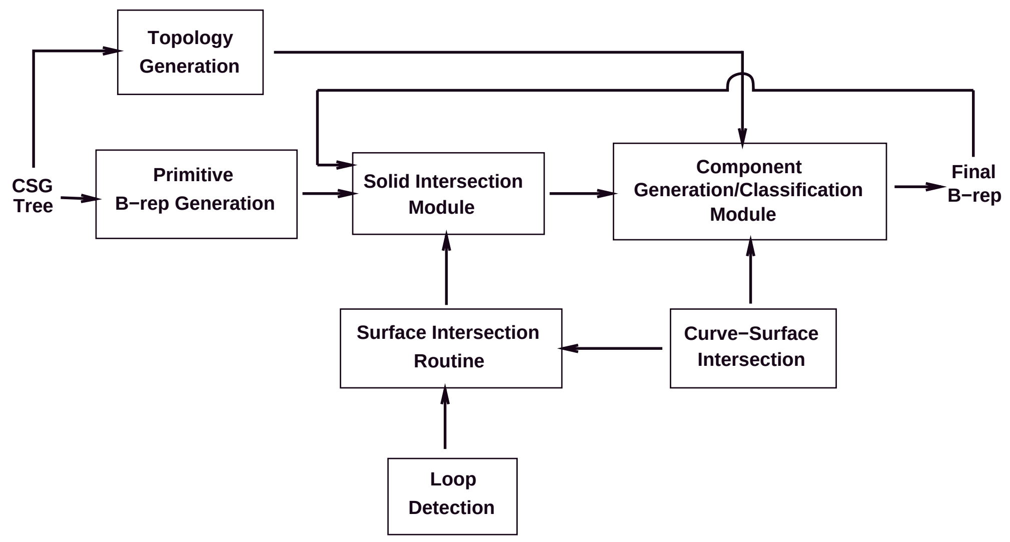
Topology Generation
CSG Tree
Primitive B−rep Generation
Solid Intersection Module
Component Generation/Classification Module
Final B−rep
Surface Intersection Routine
Curve−Surface Intersection
Loop Detection

Fig. 12. Functional modules in the BOOLE system

complicated nature of the intersection curve. We use an algorithm based on ray-shooting to perform the classification tests. Ray-shooting is based on the following simple fact: A point is inside a closed solid if any semi-infinite ray originating from that point intersects the boundary of the solid odd number of times; otherwise, it is outside. We use our curve-surface intersection algorithm to perform ray-shooting. The curve-surface intersection algorithm generates all points that lie in the entire domain of the surface. However, the actual portion of the surface that is part of the solid boundary is trimmed. We have to check if the intersection points obtained by the curve-surface intersection algorithm actually lie inside the trimmed region of the domain. We maintain a triangulation of the trimmed domain, and use point location queries to perform this test. Further, curve-surface intersection is a fairly expensive operation (roots of a high degree univariate polynomial), so it behooves us to reduce the number of such invocations. Our algorithm uses the connectivity information between the various components and performs just one ray-shooting test per solid per operation. This significantly speeds up our computation.

The accuracy of the ray-shooting test is very important in determining the final solid. Double precision arithmetic or degenerate ray-surface intersections could possibly change a result from inside to outside or vice-versa. We use an analytic representation of the intersection curve and stable matrix computations to prevent such catastrophic errors.

A detailed version of the overall boundary evaluation algorithm can be found in Ref. [45,44,42].

6. Implementation of BOOLE

One of the main contributions of this paper is a complete implementation of all the algorithms presented. The implementation of algebraic pruning, loop detection, surface-surface intersection and boundary evaluation algorithms are parts of the BOOLE solid modeling system. Given a CSG tree whose leaves are chosen from a pre-defined set of primitive solids, BOOLE generates the surface representation of the boundary of the final solid as a collection of trimmed Bézier patches as well as the topological information in a graph structure. The various modules in our system and their dependency relations are shown in Figure 12.

We have implemented our system on single processor architectures like SGI Maximum Impact (with one 250MHz R4400 CPU) and Sun-Solaris, as well as a parallel version of the algorithms on shared memory multiprocessor architectures like SGI Onyx (with up to 6 194MHz R10000 CPUs, 1MByte main memory). Our current sequential implementation can perform one Boolean operation on common solids with quadric or quartic degree surfaces (spheres, ellipsoids, tori, cylinders and cones) in about 3-4 seconds, while the parallel version can do the same in one second or less.

Given a CSG tree, our system generates the boundary representation of all the primitives involved in the form of trimmed Bézier patches along with their topology information. For each Boolean operation, the B-reps of the two solids are passed to the solid intersection module. This module is responsible for generating the intersection curve between the two solids. The curves are generated in the domain of each patch as well as in 3-space. We maintain the curve in 3-space (space curve) so that we can verify if the intersection curves form a closed loop. This is a check-pointing operation, and if the curve is not closed, we declare an error and try to recompute the curve. The space curve is also used during model visualization. The solid intersection module relies on the surface-surface and curve-surface intersection algorithms to generate the curves. These algorithms are implemented in C and makes use of a number of matrix operations like SVD, matrix eigendecomposition and inverse iterations. These routines are available in public domain in the form of Fortran libraries like EISPACK[24] and LAPACK[2]. The main advantage of using these libraries is that they are carefully and efficiently implemented by numerical analysts and well tested on a number of benchmarks. Further, most of the matrix routines also return the condition number of the problem. We use this information to predict the conditioning of our original problem or to detect inaccuracies in our computation.

The intersection curves are fed into the component generation/classification module. Initially, we partition the domain of each patch as determined by the intersection curve and determine the connectivity structure of the partitions within each patch. Using this information and the original topology of the two solids, we create the graph whose connected components generate orientation invariant surface partitions. Construction of the graph , (connectivity information between various orientation invariant surface partitions) is described in Ref. [44]. Classification is done by ray-shooting. The ray-shooting test can be reduced to a collection of ray-surface intersections. In our implementation, we use *algebraic pruning* to perform this operation.

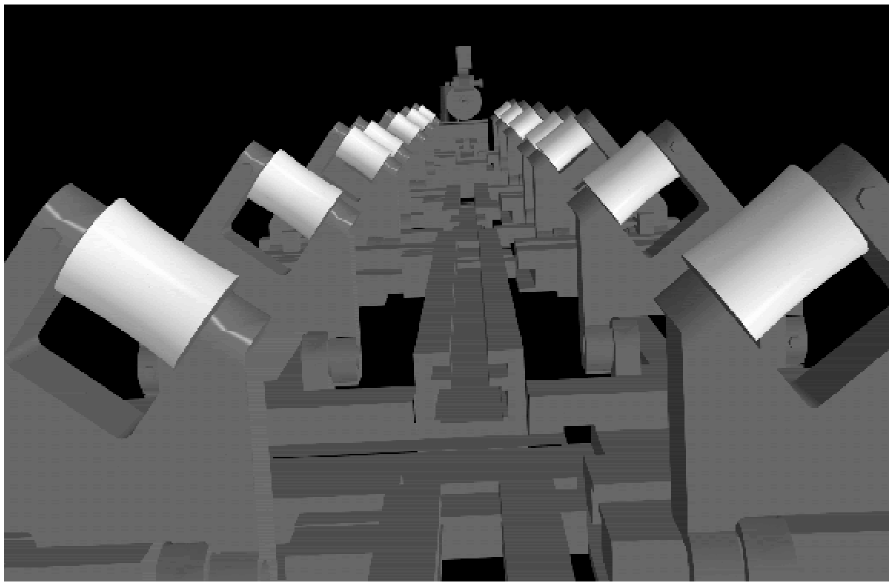

Fig. 13. B-rep of Pivot from Submarine model (4100 Bézier patches) [Courtesy: Electric Boat]

The algorithm for component classification proceeds by computing all the intersections of a randomly directed ray with all the trimmed patches of the other solid. The parity (odd/even) of number of intersections decides the orientation (inside/outside) of the component. Guaranteeing the correctness of this operation is very crucial for the correctness of the final B-rep. In our system, we perform a number of redundant computations to ensure this. The ray-surface intersection algorithm generates intersection points in the domain of each surface. If the chosen ray passes through the boundary of two adjacent patches, this point may be counted twice (once for intersection with each patch). To avoid this, we compare the corresponding intersections in 3-space and eliminate duplications. We also shoot multiple random rays to ensure correct parity. Since BOOLE does all its computation in double-precision floating point, it is possible to misinterpret the result of a single computation. For example, if the random ray is nearly tangential to the surface, we might eliminate multiple intersections because of their proximity. By performing multiple ray-shooting tests, the probability of misinterpretation is reduced. The result of the classification of one component is propagated throughout the adjacency graph , to resolve the other components. The propagation prevents us from having to do ray-shooting for each component, although it increases the chance propagating a wrong result. In our experience, because of our careful ray-shooting computation, we have not run into this problem.

The B-rep of the resulting solid and its topological structure are generated based on the Boolean operation being performed. This data is fed back to the solid intersection module if the new solid enters into another Boolean operation.

7. Architecture of the BOOLE system

Figure 14 shows the basic architecture of the BOOLE system. The bottommost layer (Layer I) is composed of five major modules - the set of numeric libraries, symbolic module, geometric module, routines to manipulate parametric curves and surfaces, and graph algorithms. Here is a brief description about each.

- **Numeric libraries:** We make use of the public domain Fortran libraries EISPACK[24] and LAPACK[2]. These libraries provide most of the routines required by our algorithms like QR decomposition for computing eigenvalues and eigenvectors, LU decomposition for solution of linear systems and Singular Value Decomposition. Various parts of our surface-surface intersection algorithm use these numerical algorithms. We have also implemented the algorithm for local minimization given in Press et. al[62]. The minimization routine is used in conjunction with the tracing algorithm to improve the accuracy of the intersection curve.

- **Symbolic module:** This module comprises basically of routines for computing various resultants. We require only two kinds of resultant routines - Sylvester[68] (eliminating one variable from system of two equations) and Dixon[14] (eliminating two variables from system of three equations). We use Sylvester resultant during curve-curve intersection as part of the algebraic pruning algorithm. Dixon's resultant is mainly used to compute implicit forms of surfaces. These routines are implemented in double precision arithmetic.

- **Geometric module:** The geometric module contains algorithms for triangulation of simple polygons, point location in planar arrangements, linear programming and bounding box overlap tests. We use a very fast implementation of Seidel's triangulation algorithm[73] provided by Atul Narkhede et. al[58]. The point location algorithm based on the triangulation algorithm was also implemented by Atul Narkhede. We use Mike Hohmeyer's[32] implementation of Seidel's randomized linear programming algorithm[72].

- **Curve/Surface manipulation module:** This module primarily handles all the low-level routines for manipulating parametric curves and surfaces. Typical algorithms are curve and surface subdivision (at certain parameter values), point evaluation on surfaces, pseudo-Gauss map evaluation for loop detection and curve fitting. Curve fitting is a part of the BOOLE system that fits a parametric curve to an ordered set of points obtained after curve tracing. This routine is not used by the BOOLE system directly for B-rep computation. Rather, it is used as a means of data compaction by a display system (developed at UNC) that renders large NURBS models.

- **Graph Algorithms:** This final module is used in maintaining topology information for each solid in our system. Apart from the simple tools to manipulate graph structures, it contains an algorithm to generate connected components in graphs. The algorithm uses repeated calls to a depth-first traversal routine

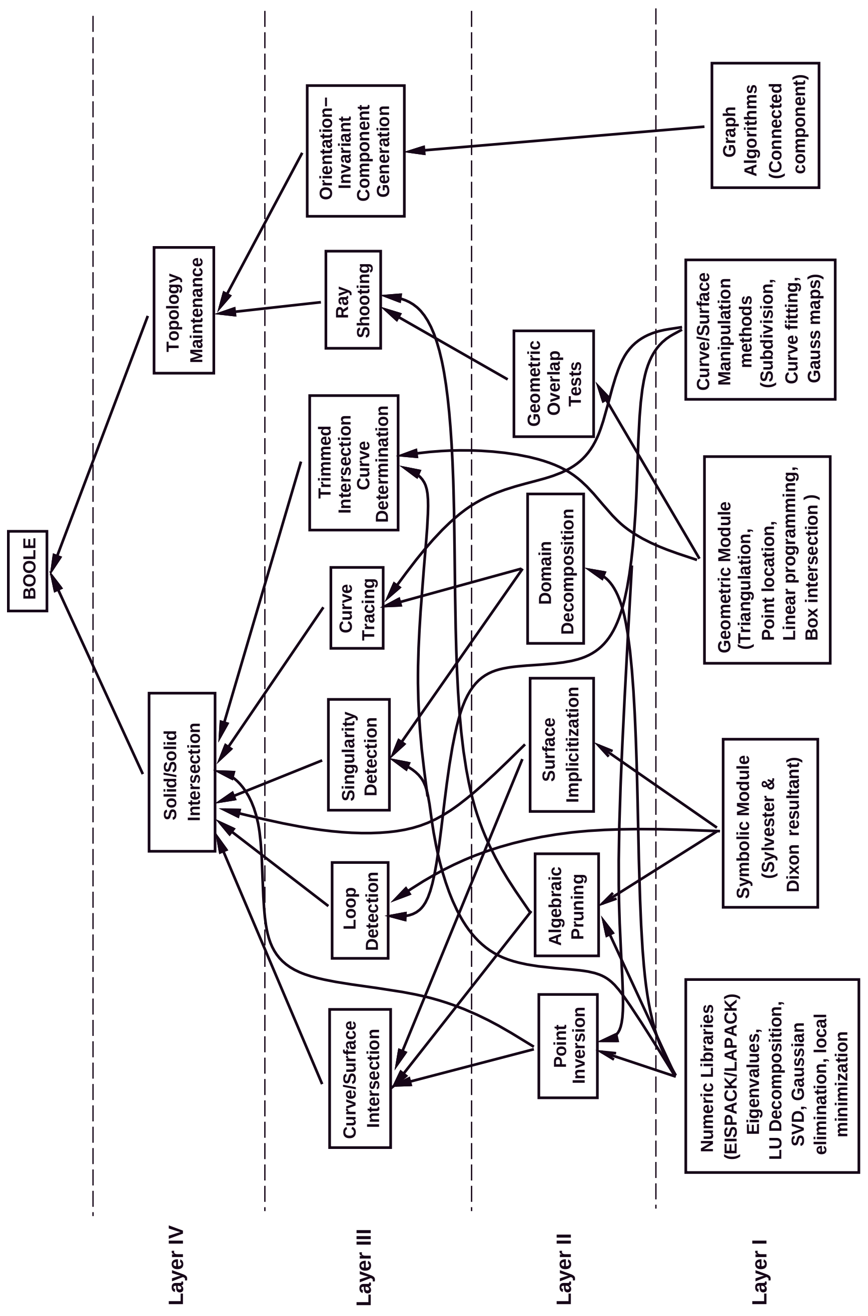
BOOLE
Topology Maintenance
Solid/Solid Intersection
Orientation–Invariant Component Generation
Ray Shooting
Trimmed Intersection Curve Determination
Curve Tracing
Singularity Detection
Loop Detection
Curve/Surface Intersection
Geometric Overlap Tests
Domain Decomposition
Surface Implicitization
Algebraic Pruning
Point Inversion
Graph Algorithms (Connected component)
Curve/Surface Manipulation methods (Subdivision, Curve fitting, Gauss maps)
Geometric Module (Triangulation, Point location, Linear programming, Box intersection)
Symbolic Module (Sylvester & Dixon resultant)
Numeric Libraries (EISPACK/LAPACK) Eigenvalues, LU Decomposition, SVD, Gaussian elimination, local minimization
Layer IV
Layer III
Layer II
Layer I

Fig. 14. Various implementation layers in BOOLE

in graphs. The running time of this algorithm is linearly proportional to the number of edges in the graph.

Layer II of our system contains routines that are directly called by our algorithms for curve-surface intersection, loop and singularity detection, curve tracing etc. These routines are listed in Figure 14. Given a point on a curve or surface, the problem of *point inversion* deals with the determination of parameter values which results in that point. Mathematically speaking, given a rational parameterization of a surface, $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and a point $(x,y,z) \in \mathcal{R}^3$, find the parameters (s_1, t_1) such that

$$\begin{aligned} X(s_1,t_1) &= xW(s_1,t_1) \\ Y(s_1,t_1) &= yW(s_1,t_1) \\ Z(s_1,t_1) &= zW(s_1,t_1) \end{aligned}$$

This operation is performed very often during curve tracing. *Algebraic pruning* is our method of solving zero-dimensional systems based on inverse power iterations. This algorithm relies heavily on the numeric libraries. We use algebraic pruning for curve-surface intersection queries and ray-shooting. The role of *surface implicitization* and *domain decomposition* in the surface-surface intersection algorithm are described in Ref. [46]. *Geometric overlap* tests are performed to quickly prune out non-intersecting curves and surfaces. We use the implementation of linear programming and bounding box overlaps from layer I for this purpose.

The modules in Layer III include curve/surface intersection, loop and singularity detection, curve tracing, trimmed intersection curve determination, ray-shooting and orientation-invariant component generation. Each of these modules call a number of routines from layers I and II. The dependency structure of the various modules is shown in the figure. The modules in Layer III are in turn called by the topmost layer which includes solid-solid intersection and topology maintenance modules.

8. Robustness and Accuracy

One of the main problems in B-rep generation is robustness. An algorithm is said to be robust if for every valid input instance of the problem, it generates the corresponding valid output member. Consider the algorithm as a function $\mathcal{F}$ from the input set $\mathcal{I}$ to the output set $\mathcal{O}$.

$$\mathcal{F} \; : \; \mathcal{I} \; \rightarrow \; \mathcal{O}$$

In this definition, it is important for the algorithm to identify the type of input instance $i \in \mathcal{I}$ because the sequence of steps executed by the algorithm depends directly on i.

Most geometric algorithms are developed assuming that the input data are in general position, and that exact arithmetic provides reliable geometric primitives.

However, for reasons of efficiency and feasibility, most implementations use floating point instead of exact arithmetic. Thus, the correctness of the mathematical algorithm does not extend directly to the implementation, and the system fails for seemingly innocuous input data (failure to classify the input instance correctly). This is the problem of "robustness" in geometric computing.

However, if a particular instance is degenerate, the value of the corresponding expression is smaller than the errors accumulated due to fixed precision. There are two ways of dealing with this problem - tolerances and error estimates[23]. Estimating tolerances when evaluating a complex sequence of predicates is non trivial, and error estimates are too pessimistic to be useful.

We shall now identify two areas where our algorithm is susceptible to failure when using floating point arithmetic. Most of these errors finally boil down to either point orientation tests or comparison between two floating point numbers. We do not guarantee that these are the only two areas where our algorithm could fail. However, based on the tests we performed on the system for the last couple of years, we found that the source of failure was because of the above two reasons.

Inaccurate point inversion for curve merging: It is a well-known fact that the intersection curve of two parametric surfaces is not rationally parameterizable in general. As a result, these curves are approximated as piecewise linear curves or splines to within a fixed tolerance (which is either too conservative or arbitrarily chosen). 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. To compute the actual intersection curve for trimmed patches, we need to compute the intersection points of the curve with the trimming boundary. If the boundary curves or the intersection curve are not accurate, neither are the intersection points. They may not even lie on the actual intersection curve. Corresponding to these intersection points, we need to compute points on the other patch (let us call them "inverted points") which determine the portions of the intersection curve to retain. This process is *point inversion* which was described in the previous section. Two problems can arise in inversion: (a) there may not be any corresponding point on the other patch (because the intersection points do not lie exactly on the intersection curve), or (b) the inverted points could be positioned such that they do not match up for curve merging.

Using our analytic representation of the intersection curve (as the singular set of a bivariate matrix polynomial[46]), we ensure accurate computation of $\mathbf{p_i}$'s. We estimate an approximate value for the $\mathbf{p_i}$'s by performing intersections with the piecewise linear approximations obtained during curve tracing. This approximate value is refined by performing minimization on an objective function based on the analytic representation of the curve. The corresponding inverted point is also obtained using the minimization function.

Inaccurate point classification: Another area where floating point errors result in failure of the algorithm is during component classification. As described earlier, we use ray shooting for this purpose. The entire computation boils down to classifying whether a point lies inside or outside the trimming region. Figure 15

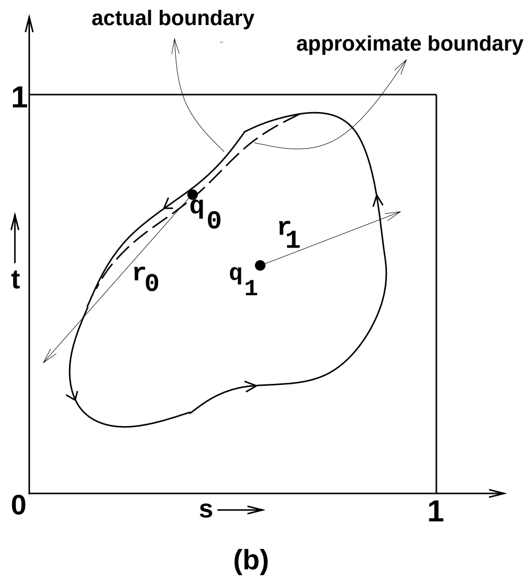

Fig. 15. Inaccurate point classification

shows an example. In most cases, classifying points like $\mathbf{q_1}$ is not a problem. One ray-shooting query will determine it. 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 could result in topologically inconsistent answers. We improve the accuracy of the classification test by using the analytic representation of the trimming curve (bivariate matrix polynomial). Since the algebraic curve is a zero set of a polynomial, there is a sign change on either side of the curve in the local neighborhood of the boundary. The sign of the polynomial with the point p substituted for the variables gives the classification of the point. Since the curve is represented as the determinant of a matrix polynomial, we have to evaluate the sign of this determinant. We use *singular value decomposition* (SVD) to accomplish this task.

Given a numerical square matrix $\mathbf{A}$, SVD decomposes it into the form

$$\mathbf{A} = \mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V^T},$$

where $\mathbf{U}$ and $\mathbf{V^T}$ are orthonormal matrices, and $\mathbf{\Sigma}$ is a diagonal matrix whose entries are all positive. This implies that the sign of the determinant of $\mathbf{A}$ is the same as the product of the signs of the two orthonormal matrices (determinant is +1 or -1). We can safely perform Gaussian elimination to determine the sign of these determinants. The results provided by SVD can actually be verified by computing an upper bound on the absolute error, ϵ, in the smallest singular value, σ [28]. If the interval $[\sigma - \epsilon, \sigma + \epsilon]$ does not contain zero, we can guarantee the correctness of the sign of the determinant. If the above interval includes zero, we do not know of any floating-point based method to compute the sign of the determinant. We must

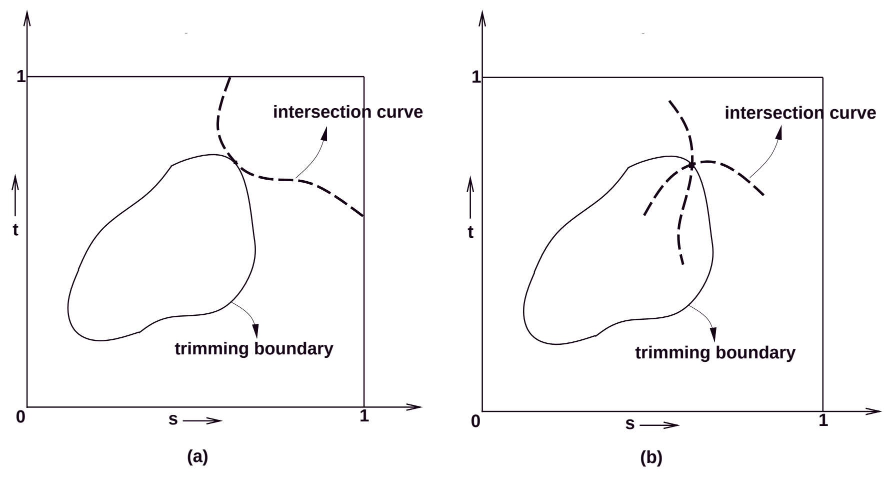

Fig. 16. (a) Surface-edge contact degeneracy (b) Four surfaces meeting at a point

resort to exact rational techniques.

We also perform a number of check-pointing operations in our implementation that control the accumulation of floating point error. Since the implementation was done using floating point arithmetic, we also use tolerances to compare such values. Finding a tolerance that works for all models is very difficult. In some cases, we had to change tolerances to make our system work.

8.1. Incorporating Exact Rational Arithmetic to BOOLE

Currently, we are incorporating B-rep computation using exact rational arithmetic[39] to prevent most robustness and accuracy problems in BOOLE. The use of exact arithmetic can slow down the computation time significantly (about 10-100 times based on our initial estimates) for low degree primitives, and even slower for higher degree solids. We have identified a few lower-level routines where the algorithms based on floating-point arithmetic are susceptible to failure. These include determinant sign evaluation, orientation of points with respect to curves, and component classification. We perform such tests *reliably* using exact arithmetic. The implementation of these lower-level routines into a separate system has just been completed. We have not yet integrated this part into BOOLE. We hope to perform these exact tests only to identify and resolve degenerate or nearly degenerate situations.

The accuracy of the B-rep generated is determined by the accuracy of the intersection curves between solids. In our system, the accuracy of these curves can be controlled by the user. Depending on the application, our system can generate very accurate B-reps at the expense of computation time.

9. 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. These include

- **Two surfaces meeting at a point:** This case is particular only to curved surfaces. Since the surfaces meet at a point which lies in the interior of their respective domains, their normals are coincident. This corresponds to a singularity. We determine this by minimizing an energy function used to determine singularities[46].
- **Two surfaces tangentially intersecting at a curve:** This is a degenerate case when the surfaces are tangent to each other along that curve. This case also occurs only with curved surfaces. We will be able to detect this when we generate the adjacency graph by finding that two adjacent components actually have the same orientation with respect to the other solid. Another scenario when this case occurs is if the intersection curves do not form a closed loop in space.
- **Two surfaces overlapping:** This corresponds to a face-face overlap in the polyhedral domain. If two surfaces are overlapping, their intersection set is two-dimensional. Essentially, our bivariate matrix polynomial representing the intersection curve is singular for all values in the domain. We perform this test by sampling the domain and determining the ranks of the resulting numeric matrices using SVD.
- **A surface just touching an edge:** This is an edge-face contact in the polyhedral domain, and can happen when three surfaces meet in a curve. In our representation, this will appear as an intersection curve which is tangent to a trimming curve (see Figure 16(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 Figure 16(b)). Assume these three curves are $f1$, $f2$, and $f3$. We can find out whether this case has occurred by checking equality of the intersection of $f1$ and $f2$ with the intersection of $f1$ and $f3$ (or $f2$ and $f3$). Currently, these equality tests are performed with tolerances. Once the rational arithmetic module is added, we hope to do these tests exactly.

Degeneracies in the polyhedral case can generally be classified into the category of four planes meeting at a point. It has been shown [23] that a simple perturbation scheme applied to a single basic geometric predicate can eliminate these degeneracies. No obvious extension of this method is known for curved parametric surfaces (there are some theoretical notions of perturbation for implicit surfaces), though there is hope that some perturbation method can be developed using exact rational arithmetic which would work similarly.

10. Parallel Implementation

In this section, we present the parallel version of our implementation of BOOLE. Since we are dealing with sculptured solids with trimmed Bézier patches, as opposed to polyhedral solids, the complexity of the whole boundary evaluation system is increased significantly. The time taken for the surface-surface intersection algorithm is a cubic function of the degree of the patch [42] in the worst case. Further, the complexity of ray-patch intersection evaluation is again dependent on the degree of the patch. These parts are computationally most intensive and form the main bottleneck in terms of system performance. However, it is easy to see that for two different of surface-surface or curve-surface pairs the computation can be independently carried out. To improve the computation time, we have implemented a parallel version of the algorithm on existing shared memory multiprocessor architectures like SGI-Onyx. The various stages of our algorithm is explained using an example in Figure 17 and Figure 18. These stages are quite similar to the overview described for the sequential algorithm except that in each of the bounding-box and linear programming tests, surface-surface intersection and component classification steps, the computation is distributed among various processors. A preliminary version of the parallel algorithm was presented at Eurographics'97[43]. We would like to emphasize that while the load balancing algorithm is fairly straightforward, the implementation issues in the context of our system were fairly involved.

One of the main issues that arise while parallelizing an algorithm over many processors is to ensure that each processor performs roughly equal amount of work. This issue of load balancing is discussed next.

Load Balancing Algorithm

The problem of load balancing arises when an algorithm has to be parallelized among a number of processors. The running time of the parallel algorithm is directly related to the maximum execution time of the task at a single processor. It is clear that the most effective parallel algorithm is one where the tasks are equally distributed among all the processors. The problem of load balancing has received considerable attention for a long time due to the fact that a single scheme is not applicable for parallelizing all algorithms[49,82,81,26,27]. The effectiveness of different

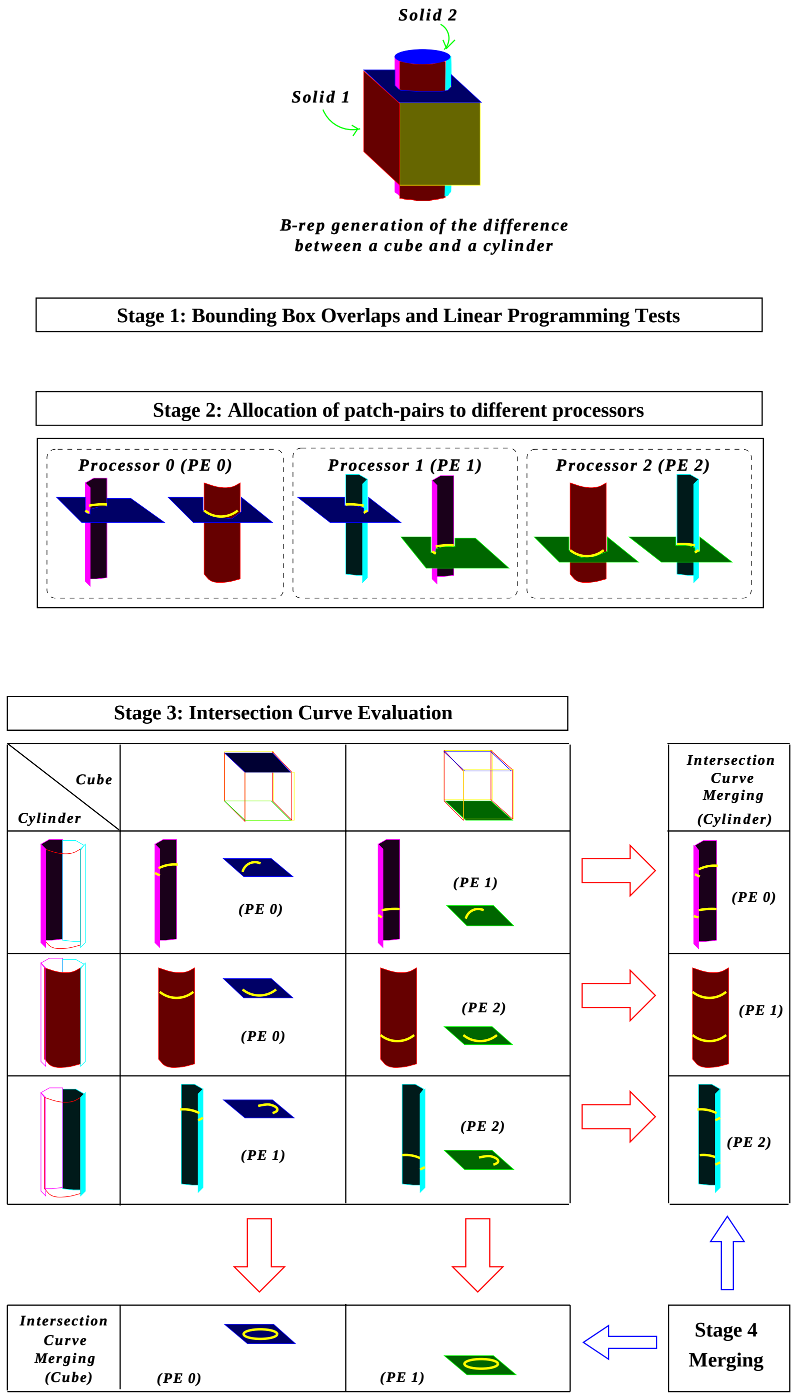
Solid 2
Solid 1
B-rep generation of the difference between a cube and a cylinder
Stage 1: Bounding Box Overlaps and Linear Programming Tests
Stage 2: Allocation of patch-pairs to different processors
Processor 0 (PE 0)
Processor 1 (PE 1)
Processor 2 (PE 2)
Stage 3: Intersection Curve Evaluation
Cube
Cylinder
(PE 0)
(PE 1)
(PE 0)
(PE 2)
(PE 1)
(PE 2)
Intersection Curve Merging (Cylinder)
(PE 0)
(PE 1)
(PE 2)
Intersection Curve Merging (Cube)
(PE 0)
(PE 1)
Stage 4 Merging

Fig. 17. Intersection curve computation and curve merging

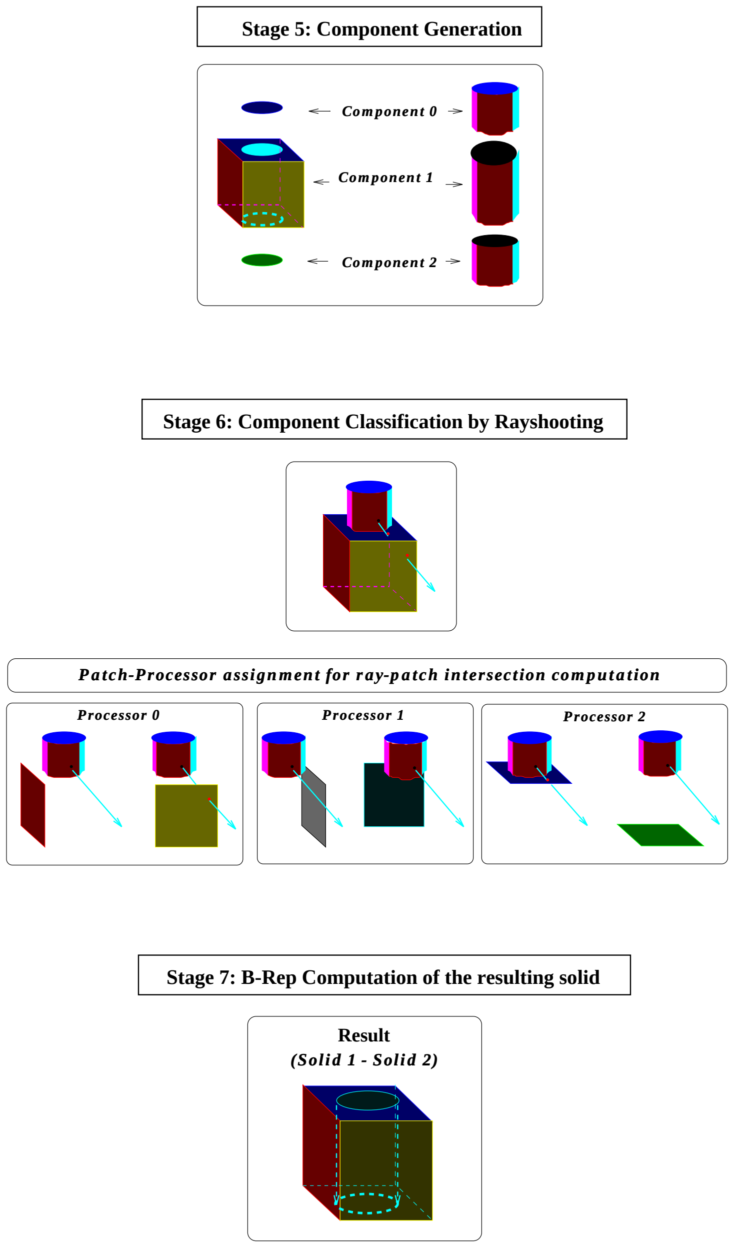
Stage 5: Component Generation
Component 0
Component 1
Component 2
Stage 6: Component Classification by Rayshooting
Patch-Processor assignment for ray-patch intersection computation
Processor 0
Processor 1
Processor 2
Stage 7: B-Rep Computation of the resulting solid
Result
(Solid 1 - Solid 2)

Fig. 18. Component generation, classification and B-rep computation

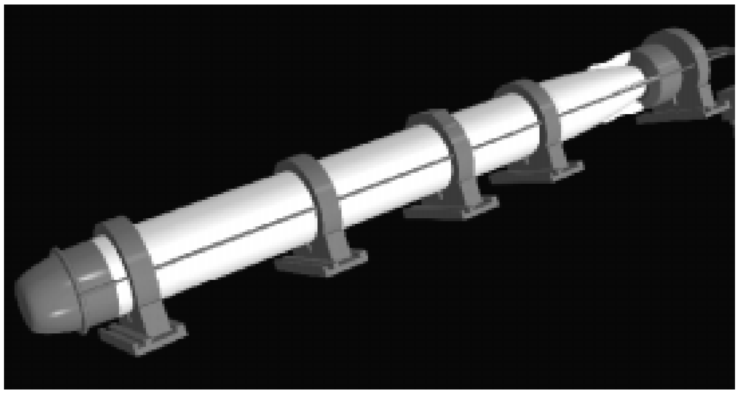

Fig. 19. B-rep of Shipping line from Submarine model (3400 Bézier patches) [Courtesy: Electric Boat]

techniques varies with the nature of the problem it is used for. Hence there arises a need for newer problem specific analysis methods which help in choosing the most effective load balancing technique. We shall now describe three such techniques that we use to shared memory multiprocessor architectures for boundary computation.

- **Static load balancing:** Static load balancing is done by dividing the given problem consisting of n tasks into p (number of processors) parts and submitting each part to a single processor. The size of each problem piece is precomputed and is not changed during execution. This technique works best when the processing time of each of the tasks is known, and the number of tasks does not change during execution. Extracting parallelism in our B-rep converter starts from computing the bounding boxes for all the patches (Stage 1 in Figure 17). As the bounding box computation for each patch is independent of the other, this can be easily parallelized. Further as the amount of work that is to be done for the bounding box computation for each patch is approximately the same, load balancing is achieved statically. Once the bounding boxes for all the patches have been computed, the overlap tests is also performed in parallel.

- **Global queue:** In many algorithms, it is not possible to estimate the execution time of each task. For example, execution time for computing the intersection curve between two surfaces can vary depending on the number of curve components, and length (in terms of number of points traced) of each component. In this technique, when one processor is accessing the task queue, the queue should be locked to ensure exclusive access (mutual exclusion). This technique achieves the best load balancing, though the extra work done for balancing the load in the form of locks might offset its advantage. In our system, using global queues with locks to perform load balancing was not as efficient as dynamic load balancing (described below). We believe it is because of the reasons cited above.

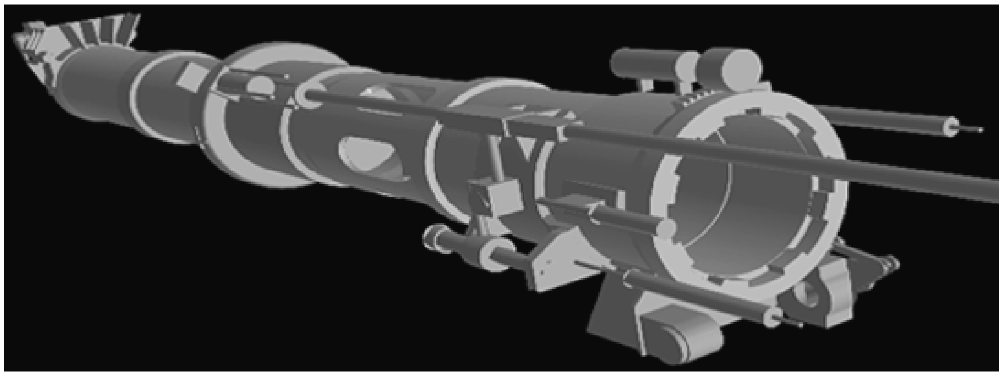

Fig. 20. B-rep of Torpedo tube from Submarine model (1200 Bézier patches) [Courtesy: Electric Boat]

- **Dynamic load balancing:** In this technique, a local job queue is maintained for each processor. Initially, tasks are assigned to every processor similar to the static load balancing scheme. However, due to suboptimal task division, some processors might complete their tasks before others. In this scenario, the idle processors share the load with the busy processors, thereby balancing the load dynamically. If we can ensure that each busy processor is accessed by only one idle processor at any time, then a lock-free implementation of this scheme is possible. We can also ensure that each task is processed only once, and no task is left out. In our application, load balancing is efficiently achieved by minimal use of locks. Therefore, we use this approach for our most computationally intensive tasks like surface-surface intersection (Stage 3 in Figure 17) and ray-shooting computation (Stage 6 in Figure 18).

If we ensure that only one idle processor will access a particular busy processor, then a lock free implementation of dynamic load balancing is possible. We enforce a unique one to one correspondence between an idle and busy processor using the following algorithm. A shared global variable *WhichIdleProc* stores the id of the idle processor, which now has the chance to choose its busy processor. This serializes the operation of finding an idle-busy processor pair. In our implementation, we choose a single lock to guard this critical section because the computation time for surface-surface intersection and ray-shooting dominates one locking operation. With each busy processor, we associate a shared variable *MyIdleProc*, which stores the idle processor id that has been paired up with that particular busy processor. These variables are initialized to **NIL**, referring to none of the processors. Each processor also maintains its processor number in a local variable **myid**. Whenever a processor becomes idle, it executes the following code.

```
{
 If ( WhichIdleProc == NIL) then {
   GetLock( GetMeAccess);
```

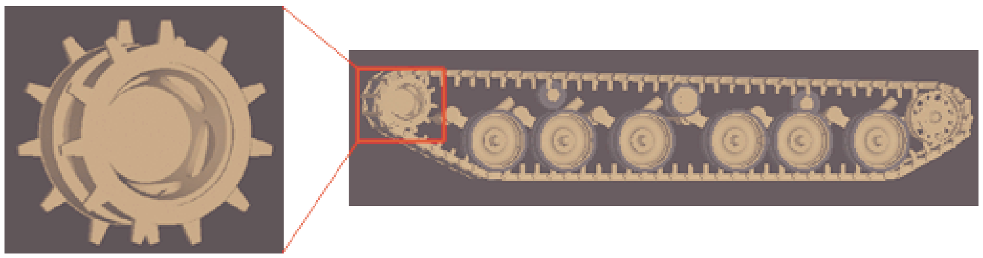

Fig. 21. Track from the Bradley model showing placement of drivewheel model (15000 Bézier patches) [Courtesy: Army Research Labs]

```
    if($GetMeAccess$ == NIL) {
      $GetMeAccess$ = **myid**;
      $WhichIdleProc$ = **myid**;
    }
  ReleaseLock($GetMeAccess$);
}
/* Waiting for my chance */
while ($WhichIdleProc \neq$ **myid**);

/* All tasks completed */
If (NoMoreBusyProc()) then exit;

/* All Busy processors are being load
balanced by some idle processor */
while (GetBusyProc() == **NIL**);

/* Got a Busy Processor to pair up with */
$MyBusyProc$ = GetBusyProc();

/* Make sure no one else captures this busy processor */
$MyIdleProc[MyBusyProc]$ = **myid**;

/* Give chance to next idle proc to find its partner */
If (NextIdleProc()) then $WhichIdleProc$ = NextIdleProc();

/* No one to grab the chance */
                      else {
                       GetLock($GetMeAccess$);
                          $WhichIdleProc$ = **NIL**;
                          $GetMeAccess$ = **NIL**;
                       ReleaseLock($GetMeAccess$);
                      }
```

Model	# of CSG opns.	Running time (in secs.)				# of patches (in B-Rep)
		BB & LP test	SSI	Ray-shooting	Total	
Fig. (a)	20	1.7	41.3	13.6	77.0	137
Fig. (b)	5	0.3	9.7	3.6	16.3	89
Fig. (c)	5	0.8	11.6	5.4	18.5	116
Fig. (d)	27	2.3	58.9	17.8	98.7	169
Fig. (e)	10	1.8	28.5	6.7	41.1	69
Fig. (f)	21	2.0	35.1	13.8	64.2	146

Table 1. Performance of our system on parts of the submarine model (Figure 22)

```
  /* Balancing the load with the partner */
  LoadBalance(MyBusyProc);

  /* Finished load sharing; Freeing my partner */
  MyIdleProc[MyBusyProc] = NIL;

  /* Register myself as busy */
  If (IHaveLoad()) BUSY[myid] = TRUE;

  /* Work on new list of tasks */
  PerformSurfaceIntersection(); or PerformRayShooting();

  /* Register myself as idle */
  BUSY[myid] = FALSE;
}
```

Initially, the variable *WhichIdleProc* has to be set by the idle processor to gain access to the list of busy processors. Race condition occurs only when the variable *WhichIdleProc* is **NIL** and more than one idle processor try to access it. By making *WhichIdleProc* a critical resource, we can ensure mutual exclusion while setting this variable. This can be achieved by using locks. The number of locking operations can be reduced by allowing free access to *WhichIdleProc* and introducing a new shared variable *GetMeAccess*, which is locked only when a race condition occurs. Locks can be totally avoided by maintaining a random permutation of the busy processor list locally in every processor. This does not guarantee that a single idle processor captures a busy processor, however, the probability of a race condition is very small.

11. Performance

In this section, we highlight the performance of both the sequential and parallel algorithm on some real-world models. We obtained a model of a submarine storage and handling room through the courtesy of Electric Boat Inc., a division of General Dynamics. This model consists of about 5000 solids. Many of the primitives are composed of polyhedra, spheres and cylinders. Additional primitives include

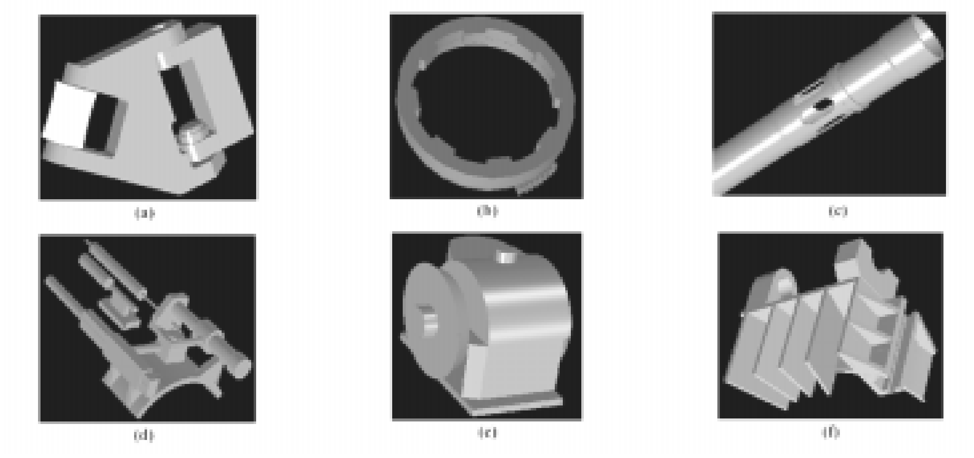

Fig. 22. B-reps of some solids from the submarine storage and handling room

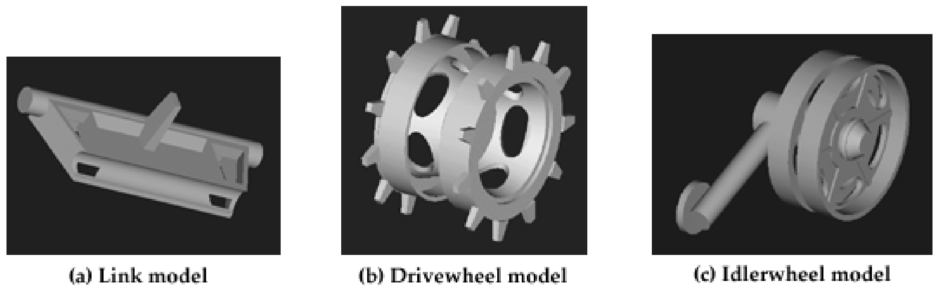

Fig. 23. B-reps of some solids from the Bradley fighting vehicle

generalized prisms and surfaces of revolution of degrees 6 or more. A few of the primitives are composed of Bézier surfaces of degree as high as 12. Most of the CSG trees have heights ranging between 6 and 12 and some of them are as high as 30. Table 1 shows the performance of the sequential algorithm on some solids from this model (see Figure 22). The column with running time is broken into four parts: the bounding box and linear programming, surface-surface intersection, ray-shooting and total. The final column indicates the number of trimmed patches that the final model has.

The model of the Bradley fighting vehicle was obtained from Army Research Laboratories. It is composed of more than 8500 solids each consisting of about 5-8 Boolean operations. The primitives in the Bradley are solids like spheres, cylinders, ellipsoids and tori whose B-reps can be represented by biquadric (degree 2 $\times$ 2) Bézier patches. We present the performance of our sequential and parallel algorithms on three of the solids in the Bradley fighting vehicle.

- **Link model:** It consists of 16 Boolean operations and the B-rep contains 76 trimmed Bézier patches. Figure 23(a) shows the model. The graph in Fig-

Model	# of CSG opns.	Running time (in secs.)				# of patches (in B-Rep)
		BB & LP test	SSI	Ray-shooting	Total	
Link	16	1.3	26.3	9.6	47.81	76
Drivewheel	44	5.8	54.3	27.1	97.23	289
Idlerwheel	48	5.1	59.8	28.9	106.93	235

Table 2. Performance of our sequential algorithm on parts of the Bradley model (Figure 23)

Model	Total running time (in secs.)				
	1 proc.	2 proc.	3 proc.	4 proc.	5 proc.
Link	51.95	30.88	26.93	20.55	23.44
Drivewheel	102.32	77.39	53.39	49.02	35.67
Idlerwheel	112.40	74.51	58.96	46.10	44.23

Table 3. Performance of our parallel algorithm on parts of the Bradley model (Figure 23)

ure 24 shows the performance of our system on varying number of processors. It can be seen that the performance becomes worse when we go from four to five processors. Since this is not a very complex model, the setup costs of using five processors outweigh the benefit of parallelism.

- **Drivewheel model:** This model is constructed using 44 Boolean operations. The B-rep is shown in Figure 23(b) and consists of 289 trimmed Bézier patches.
- **Idlerwheel model:** The B-rep of the idlerwheel (composed of 235 trimmed Bézier patches) is shown in Figure 23(c) and took 48 Boolean operations to generate. Again increasing the processor count reduces the running time because of complexity of the model.

Table 2 and Table 3 shows the performance of our sequential and parallel algorithm on the parts of the Bradley model shown in Figure 23 respectively.

12. Public Domain Release

BOOLE is currently available for download at http://www.cs.unc.edu/~geom/CSG/boole.html. Our implementation runs on single processor architectures like SGI Maximum Impact and Sun-Solaris, as well as a parallel version of the algorithms on shared memory multiprocessor architectures like SGI Onyx. The entire implementation of the system is in C.

Using BOOLE: Here we document

- **Fundamental data structure, TRIM_PATCH_INFO:** A solid is represented by an array of TRIM_PATCH_INFO structures, each of which defines a trimmed Bezier patch. Each trimming curve is represented by spline curves

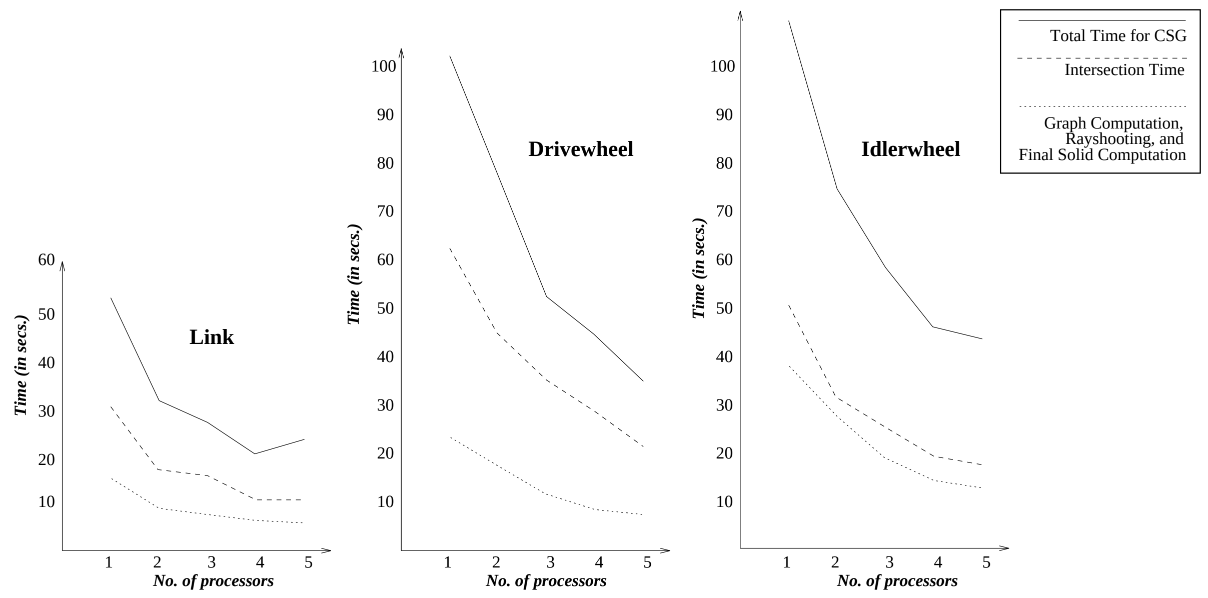

Fig. 24. Performance of our parallel algorithm as a function of processor count

in the domain of the patch as well as in 3-space. The spline curve provide the interface. The algebraic and the piecewise linear curve are used for internal computations only.

- **Boolean operations using operate():** The high-level routine in BOOLE is operate(). It performs a boolean operation on the two given solids. The argument "operation" can take one of the following integer values:
  - 1: Union
  - 2: Intersection
  - 3: Difference

  A step size must be specified for tracing the intersection curve between Bezier patches. We have found that 0.03 works sufficiently most of the time. The stepsize is in parametric space, and is not related to the size of the model.

- **Generating geometric primitives:** We have included some routines to generate TRIM_PATCH_INFO objects (B-reps) of some common geometric primitives. For example, new_cone() generates a truncated cone given a center point for the base, an axis vector (whose length is immaterial), the length of the cone, and the radii of the two disks. Other examples of routines that generate primitives are new_cylinder(), new_ellipsoid(), new_sphere() and new_torus().

13. Conclusion

Evaluating Boolean set operations of sculptured solid objects is one of the most powerful facilities available in a solid modeler. In modelers based on boundary representations, the Boolean set operation algorithm is also technically one of the most demanding component. A significant portion of the complexity is due to the computation and representation of intersection curves between free-form surfaces. Apart from the algebraic and geometric difficulties, a convenient representation of the intersection curve is essential to effectively compute the boundary. Another important issue in this context is that of robustness on models of large scale. Our experience with the Bradley fighting vehicle and submarine model shows that extremely large CAD models are designed using Boolean set operations for physical analysis and model verification. Individual solids are generated using a large number of successive Boolean operations. In such cases, systematically dealing with the growth of errors (sharply bounding the maximum errors) due to finite precision arithmetic is very difficult and impractical, especially for solids with curved primitives. The best way to deal with such problems is to combine numerically stable algorithms with the use of exact arithmetic check-pointing routines that control the growth of error and are able to identify and resolve degenerate and nearly degenerate situations.
In this paper, we have described a complete implementation of a system to evaluate B-reps of Boolean combinations of sculptured solids. It employs a combination of symbolic and numeric methods to compute the B-reps accurately and efficiently. The input to our algorithm is a CSG tree that describes the solid as a Boolean expression of primitive solids. The choice of the set of primitive solids is arbitrary as long as they can be represented as a piecewise collection of parametric surface patches. Our portable implementation , called BOOLE, has been successfully applied to generate the boundary representations of industrial models composed of thousands of Boolean set operations.

Acknowledgements

We would like to thank Mike Hohmeyer and Atul Narkhede for providing their implementations of linear programming and polygon triangulation respectively. Thanks also to Ken Fast, Greg Angelini and Jim Boudreaux at Electric Boat and Mike Muuss at Army Research Labs for providing their CSG models for me to use.

References

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

2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

3. C.L. Bajaj, C.M. Hoffmann, J.E.H. Hopcroft, and R.E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5:285–307, 1988.

4. R.E. Barnhill and S.N. Kersey. A marching method for parametric surface/surface intersection. *Computer Aided Geometric Design*, 7:257–280, 1990.

5. B. Baumgart. A polyhedron representation for computer vision. In *National Computer Conference, AFIPS Conf. Proc.*, pages 589–596, 1975.

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

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

8. S. A. Cameron. A study of the clash detection problem in robotics. *IEEE Conference on Robotics and Automation*, pages 488–493, 1985.

9. M. S. Casale. Free-form solid modeling with trimmed surface patches. *IEEE Computer Graphics and Applications*, pages 33–43, January 1987.

10. 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.

11. M.S. Casale and E.L. Stanton. An overview of analytic solid modeling. *IEEE Computer Graphics and Applications*, 5:45–56, February 1985.

12. K. Chan. *Solid Modelling of Parts with Quadric and Free-form Surfaces.* PhD thesis, University of Hong Kong, 1987.

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

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

15. E. Driskill and E. Cohen. Interactive desigb, analysis, and illustration of assemblies. In *Proc. of 1995 Symposium on Int. 3D Graphics*, pages 27–34, 1995.

16. Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

17. J. L. Ellis, G. Kedem, T. C. Lyerly, D. G. Thielman, R. J. Marisa, J. P. Menon, and H. B. Voelcker. The raycasting engine and ray representations. In *Proceedings of Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267, 1991.

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

19. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1990.

20. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1993.

21. R.T. Farouki. The characterization of parametric surface sections. *Computer Vision, Graphics and Image Processing*, 33:209–236, 1986.

22. R.T. Farouki and J.K. Hinds. A hierarchy of geometric forms. *IEEE Computer Graphics and Applications*, 5:51–78, May 1985.

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

24. B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51. Springer-Verlag, Berlin, 1977.

25. A. Geisow. *Surface Interrogations*. PhD thesis, School of Computing Studies and Accountancy, University of East Anglia, 1983.

26. G. Georgiannakis and C. Houstis et. al. Description of the adaptive resource management problem, cost functions and performance objectives. Technical Report TR130, The Institute of Computer Science, Foundation for Research and Technology, 1995.

27. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. *Proc. Supercomputing '95*, 1995.

28. N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, PA, 1996.

29. R. C. Hillyard. The build group of solid modellers. *IEEE Computer Graphics and Applications*, 2:43–52, 1982.

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

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

32. M.E. Hohmeyer. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1(4):473–490, 1991. Special issue on Solid Modeling.

33. M.E. Hohmeyer. *Robust and Efficient Intersection for Solid Modeling.* PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.

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

35. G.E.M Jared. Synthesis of volume modeling and sculptured surfaces in build. In *CAD84, Computers in Design Engineering Conference Proceedings*, pages 481–495, 1984.

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

37. Y.E. Kalay. Modeling polyhedral solids bounded by multi-curved parametric surfaces. *ACM IEEE Nineteenth Design Automation Conference Proceedings*, pages 501–507, 1982.

38. S. Katz and T.W. Sederberg. Genus of the intersection curve of two rational surface patches. *Computer Aided Geometric Design*, 5, 1988.

39. 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.

40. F. Kimura and Geomap-III. Designing solids with free-form surfaces. *IEEE Computer Graphics and Applications*, 4:58–72, 1984.

41. G.A. Kriezis, N.M. Patrikalakis, and F.E. Wolter. Topological and differential equation methods for surface intersections. *Computer-Aided Design*, 24(1):41–55, 1990.

42. S. Krishnan. *Efficient and Accurate Boundary Evaluation Algorithms for Boolean Combinations of Sculptured Solids.* PhD thesis, Department of Computer Science, University of North Carolina, Chapel Hill, December 1997.

43. S. Krishnan, M. Gopi, D. Manocha, and M. Mine. Interactive boundary computation of boolean combinations of sculptured solids. In *Proc. of Eurographics*, 1997.

44. S. Krishnan and D. Manocha. Computing boolean combinations of solids composed of free-form surfaces. Manuscript [submitted for publication].

45. 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.

46. S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, 1997.

47. S. Krishnan and D. Manocha. Symbolic-numeric methods of loop detection for curve and surface interrogations. In *Journal of Symbolic Computation: Special Issue on Symbolic-Numeric Algebra for Polynomials [In Review]*, 1997.

48. D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. *ACM Computer Graphics*, 20:161–170, 1986.

49. L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

50. D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

51. D. Manocha and S. Krishnan. Algebraic pruning: A fast technique for curve and surface intersections. *Computer Aided Geometric Design*, 20:1–23, 1997.

52. M. Mantyla. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5:1–29, 1986.

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

54. M. Mantyla and M. Ranta. Interactive solid modeling in hutdesign. In *Proceedings of Computer Graphics, Tokyo*, 1986.

55. M. Mantyla and M. Tamminen. Localized set operations for solid modeling. In *Computer Graphics*, volume 17, pages 279–288, 1983.

56. D. J. Meagher. The solids engine: a processor for interactive solid modeling. In *Proceedings of Nicograph*, 1984.

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

58. A. Narkhede and D. Manocha. Fast polygon triangulation based on seidel's algorithm. In A. Paeth, editor, *Graphics Gems V*, pages 394–397, Academic Press, 1995.

59. B.K. Natarajan. On computing the intersection of b-splines. In *ACM Symposium on Computationl Geometry*, pages 157–167, 1990.

60. N. Okino, Y. Kakazu, and H. Kubo. *TIPS-1: Technical Information Processing System for Computer Aided Design and Manufacturing*. Computer Languages for Numerical Control, J. Hatvany, ed., North Holland, Amsterdam, 1973.

61. A. Paoluzzi, M. Ramella, and A. Santarelli. Un modellatori geometrico su rappresentazioni triango-alate. Technical Report Rept. TR 13.86, Department of Inf. and Systems, University of Rome, Italy, 1986.

62. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1990.

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

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

March 1982.

65. 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.

66. J. Rossignac, A. Megahed, and B.D. Schneider. Interactive inspection of solids: cross-sections and interferences. In *Proceedings of ACM Siggraph*, pages 353–60, 1992.

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

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

69. T. Satoh. Boolean operations on sets using surface data. In *Proceedings of Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 119–127, 1991.

70. T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.

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

72. R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

73. R. Seidel. A simple and fast randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory & Applications*, 1(1):51–64, 1991.

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

75. M. S. Tawfik. An efficient algorithm for csg to b-rep conversion. In *Proceedings of Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 99–108, 1991.

76. T. Varady and M.J. Pratt. Design techniques for the definition of solid objects with free-form geometry. *Computer Aided Geometric Design*, 1(3):207–225, 1984.

77. H. B. Voelcker. An introduction to padl: Characteristics, status, and rationale. Technical Report Research Memo. #22, University of Rochester, 1974. Production Automation Project.

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

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

80. M. Wesley. A geometric modeling system for automated mechanical assembly. *IBM Journal of Research and Development 24*, pages 64–74, 1980.

81. S. Whitman. Dynamic load balancing for parallel polygon rendering. *IEEE Computer Graphics and Applications*, 14(4):41–48, 1994.

82. J.H. Yang and J. Anderson. Fast, scalable synchronization with minimal hardware support. In *ACM symposium on Principles of Distributed Computing*, pages 171–182, 1993.