Representation and Fast Display of Complex CSG Models*

Subodh Kumar **Shankar Krishnan** **Dinesh Manocha** **Atul Narkhede**

{kumar,krishnas,manocha,narkhede}@cs.unc.edu
Department of Computer Science
University of North Carolina
Chapel Hill NC 27599

Abstract:
We present a system for fast and accurate display of CSG (constructive solid geometry) models. Such models have as primitives, polyhedra and solids whose boundaries can be represented using rational spline surfaces. As part of pre-processing, we compute the B-rep (boundary representation) from the CSG tree and represent the resulting solid using trimmed spline surfaces. No assumptions are made on the number of primitives or the degree of the primitives in the CSG tree. We also present an accurate, efficient and multiresolution representation of the high degree trimming curves. Given a trimmed spline model, we tessellate it into polygons as a function of the viewing parameters and render it using visibility computations and coherence between successive frames. The choice of a multiresolution representation of the model and the trimming curves is *fundamental* to the fast performance of the system. The system has been used to convert parts of a submarine storage and handling system model represented as more than $2,000$ CSG trees. The B-rep consists of more than $30,000$ trimmed spline surfaces and is displayed at interactive rates on the Pixel-Planes 5 graphics system.

1 Introduction

Rational spline surfaces are routinely used to represent solids in engineering design. Splines also find application in modeling body parts for medical imaging and molecules for drug design, among other things. In addition, solid models composed of polyhedra, spheres, cones, tori, prisms, solids of revolution etc. are widely used in CAD/CAM, virtual reality, engineering simulation and animation applications [Hof89, RR92]. All these solids can also

*This research is supported by a Sloan Foundation, NSF Grant CCR-9319957, ARPA Contract DABT63-93-C-0048, ARPA ISTO Order No. A410, NSF Grant No. MIP-9306208, NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, NSF Prime Contract No. 8920219 and Office of Naval Research Contract, ONR N00014-94-1-0738.

be easily represented in terms of rational spline surfaces. These solids and their Boolean combinations, i.e. union, intersection and difference, are used to generate large-scale models like those of airplanes, ships, automobiles, submarines etc. These models are typically represented as thousands of CSG trees, where some of the trees are defined with more than 30 Boolean operations on primitives corresponding to surfaces of parametric degree five or six. Most of the current solid modelers are unable to compute a multiresolution and accurate B-rep for these models and therefore represent them in terms of millions of polygons. Current graphics systems are not able to render such *complex models* at *interactive frame rates*. Thus there is a wide body of application that benefits from accurate CSG display based on a spline visualizer. Immersive design and design validation are the target applications of such systems.

We present algorithms and systems to display CSG and spline models at interactive frame rates. The main components of the system are:

- *Solid Modeler*: Computing accurate B-reps from the CSG models and representing them in terms of trimmed spline models.
- An *accurate, efficient and multiresolution representation* of the high degree trimming curves.
- *Display System*: Fast and accurate rendering of models composed of trimmed spline surfaces.

Our solid modeler can accurately compute Boolean combinations of solids composed of more than 30 high degree curved primitives and their CSG combinations. Its application to parts of a submarine storage and handling system composed of about $2,000$ CSG trees (with depth ranging from 3 to 36) resulted in a B-rep consisting of more than $30,000$ trimmed surfaces, which can be rendered at interactive rates on Pixel-Planes 5 [Fe89] by our display system.

1.1 Prior Work

There is a great deal of work in the modeling and rendering literature related to model conversion, displaying large data sets and multiresolution modeling. We categorize it into four types:

Direct Rendering of CSG Models: Some graphics systems include capabilities of direct rendering of CSG models [Kea94, GMTF89]. However, they either restrict the number of CSG operations or the degrees of the primitives or are not capable of rendering this class of complex models. Algorithms based on enhanced Z-buffer often require multiple rendering passes along with fast rasterizing systems for such primitives [RW90]. Other techniques and systems for direct rendering are based on ray-casting [KE89, Rot82] and are currently not fast enough.

CSG to B-rep Conversion: There is considerable literature on computing B-rep's from CSG solids defined using polyhedral primitives, as surveyed in [Hof89, RR92]. A number of

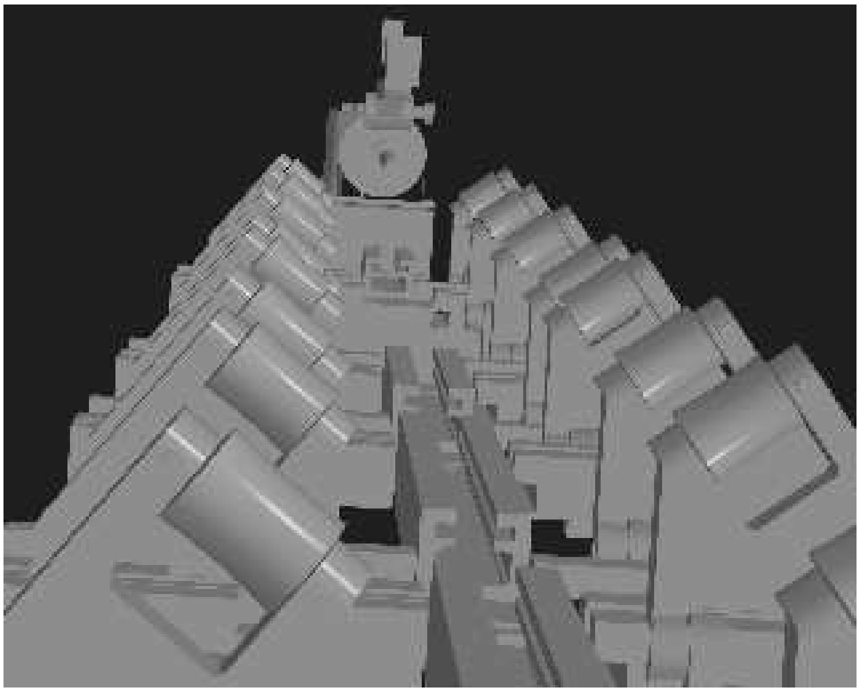

Figure 1: Pivot: Part of the Submarine Storage and Handling System Model

techniques to improve the robustness of such systems have been proposed as well [Seg90]. However, no such robust algorithms and systems are known for solid models composed of curved or spline primitives [Hof89, RR92]. The main problem lies in accurate, robust and efficient computation of the intersection of spline surfaces. Surface intersection is an active area of research and some of the recent papers have proposed algorithms to compute all components and piecewise linear or spline approximations to the intersection curves based on tracing methods [SN91, Hoh91, Hof89]. However, these representations are either inaccurate or inconsistent for Boolean operations on complex models. Therefore, many of the current solids modelers use polyhedral approximations of these primitives and compute the B-rep's of the resulting solids in terms of polygons.

Visualization of Large Data Sets: Current high-end graphics systems are not able to render polygonal representation of complex models described using millions of polygons at interactive frame rates. Many algorithms based on visibility preprocessing and multiresolution modeling have been proposed for faster display of such models [Tel92, Fun93, SZL92, Tur92, RB92, HDD$^+$93, DLW93, GKM93]. However, visibility methods have been shown useful for densely occluded polyhedral environments only [Tel92]. Most of the current simplification algorithms are restrictive and no efficient algorithms with guaranteed error bounds are known for general topologies [HG94].

Rendering Spline Surfaces: Many algorithms are known in the literature for tessellating spline surfaces and rendering the resulting polygons [AES91, RHD89, LC93, KML95, BR94]. In particular, Rockwood et. al. [RHD89] presented the first real time algorithm for rendering trimmed surfaces. Given a NURBS model, [RHD89] decompose it into a series of trimmed Bézier surfaces, split the trimming curves into monotonic regions and triangulate the resulting regions. However, it is not fast enough for complex models and its implementation as part of SGI's GL library can render surfaces composed of at most a few hundred trimmed Bézier surfaces at interactive rates on an SGI Onyx [KML95].

1.2 Overview

Given a CSG model, our solid modeler represents each primitive as a collection of Bézier surfaces, performs accurate Boolean operations and represents the B-rep as a collection of trimmed Bézier surfaces. It represents the trimming curves as piecewise algebraic space curves along with bounding volumes. It makes use of algorithms based on surface intersection, polygon triangulation, domain partitioning and ray-shooting to compute the B-reps. Given a collection of trimmed Bézier surfaces, our display system tessellates them into triangles and renders them on the graphics pipeline. It makes use of view-frustum culling and back-patch culling for visibility computations and dynamically tessellates the surface as a function of viewing parameters. As opposed to decomposing the trimming curves into monotonic segments, our system uses an algorithm for triangulating non-convex polygons. In particular, it makes use of frame to frame coherence using *incremental triangulation*. On parallel graphics systems, the display system makes use of multiple processors for tessellation, minimizes communication between processors and load balances the work between *polygon generation* and *polygon rendering*.

The multiresolution representation of the B-reps in terms of trimmed curves and surfaces is fundamental to the performance of the overall system. As compared to earlier systems for rendering such models, it has the following advantages:

- **Fidelity:** A high degree of fidelity of display is essential for any meaningful visualization and design validation. Our B-rep's for CSG models are computed more accurately than by the modelers using polygonal representation for the curved primitives and the final solids. Besides rendering, it is useful for other applications like collision detection.

- **Rendering:** Our algorithms based on visibility culling and dynamic tessellation generate fewer polygons for the spline models. Furthermore, we can easily generate on-line any *level-of-detail* with correct topology using incremental computations. This is in contrast with the difficulty of computing multiresolution models for large polygonal datasets of arbitrary topology and *visible artifacts* introduced due to few and discrete levels of detail. This results in *better and faster* display.

- **Memory:** The memory requirements for our B-reps are about one or two order of magnitudes lower than polygonal models and their multiresolution representations. On the other hand our display system needs more processing power for on-line triangulation of the multiresolution representations.

The rest of the paper is organized in the following manner. Section 2 presents the notation used in the rest of the paper and the multiresolution representation for the trimming curves. Section 3 describes model generation, representation and the underlying algorithms. We present the display system in Section 4 and discuss the overall implementation and performance on parts of the submarine storage and handling system model in Section 5.

2 Model Representation

We use bold face letters to represent vector quantities and surfaces and *italic* fonts to represent curves in 2D and 3D. Lower case letters are used for representing points and curves in a plane and upper case letters for points, curves and surfaces in 3-space. A tensor product Bézier surface, $\mathbf{F}(s,t)$, is represented as: $\mathbf{F}(s,t) = \Sigma_{i=0}^{m}\Sigma_{j=0}^{n}\mathbf{V}_{ij}B_i^m(s)B_j^n(t)$ where $\mathbf{V}_{ij} = \langle w_{ij}x_{ij}, w_{ij}y_{ij}, w_{ij}z_{ij}, w_{ij}\rangle$ are the *control points* of the patch in homogeneous coordinates and $B_i^m(s) = \begin{pmatrix} m \\ i \end{pmatrix} s^i(1-s)^{m-i}$ is the Bernstein polynomial [Far93]. The domain of the surface is defined on the unit square $0 \leq s,t \leq 1$ in the (s,t) plane. Trimmed surfaces are defined on a subset of $[s,t] \in [0,1] \times [0,1]$ domain using *trimming curves*, $t_1, t_2, \ldots, t_p$. The trimming curves are represented as piecewise linear curves, Bézier curves or piecewise algebraic space curves. Furthermore, each of them is an oriented curve, i.e. it has a starting point, $\mathbf{q}_0$ and an ending point, $\mathbf{q}_k$ (as shown in Fig. 3). Each trimming curve *trims out* the part of the surface that lies on its right. This is called the *handedness rule* of trimming [RHD89]. It does not allow self intersecting trimming curves. The region of the surface that is not trimmed out is also referred to as the *trimmed region*.

2.1 Representation of Solids

Each solid, $\mathbf{S}$, comprises of planar faces and trimmed Bézier surfaces. The data structure for the solid consists of a number of surfaces, representation of each surface and an *adjacency graph* (Fig. 4, $\Gamma(\mathbf{S})$, highlighting the connectivity between all the surfaces. Furthermore, each planar face is defined as an anti-clockwise ordering of the vertices and for each Bézier surface $\mathbf{F}(s,t)$, the cross-product $\mathbf{F}_s \times \mathbf{F}_t$ points to the outer normal. As a result, we can perform *local* in/out tests at any point on the boundary of the solid.

Each vertex, v_i, in the adjacency graph corresponds to a surface $\mathbf{F}_i$, defining the solid. Two vertices v_j and v_k are connected by an edge, iff the two surfaces $\mathbf{F}_j$ and $\mathbf{F}_k$ share a common boundary. The common boundary may be along the edges of a planar face or a boundary curve or trimming curve for a Bézier surface.

2.2 MultiResolution Representation of Trimming Curves

The intersection curves of two surfaces correspond to the vector equation $\vec{\mathbf{F}}(s,t) = \vec{\mathbf{G}}(u,v)$. This results in the following three scalar equations:

$$\begin{aligned} F_1(s,t,u,v) = X(s,t)\overline{W}(u,v) - \overline{X}(u,v)W(s,t) &= 0 \\ F_2(s,t,u,v) = Y(s,t)\overline{W}(u,v) - \overline{Y}(u,v)W(s,t) &= 0 \\ F_3(s,t,u,v) = Z(s,t)\overline{W}(u,v) - \overline{Z}(u,v)W(s,t) &= 0, \end{aligned} \tag{1}$$

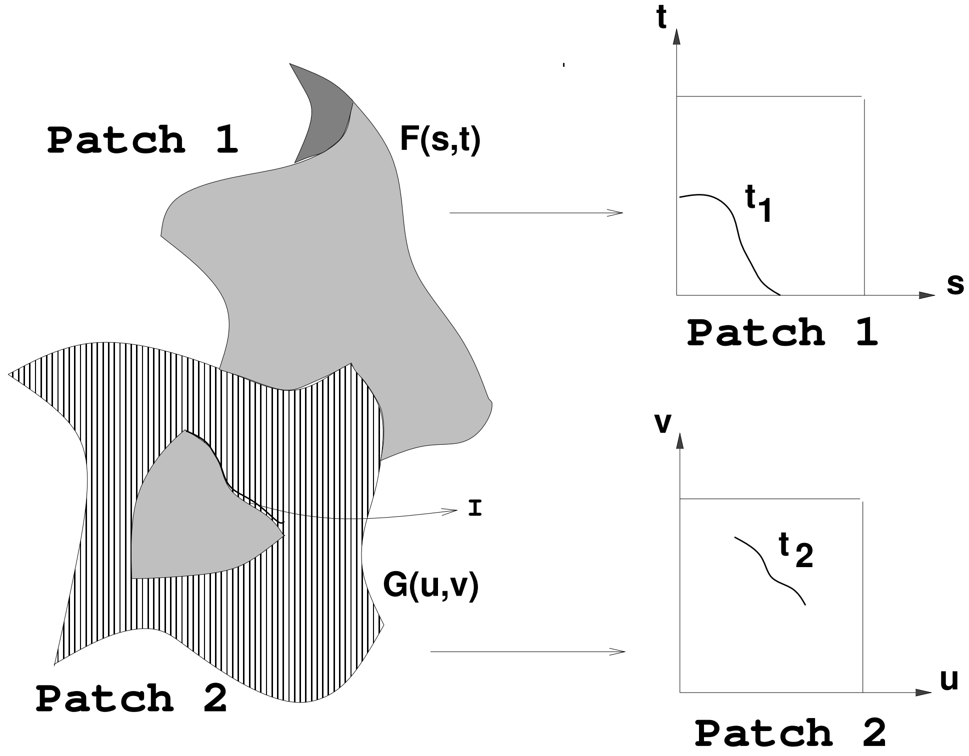

Figure 2: Intersection of two surfaces

where the domain of the variables is restricted to $0 \leq s, t, u, v \leq 1$. We use a recently developed algorithm for computing the intersection of rational parametric surfaces [KM94]. It is based on computing a start point on each component of the intersection curve and decomposing the domain such that each resulting region contains at most one component of the intersection curve. The algorithm computes a series of points on each component. Its performance is output sensitive and for solids composed of about 100 surfaces, it can compute all the components in about 20 seconds on an SGI Onyx. Its application to the representation of $\mathbf{F}(s,t)$ in Fig. 2 after a few intersections has been shown in Fig. 3 The trimming curves corresponding to the surface intersection are represented as a union of components (c_i's) of algebraic space curve. Each component consists of:

1. Start and end points $\mathbf{q_0}$ and $\mathbf{q_k}$.

2. Rectangular bounding box ($\mathbf{b}_i$) $[u_1, v_1] \times [u_2, v_2]$ in the domain of the surface bounding c_i.

3. An axis-aligned bounding box ($\mathbf{B}_i$) in the X, Y, Z space bounding the image of that component (I_i).

4. The surface parametrizations, $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$ and the equations, $F_1(), \ldots, F_3()$ in (1) describing the curve.

The incremental rendering algorithm in Section 4 needs a piecewise linear representation of the trimming curve as a function of viewing parameters. At the lowest level is the straight

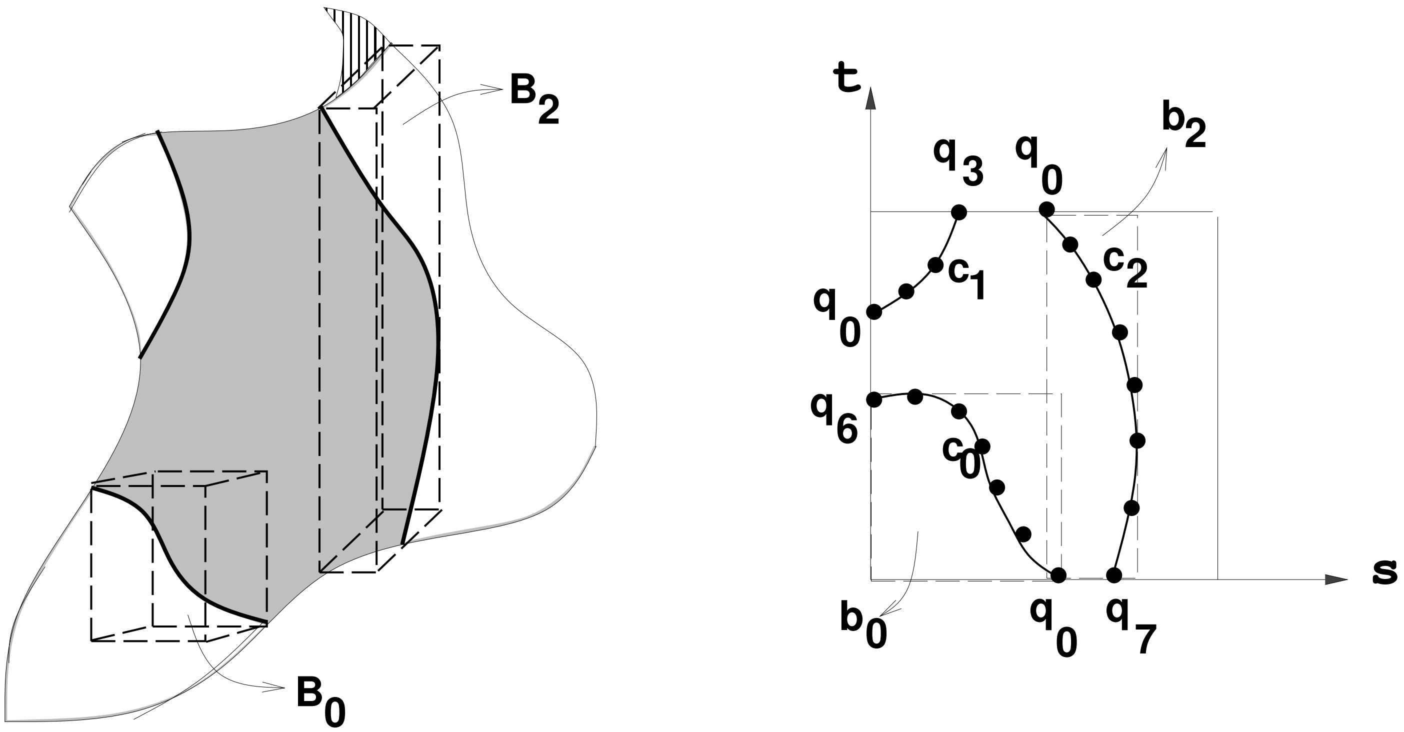

Figure 3: Representation of Trimming Curves

line joining $\mathbf{q_0}$ and $\mathbf{q}_n$. To generate more points on the curve, the algorithm simultaneously *traces* the trimming curve c_i in the domain and the intersection curve I_i on the surface. The **tracing algorithm** generates more points in the following manner:

1. Given a point $\mathbf{P} = \mathbf{F}(s_0, t_0) = \mathbf{G}(u_0, v_0)$ on the intersection curve and a step size Δ, compute the tangent to the curve at $\mathbf{P}$ using cross-products of the surface normals

2. Take a step size Δ along the tangent direction and let the new point be $\mathbf{P}_0 = (x_0, y_0, z_0)$.

3. Compute the closest point on the intersection curve from $\mathbf{P}_0$ using constrained optimization algorithm. The optimization function is defined as the sum of the squares of the distance from $\mathbf{P}_0$ to $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$, $\parallel \mathbf{P}_0 - \mathbf{F}(s,t) \parallel^2 + \parallel \mathbf{P}_0 - \mathbf{G}(u,v) \parallel^2$, and the constraints correspond to the equations $F_1(), \ldots, F_3()$. We transform it into an unconstrained problem using Lagrange's multipliers, use (s_0, t_0, u_0, v_0) as the initial guess and make a series of Newton steps towards the solution. If it does not converge in three or four iterations, we replace the step size by $\Delta/2$ and repeat the process.

This tracing step is slow enough that it is impractical to trace each trimming curve each frame. Due to coherence between frames, this step is not required very often. In fact for each curve of length ℓ in $\Re^3$ we compute off-line, $K.\ell$ points on the curve, where K is a user specified constant. At run time we evaluate more points as we need them. Further, when the required tessellation of the curve goes down, we retain the extra points subject to availability of memory. For most frames we just pick points on the (possibly densely) pretessellated curves. The points closest in parametric space of the curve to the required tessellant is picked.

3 Model generation

In this section, we describe the solid modeling system used for computing B-reps from CSG trees. Free–form surfaces have been used previously in modeling systems. The Alpha_1 [Coh83] and Geomod [Til83] modeling systems use NURBS to describe the B-rep. The primitives may correspond to polyhedra or solids whose boundaries can be represented in terms of rational spline surfaces. This is a fairly rich class of objects and can be used to model almost all the solids used in CAD/CAM and simulation applications. Currently our system handles primitives composed of tensor-product surfaces only, but can be easily extended to include solids defined using triangular surfaces. Furthermore, it makes the following set of *assumptions* for each Boolean operation:

- All primitives are *regularized* solids [Hof89] and all Boolean operations result in regularized solids.
- The intersection between all pairs of overlapping surfaces is well-conditioned and there are no degeneracies.

The system does not explicitly check whether these assumptions are satisfied. In some cases, it can detect these cases while running the algorithm.

The algorithm for B-rep computation is based on depth first traversal of the CSG tree and computing the B-rep of solids at each intermediate node of the tree. The algorithm for Boolean operation between a pair of solids involves trimmed surface intersection, ray-shooting, adjacency graph computation and surface normal orientation of the resulting solid. In this section, we give a brief overview of the algorithm and discuss its implementation and performance on a number of solids.

3.1 Surface Intersection

A basic operation in the computation of B-reps from a CSG model is the intersection of two surfaces. Given two Bézier surfaces, $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and $\mathbf{G}(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))$, represented in homogeneous coordinates, the intersection curve is defined as the set of common points in 3-space. However, the degree of the intersection curve can be as high as $4m^2n^2$ for two $m \times n$ tensor-product surfaces and the curve cannot be exactly represented as a Bézier curve (for most cases) [Hof89]. Typical values of m and n are two or three and can be as high as 10 or 12, in practice. Furthermore, the intersection curve may consist of multiple components, closed loops, singularities etc., which add to its geometric complexity. A number of algorithms based on subdivision methods, lattice evaluation and marching methods [SN91, Hoh91, Hof89, KM95a] are known. These algorithms compute piecewise linear or spline approximations of the intersection curve. However, when it comes to computing the B-reps of CSG models defined using a series of Boolean operations, these representations suffer from two important problems:

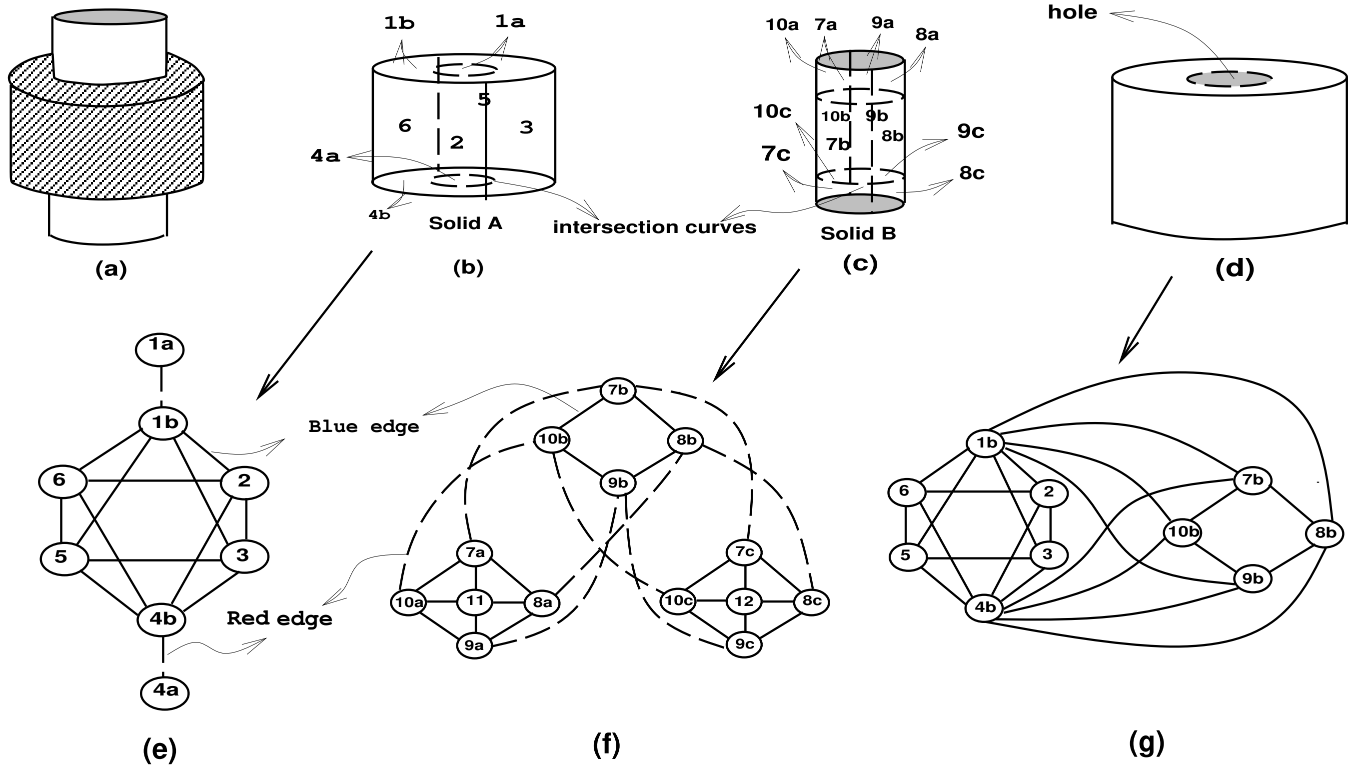

Figure 4: Adjacency Graph Computation for a Difference Operation

1. **Robustness and accuracy**: The algorithm for Boolean operations repeatedly computes intersection of trimmed surfaces with rays in space and other trimmed surfaces. With approximate representations, the errors at each level of the tree propagate and it becomes difficult to compute B-reps accurately and robustly for models defined with more than seven or eight CSG operations.

2. **Inconsistent Representation:** The rendering algorithm based on dynamic tessellation needs a multiresolution representation of the trimming curves. Furthermore, the trimming curves (e.g. curves t_1 and t_2 in Fig. 2) corresponding to the intersection curve (curve I in Fig. 2) should be consistently tessellated into piecewise linear segments in the domains of surface $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$, to prevent cracks. An off-line piecewise linear approximation does not provide a multiresolution representation. Generating a densely tessellated linear approximation using a small step size can lead to data proliferation. It is non-trivial to compute consistent spline approximations to t_1 and t_2 such that their images correspond to the same curve and there are no cracks while rendering.

3.2 Intersection between Trimmed Surfaces

Given two solids, $\mathbf{S}_1$ and $\mathbf{S}_2$, with m and n surfaces, respectively, the algorithm for Boolean operations initially computes the intersection curve between their boundaries. This is based on computing the intersection of all possible overlapping surfaces. In the worst case, all the mn pairs are intersecting, but that is rarely the case. To speed-up the computation, the algorithm first checks their bounding volumes and convex hulls for intersection. It encloses each surface with an axis-aligned bounding box, projects the bounding box along the X, Y and Z axis, sorts the resulting intervals and computes the overlapping bounding boxes based on them. Each Bézier surface is contained in the convex hull of its control points [Far93]. For each pair of overlapping bounding boxes, the algorithm checks whether the convex hulls of their control points intersect. This is reduced to a linear programming problem in three dimensions and solved using Seidel's incremental linear time algorithm [Sei90].

Let $\mathbf{F}(s,t)$ be a trimmed surface and its trimmed subset in the domain $0 \le s,t \le 1$ be bounded by curve segments $t_1, t_2, \ldots, t_k$. To compute its intersection with another trimmed or non-trimmed surface $\mathbf{G}(u,v)$, it initially treats each parameterization as a non-trimmed surface and computes all the components of the intersection curves in the domain $0 \le s,t,u,v \le 1$. Let these components be denoted as $c_1, c_2, \ldots, c_p$, as shown in Fig. 5. Let the corresponding components in the domain of $\mathbf{G}(u,v)$ be $\overline{c}_1, \overline{c}_2, \ldots, \overline{c}_p$. Given these components, it computes a partition of the domain of $\mathbf{F}(s,t)$ using the following algorithm:

1. Compute a dense linear approximation for each algebraic space curve and a polygonal approximation (P) of the domain of $\mathbf{F}(s,t)$. Let Q be the corresponding polygonal approximation of the domain of $\mathbf{G}(u,v)$. Compute a triangulation of the domains based on Seidel's algorithm [Sei91].

2. Using a dense linear approximation of c_i and $\overline{c}_i$, compute their maximal subsets that lie within P and Q. Approximations to the intersection points (with either P or Q) are computed between the resulting line segments. Fig. 5 shows this process on the domain of $\mathbf{F}(s,t)$.

3. Each point of intersection is used as a starting guess for the accurate intersection between the algebraic representation of the trimming curves. The actual intersection is represented as a solution of six algebraic equations in six unknowns and computed using Newton's method. The accurate solution is used to split the intersection curve and the trimming curves [KNM95].

The new trimming curves arising from intersection with various surfaces are *merged* together and eventually the domain of intersecting surfaces is partitioned into two or more *regions*. The boundaries of each component are composed of portions of t_i's and c_i's.

3.3 Computation of the New Solid

The location of a point with respect to the solid (in/out) is a fundamental operation in the computation of the new solid. We make sure that the point is not on the boundary of the solid. Given a point $\mathbf{P}$, we shoot a semi-infinite ray from the point in any direction and compute all its intersections with the boundaries of the solid. If the number of intersections is odd the point is inside the solid, otherwise it is outside. The intersection of rays with trimmed Bézier surfaces is computed using eigenvalue methods [MD94] and checking whether

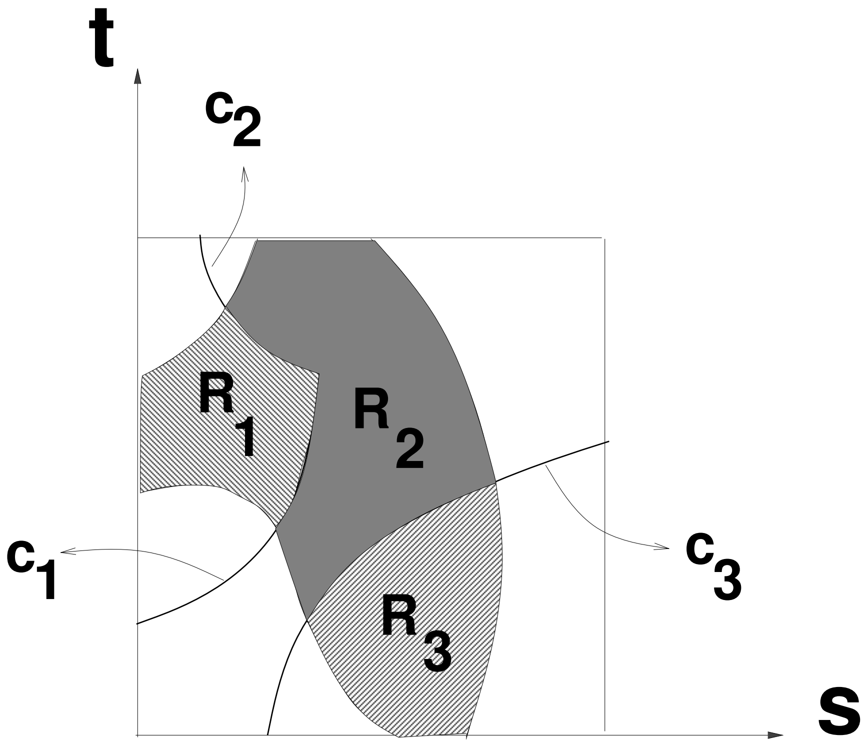

Figure 5: Partitioning the domain based on intersection

the resulting intersection point lies inside the trimmed domain is based on Seidel's algorithm [Sei91].

The intersection curves between the boundaries of the two solids partition the surfaces into multiple regions. These regions have the property that all points in their interior are either inside or outside the other solid. Further, it is easy to show that two *adjacent* regions, separated by the intersection curve, cannot *both* lie inside or outside. Depending on the Boolean operation, the regions composing the B-rep of the solid are chosen as:

- *Union:* A region of $\mathbf{S}_1$ is part of the new solid if it lies outside $\mathbf{S}_2$ and vice-versa for the regions of $\mathbf{S}_2$.
- *Intersection:* A region of $\mathbf{S}_1$ is part of the new solid if it lies inside $\mathbf{S}_2$ and vice-versa for the regions of $\mathbf{S}_2$.
- *Difference:* A region of $\mathbf{S}_1$ is part of the new solid if it lies outside $\mathbf{S}_2$. A region of $\mathbf{S}_2$ is part of the new solid if it lies inside $\mathbf{S}_1$.

We make use of adjacency graphs of the two solids, $\Gamma(\mathbf{S}_1)$ and $\Gamma(\mathbf{S}_2)$. The partitioning of surfaces into multiple regions induced by the intersection curves changes the structure of the adjacency graphs. For example, in Fig. 4(e), the vertex 1 corresponding to a surface in solid $\mathbf{S}_1$ is split into regions corresponding to $1(a)$ and $1(b)$. New adjacencies are introduced between the vertices due to intersection curves. We refer to the new edges as *red* edges and the original set of edges as *blue* edges. All the red edges in Fig. 4(e) and 4(f) are shown

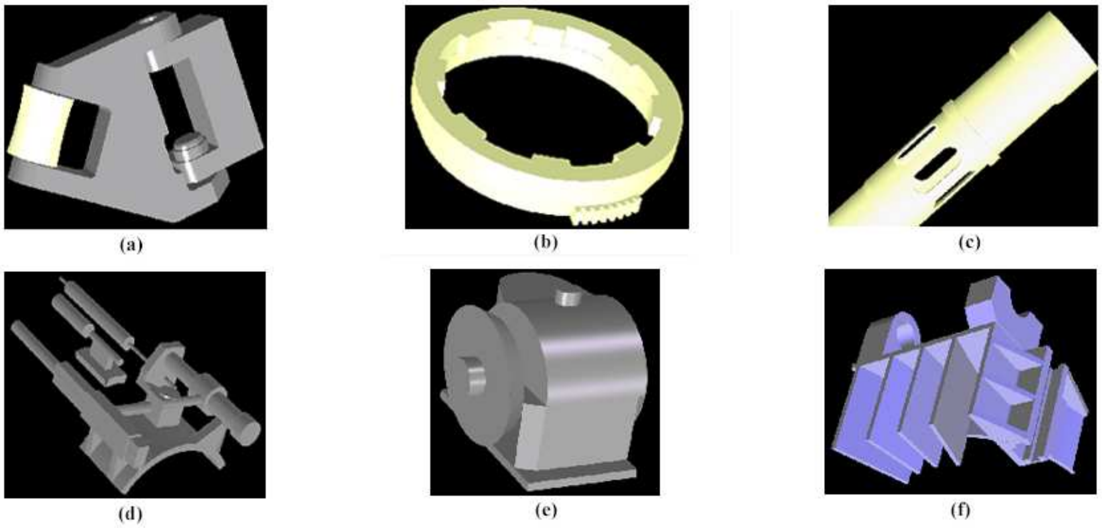

Figure 6: Application of the Algorithm and System to Different Solids

using "dashed" lines. After computing the new vertices and edges, the algorithm computes all connected components of the graph based on blue edges only. For example, the subgraph consisting of vertices $7a, 8a, 9a, 10a, 11$ represents one connected component in Fig. 4(f). Each connected component of one solid lies completely inside or outside the other solid. We perform ray shooting tests on one of these components. Based on the result, we propagate it to the rest of the graph such that no two adjacent components have the same result. The adjacency graph of the solid resulting from the difference operation is shown in Fig. 4(g).The new edges between these solids correspond to pairs of intersecting surfaces between $\mathbf{S}_1$ and $\mathbf{S}_2$ (e.g. 1b and 7b).

3.4 Implementation Details

Normal orientation: The rendering algorithm presented in Section 4 makes use of back-patch and back-face culling for fast display. As a result, the algorithm relies on the fact that all the surface normals are pointing outside the solid. Before performing any CSG operation, our system checks to see if all the primitives have outward facing normals. In case it is not, it initially computes a point in the interior of each surface and takes a small offset along the surface normal at that point. It uses ray-shooting to determine whether the resulting point is inside or outside the solid. If the point is inside the solid, the surface normals are reversed by reparameterization. Given two solids with surface normals oriented towards outside, the B-reps of their union and intersection preserve the normals. For a difference operation, however, the normals of the surfaces belonging to the second solid are reversed.

Overlapping Planar Surfaces: The algorithm presented above assumes that there are no degenerate intersections. However, many solids are often formulated by gluing one solid

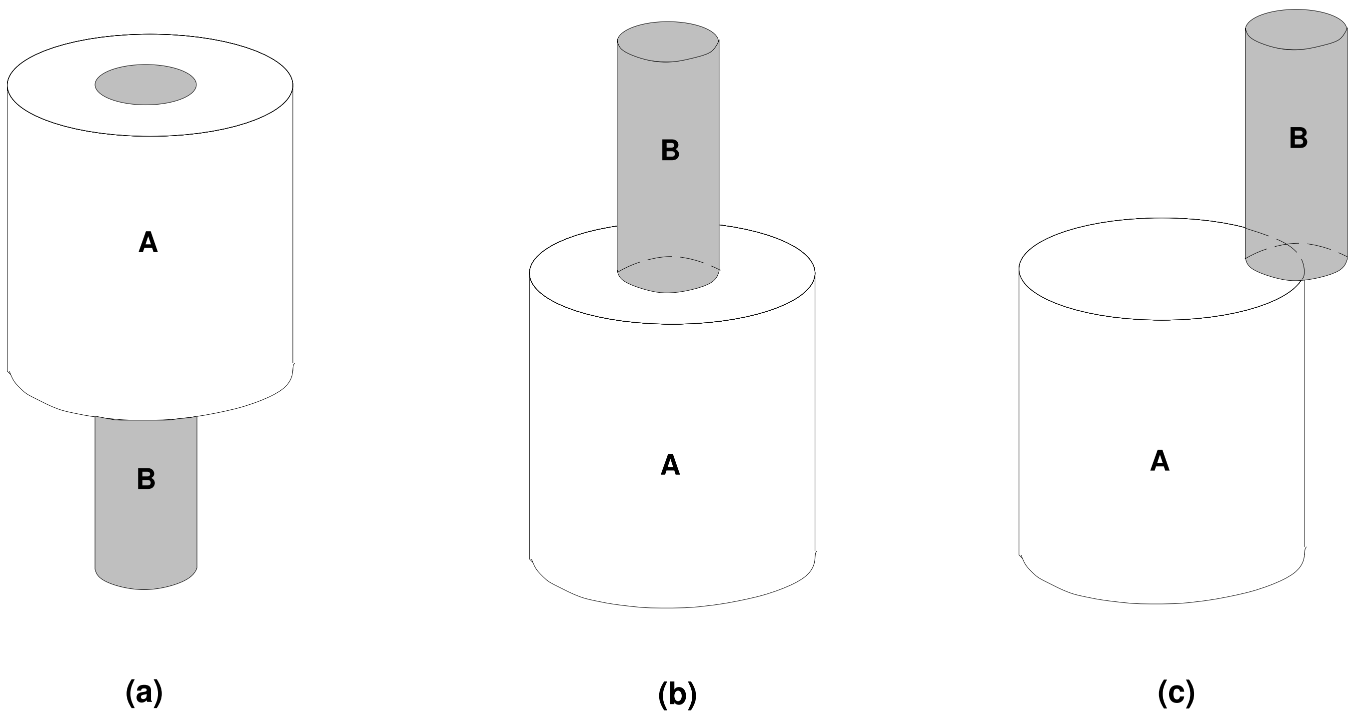

Figure 7: Degenerate Boolean Operation, Gluing

on top of another or as drilling a hole through one solid using the other. Most of these operations are restricted to planar surfaces and our system is able to handle such cases. Let A and B be the two planar surfaces overlapping. We transform the parameterization of B to that of A and compute the exact overlap using polygon intersection routines. This partitions the domains of A and B. Depending on the operation and the type of overlap, the relevant partitions of A and B are retained.

The complete details of the solid modeling system is described in [KNM95].

4 Display System

In this section, we describe the trimmed NURBS visualization system. This system subdivides NURBS into trimmed Bézier surfaces [KML95, Far93], which allow more efficient display than general splines. For trimmed Bézier surfaces we compute and store the smallest rectangular domain containing the trimmed region. Our algorithm tessellates the surface into triangles and renders them using the standard graphics pipeline. Special attention is paid to crack prevention between adjacent surfaces along the common boundaries and intersections curves. The system makes use of algorithms for back-patch culling (an extension of back-face culling for polygons to Bézier patches), bounds for polygonization and coherence between successive frames for general NURBS models, presented in [KML95]. In this section, we extend and improve the algorithms presented in [KML95] and [KM95b] to handle

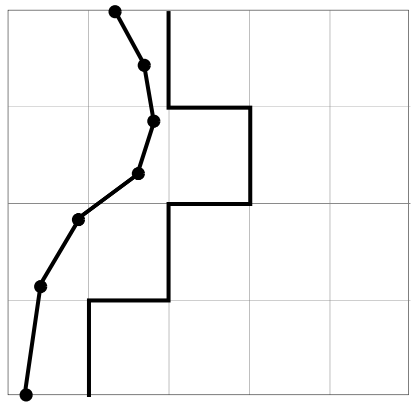

Figure 8: Trimming curve staircase

trimmed surfaces and B-reps of CSG models described in the previous section.

Given a Bézier surface and the viewing parameters, the algorithm tessellates the domain into cells based on *size criterion* and an estimation of curvature of the surfaces. It ensures that the image of each cell is smaller than a user specified tolerance, TOL, in the screen space ([KML95]) and therefore, we obtain a smooth image after Gouraud or Phong shading of the polygons corresponding to cells. This way we can achieve the required degree of fidelity in the display. To speed up the display process, the algorithm incrementally adds or deletes triangles as a function of the viewing parameters. The algorithm for trimmed surfaces traces the trimming curve on the grid of iso-parametric u and v lines in the parametric domain [KM95b].

In brief, the rendering algorithm for trimmed patches is as follows:

1. Create rectangular cells from the uniform tessellation of the surface based on TOL for the surface.

2. **For each** trimming curve:

   (a) Compute the required tessellation for the curve $n_t(TOL)$.

   (b) Tessellate the curve into $n_t(TOL)$ straight line segments.

   (c) March along the piecewise linear curve segments to mark all the domain cells that are crossed by the curve. Since the trimming curves are tessellated into piecewise linear segments, there is no need to calculate the exact intersections of the high degree algebraic curves with the cell boundaries. We only need to:

      i. Identify the cells whose bounding edges are intersected by the curve segments.

      ii. For each bounding edge of the rectangular cell, maintain the order in which the segments cross that cell.

      iii. If the trimming curve is contained in a cell, it intersects no cell edges. These cells are still marked.

3. Triangulate each unmarked cell that lies in the trimmed region of the surface by adding a diagonal.

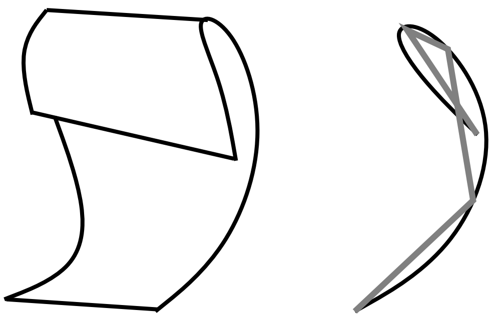

Figure 9: Topology violated by the tessellation. The picture on the left is a patch. The picture on the right shows a self intersecting tessellation of a boundary curve.

4. The marked cells form a staircase like polygonal chain[1] as shown in Fig. 8. The trimming curve forms another polygonal chain offset from the staircase. Partition this region across cell edges.

5. Triangulate each resulting simple polygon. (We use Siedel's algorithm [Sei91].)

   To prevent cracks in the display, no additional points on the curve are introduced in the region partitioning step or the triangulation step.

4.1 Tessellation of Surface

Since we are targeting engineering applications, we assume that models are regularized solids. There are no self intersecting patches. Still, some patches can be highly curved and polygonizing such patches can cause an incorrect topology. This is demonstrated by an example in Fig. 9. Such behavior occurs when a tessellant for some part of the patch intersects some other part of the patch i.e. a part of the patch is too close to another. (This distance is not measured along the patch but normal to it). Since this case rarely occurs in practice, we adopt a simple solution. For such patches we ensure that the tessellation step is smaller than the smallest such offending distance of the patch. Another solution that works well in practice is to subdivide such patches that have offending distance smaller than a user specified tolerance.

4.2 Tessellation of Trimming Curves

The trimming curves corresponding to surface intersections are represented as piecewise algebraic space curves, as described in Section 2.2. The algorithm computes projected area of the bounding box $\mathbf{B}_i$ for each trimming curve on the screen space. Based on the area it computes $n_t(TOL)$, the number of points needed for the piecewise representation of the trimming curve. The algorithm computes these points using the tracing algorithm presented in Section 2.2. The step size Δ for the tracing algorithm is set equal to $L/n_t(TOL)$, where L is the length of the main diagonal of $\mathbf{B}_i$.

[1]This chain is null if a trimming curve is contained in a cell.

Analogously to patches, trimming curves with high curvature can cause the violation of the regularity constraint on the topology – the piecewise tessellation of the curves can be non-simple. Unfortunately, this happens rather often in practice. As a result, increasing the tessellation of such curves causes an explosion of tessellants, and hence a more efficient solution is warranted.

For each trimming curve the algorithm finds a *non-simple skeleton*. This is a gross piecewise representation of the curve such that the part of the curve between any two points on the skeleton is *split-monotonic* with respect to the segment joining those points.
def: A curve c is split-monotonic with respect to a line l if the following property holds for any line l' perpendicular to l:

> If c intersects l' at points p_1 and p_2 such that there does not lie any other intersection between c and l' between p_1 and p_2 along l', then the points p_1 and p_2 split the curve in two parts none of which intersects l'.

We find the skeleton by starting with the polygon connecting all the points of inflection and do a naive[2] diagonal based subdivision to eliminate non-essential points from this polygon. If we pick the points on the skeleton, no non-simple polygons are generated. Notice this is not much unlike Rockwood's [RHD89] monotone decomposition of trimming curves, except we avoid the overhead of actually doing the decomposition and dividing the patches.

Crack Prevention: A trimming curve of a patch is a sequence of curves of intersection with neighboring patches. Each of these component curves are shared by two patches. To avoid cracks we need to make sure that the curve gets tessellated into the same number of segments for each of these patches. Since we are looking for a parallel tessellation algorithm with no inter-process communication, and we would like the freedom of assigning adjacent patches to different processors, we need to tessellate each patch independently.

Each preimage of the intersection curve, the trimming curves in the domain of two patches, uses the same bounding box in 3-space, $\mathbf{B}_i$ in its data structure. Therefore, the same values of $n_t(TOL)$, L and Δ are used for the trimming curves in different domains. The tracing algorithm evaluates points on the intersection curve as a function of Δ and uses their preimages to obtain points in the domain ((s,t) or (u,v) coordinates). Therefore, the images of piecewise linear representation of each trimming curve match identically in 3-space and the resulting image has no cracks. This evaluation of points is an expensive operation, but due to coherence only a few points need to be computed for any frame. We also use a small hysteresis factor to do over-tessellation when changing tessellation. This allows us to delay increasing or decreasing the tessellation when the tessellation bounds change.

The standard patch boundaries are not treated as curves of intersection. We do not need to evaluate bounds in 3-space for these curves.

4.3 Load Balancing

Our tessellation algorithm takes special care to eliminate any communication between processors. Currently, most graphics systems do not offer sufficient communication bandwidth

[2]This is an off-line process

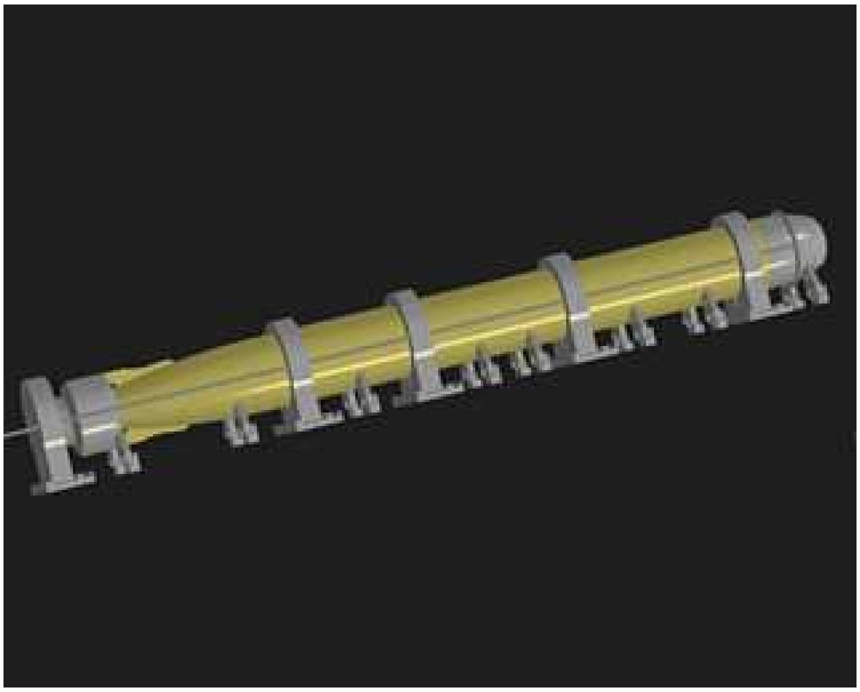

Figure 10: Part of Submarine Storage and Handling System: Shipping Model

between processors to sustain large volumes of information transfer. Most available bandwidth is typically used up by the rasterizers. Thus we have to be especially careful to distribute patches between processors so as to reduce load-imbalance. The problem is that in any visualization application, the tendency to zoom in on parts of the model is high, causing the tessellation and triangulation time of such patches to increase. Hence there is a need for patches in the zoomed-in region to be distributed across processors. This is handled by doing a round-robin distribution of primitives based on the adjacency information, so that adjacent patches do not lie on the same processor. But such distribution reduces the effectiveness of our visibility algorithm based on view frustrum culling. We group patches into clusters and cull all patches of a cluster if the bounding box of the cluster is off-screen. If adjacent patches are grouped in a cluster, the bounding box fits them tightly and is more effective. We do not yet have a good way of finding the optimal balance in this tradeoff.

4.4 Incremental Computations

In an interactive application there is typically a small change in the viewing parameters between successive frames. As a result, the bounds for tessellation of a surface and its trimming curves do not change much. The algorithm exploits this property by making use of the triangulation at the previous frame and computing an incremental number of extra polygons for the current frame. The incremental algorithm for decomposing the domain into cells divides some of them into rectangular subcells or merges adjacent cells into supercells [KML95]. These subcells or supercells are used as normal cells for successive frames. For trimmed surfaces this results in small changes in cell intersections and triangulation as opposed to

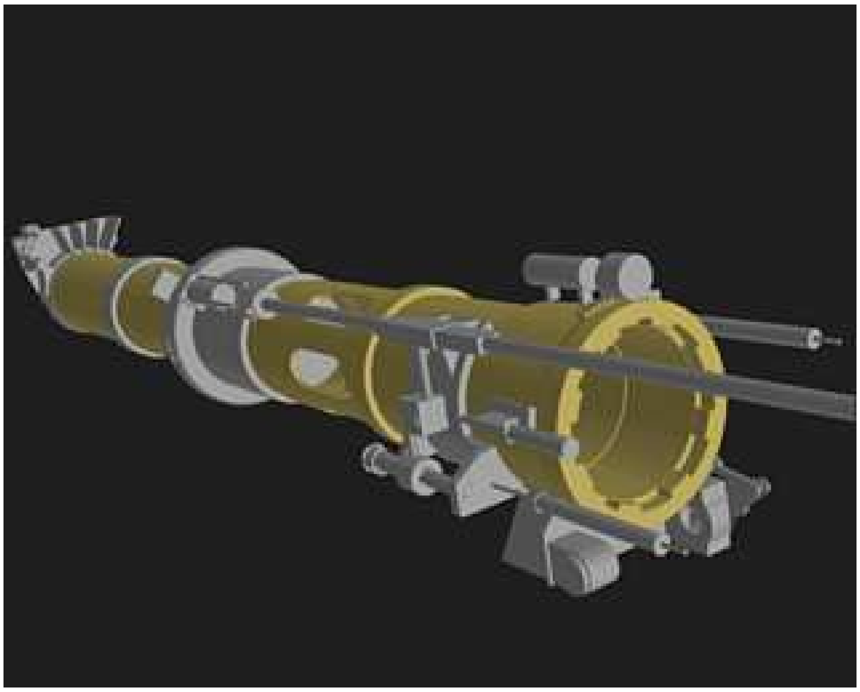

Figure 11: Part of Submarine Storage and Handling System: Tube Model

retracing all the trimming curves and retriangulating each region.

As the value of $n_t(TOL)$ for a trimming curve increases or decreases between frames, the algorithm computes additional points or takes a subset of the original set of points. The algorithm makes use of the piecewise linear approximation of the trimming curve and computes additional points corresponding to the line segments of maximum length. It uses the start point of the maximal line segment as (s_0, t_0, u_0, v_0) and new value of $\Delta = L/n_t(TOL)$ for the tracing algorithm. For deletion the algorithm removes points corresponding to line segments of minimal length. It keeps track of the maximal and minimal lengths for subsequent frames.

Depending on the change in cell decomposition and piecewise approximation of the trimming curve, the algorithm only retraces the portion of the trimming curves belonging to the subcells or supercells and computes intersection with their boundaries. The new set of crossings is used to either decompose the original triangulation of a region or merge the triangulations of two or more adjacent regions. In the former case, this boils down to decomposing some triangles into trapezoids and triangles and in the latter case some of the triangles are merged into new trapezoids or triangles. The new trapezoids are triangulated and thereby obtaining a new triangulation of the trimmed surfaces. Overall, the algorithm performs only a small number of evaluations for any frame. For a detailed analysis of the cases see [KM95b].

5 Implementation and Performance

The algorithms presented above have been implemented and applied to a number of solids comprising the model of a submarine storage and handling system, made available to us by Electric Boat. The model consists of about 2, 000 solids. Many of the primitives are composed of polyhedra and conicoids like spheres, cylinders. Additional primitives include prisms and surfaces of revolution of degrees six and more. A few of the primitives are composed of Bézier surfaces of degree as high as twelve. Most of the CSG trees have heights ranging between six and twelve and some of them are as high as 30. The B-reps of many of the solids consist of more than $40-45$ trimmed Bézier surfaces and some of them have up to 148 surfaces.

5.1 Model Generation

The running time of the system varies on different solids. In particular, it depends on the number of Boolean operations, number of intersecting pairs of surfaces and the number of connected components generated. In most cases, it spends about half the time in computing intersections between pairs of surfaces and the other half in ray-shooting and computing the components of new solids. Our current implementation has not been optimized. It performance on a small subset of solids from the model has been highlighted in Table 1. The solids are shown in Fig. 6.

A major problem in the application of our system to different models is in terms of numerical accuracy of computations and its impact on the robustness of the entire system. The problem of building robust solid modeling systems based on floating-point computation is fairly open and no good solutions are known [Hof89]. In our case, the algorithm uses ϵ tolerances at different parts of the overall algorithm. Depending on the values of the tolerances, the robustness of the algorithm can vary considerably.

As an input, the user specifies a set of four tolerance values and they are used as part of the surface intersection algorithm, for ray-shooting, merging intersection curves and detecting planar overlaps. The surface intersection algorithm normalizes the input surface parameterizations and ensures that the output of the intersection algorithm has certain digits of accuracy. The intersection algorithm is based on iterative numerical algorithms and we set the termination criterion accordingly. Similar criteria are used in computing the intersection of trimming curves represented as piecewise algebraic curves. At the end of every CSG operation, the system makes sure that the topology of the resulting solid is consistent.

Model	# of prim.	# of CSG opns.	Running time(in min.)	# of patches (in B-Rep
Fig. 6(a)	23	20	2.6	137
Fig. 6(b)	6	5	0.8	89
Fig. 6(c)	6	5	0.7	116
Fig. 6(d)	28	27	3.4	169
Fig. 6(e)	11	10	1.1	69
Fig. 6(f)	22	21	3.1	146

Table 1: Performance of the Solid Modeling System

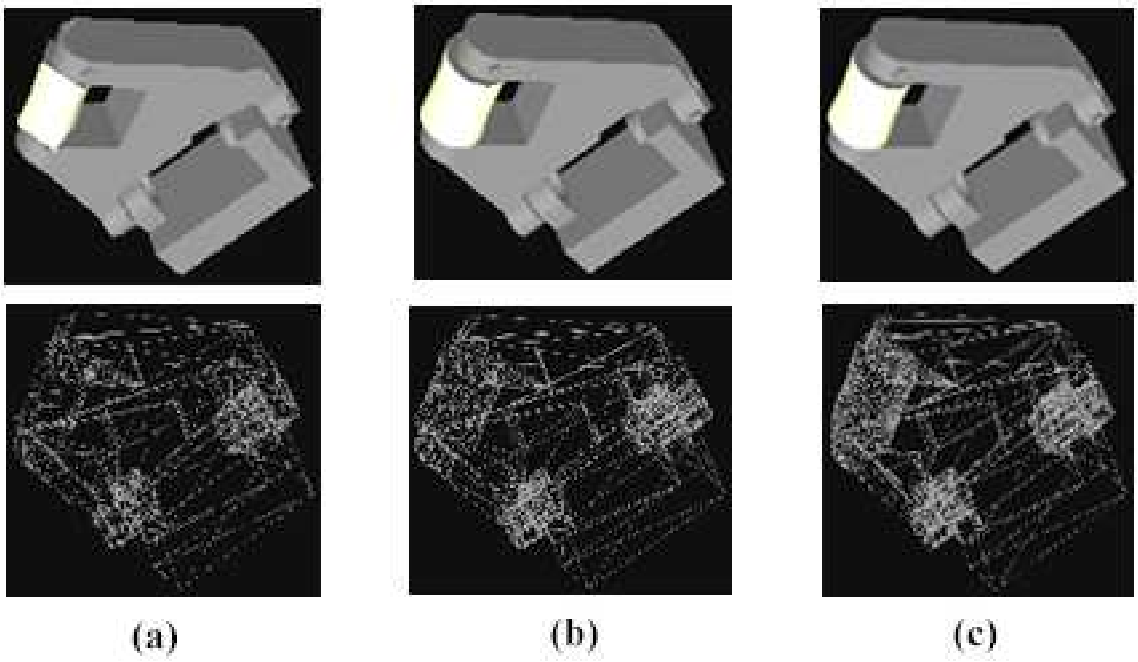

Figure 12: Different Level of Details of the Roller Model in Fig. 6(a)

5.2 Display

The display system has been implemented on an SGI Onyx and Pixel-Planes 5 graphics system. On Pixel-planes 5 it uses multiple graphics processors (GP's) for visibility computations, evaluating Bézier functions and triangulating polygons. The trimmed Bézier surfaces are evenly distributed over different GP's and the system associates each surface with the parent solid for visibility computations. Each GP has at most $2.5MB$ of memory for storing the surface representations and caching the triangle vertices and their normals. The input data for 500 surfaces on a GP takes more than half of the memory and the system uses a dynamic memory allocation scheme for caching triangles.

The rendering algorithm is *guaranteed* to produce topologically correct triangulations. For ill-parametrized patches, partial derivatives may vanish at some points on the patches. Using the non-degenerate normal at a neighboring location works well in practice. In Fig. 12 we have shown different levels of detail for the roller model (Fig. 6(a)) as a function of the viewing parameters. As we zoom in or out on a model, it produces different levels of detail incrementally and no visual artifacts appear. The trimming algorithm also works for trimming curves represented as piecewise linear or spline curves. Compared to Rockwood et. al. [RHD89] algorithm, our trimming algorithm is faster by a factor of *seven or eight* on models consisting of about 200 surfaces on an SGI Onyx. It also produces fewer triangles. We used SGI's GL library implementation of [RHD89] algorithm for comparison.

We compared the performance of our system on parts of a submarine storage and handling system. The model was designed using Catia CAD system. This version of Catia does

Model Name	Figure No.	Catia's B-rep Polygons	Our B-rep Trim. Bézier
Pivot	Fig. 1	73,269	4,435
Shipping	Fig. 10	52,107	3,346
Tube	Fig. 11	31,846	1,452

Table 2: Comparison of our Solid Modeling System with Polygonal representation (from Catia)

not support Boolean operations on curved geometries [3] and, therefore generates a dense polygonal B-rep for each model. Different models, their polygonal representation (from Catia) and trimmed Bézier representation (from our solid modeling system) are listed in Table 2. Pixel-Planes 5 graphics system can barely render $3-10$ frames on the polygonal representation on a combination of these models ($389,721$ polygons). Furthermore, the image quality is poor when we zoom onto a model. On the other hand our display system can render the multiresolution representation ($20,928$ trimmed Bézier surfaces) at $12-25$ frames at good resolutions.

6 Conclusion

We have presented a system for generating an accurate and multiresolution B-rep for CSG models composed of curved primitives and rendering the resulting primitives on current graphics systems. Its application to parts of a submarine storage and handling system model helped us improve the overall frame rate by two or three times, as compared to directly rendering the polygonal B-rep. The main benefits in image quality, memory requirements and rendering performance come from the multiresolution representation of the B-reps. However, our display system needs more processing (CPU) power for incremental triangulation computation. We are currently working on efficient memory management, load balancing and performance enhancement of the display system to achieve interactive visualization of the entire submarine storage and handling system model. We are also working on more efficient algorithms to perform split-monotic subdivision.

7 Acknowledgements

We are grateful to Fred Brooks, Anselmo Lastra and members of UNC Walkthrough group for productive discussions. The CSG model of the submarine storage and handling system was provided to us by Greg Angelini, Jim Boudreaux and Ken Fast at the Electric Boat division of General Dynamics.

[3] The latest version of Catia does support curved geometries.

References

[AES91] S.S. Abi-Ezzi and L.A. Shirman. Tessellation of curved surfaces under highly varying transformations. *Proceedings of Eurographics'91*, pages 385–97, 1991.

[BR94] C.L. Bajaj and A. Royappa. Triangulation and display of rational parametric surfaces. In *Proceedings of Visualization'94*, pages 69–76, IEEE Computer Society, Los Alamitos, CA, 1994.

[Coh83] E. Cohen. Some mathematical tools for a modeler's workbench. *IEEE Computer Graphics and Applications*, 3(7):63–66, 1983.

[DLW93] T. Derose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topology type. Technical Report TR 93-10-05, Department of Computer Science, University of Washington, 1993.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1993.

[Fe89] H. Fuchs and J. Poulton et. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Proc. of ACM Siggraph*, pages 79–88, 1989.

[Fun93] T. A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architecture Models.* PhD thesis, CS Division, UC Berkeley, 1993.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proc. of ACM Siggraph*, pages 231–238, 1993.

[GMTF89] J. Goldfeather, S. Molnar, G. Turk, and H. Fuchs. Near real-time csg rendering using tree normalization and geometric pruning. *IEEE Computer Graphics and Applications*, 9(3):20–28, 1989.

[HDD+93] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle. Mesh optimization. In *Proc. of ACM Siggraph*, pages 19–26, 1993.

[HG94] P. Heckbert and M. Garland. Multiresolution modeling for fast rendering. *Proceedings of Graphics Interface*, 1994.

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

[Hoh91] 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.

[KE89] G. Kedem and J.L. Ellis. The ray-casting machine. In *Parallel Processing for Computer Vision and Display*, pages 378–401, Springer-Verlag, 1989.

[Kea94] M. Kelley and K. Gould et. al. Hardware accelerated rendering of csg and transparency. In *Proc. of ACM Siggraph*, pages 177–184, 1994.

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

[KM95a] S. Krishnan and D. Manocha. Numeric-symbolic algorithms for evaluating one-dimensional algebraic sets. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, 1995.

[KM95b] S. Kumar and D. Manocha. Efficient rendering of trimmed nurbs surfaces. *Computer-Aided Design*, 1995. To appear.

[KML95] S. Kumar, D. Manocha, and A. Lastra. Interactive display of large scale nurbs models. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 51–58, 1995.

[KNM95] S. Krishnan, A. Narkhede, and D. Manocha. Boole: A system to compute boolean combinations of sculptured solids. Technical Report TR95-008, Department of Computer Science, University of North Carolina, 1995.

[LC93] W.L. Luken and Fuhua Cheng. Rendering trimmed nurb surfaces. Computer science research report 18669(81711), IBM Research Division, 1993.

[MD94] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves i: simple intersections. *ACM Transactions on Graphics*, 13(1):73–100, 1994.

[RB92] J.R. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. Technical Report RC 17697, IBM T.J. Watson Research Center, 1992.

[RHD89] A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. In *Proceedings of ACM Siggraph*, pages 107–17, 1989.

[Rot82] S. Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18:109–67, 1982.

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

[RW90] J. Rossignac and J. Wu. Correct shading of regularized csg solids using a depth-interval buffer. In *Eurographics Workshop on Graphics Hardware*, 1990.

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

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[Sei91] 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.

[SN91] T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.

[SZL92] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. In *Proc. of ACM Siggraph*, pages 65–70, 1992.

[Tel92] S. J. Teller. *Visibility Computations in Densely Occluded Polyheral Environments.* PhD thesis, CS Division, UC Berkeley, 1992.

[Til83] W. Tiller. Rational b–splines for curve and surface representation. *IEEE Computer Graphics and Applications*, 3(6):61–69, 1983.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In *Proc. of ACM Siggraph*, pages 55–64, 1992.