SOLID AND PHYSICAL MODELING

INTRODUCTION

Overview

A solid model (or simply a *solid*) is a digital representation of the shape of an existing or envisioned rigid object. Solids are routinely used in many applications, including architecture, entertainment, and health care. In particular, they play a major role in the manufacturing industries, where precise models of parts and assemblies are created using a *solid modeler*—a computer program that provides facilities for designing, storing, manipulating, and visualizing assemblies of solids. These solids can be designed by a human through a graphic user interface (GUI), constructed from scanned three-dimensional (3-D) data, or created automatically by applications via an application programming interface (API). The design process is usually incremental and follows one—or a combination—of the three main paradigms: *surface stitching*, where designers specify surfaces and stitch them together to define the boundary of a solid; *constructive solid geometry* (abbreviated CSG), where designers instantiate solid *primitives* (blocks, cylinders, extruded areas. . .), specify their dimensions, position, and orientation, and combine them using Boolean operators; and *feature-based* (1), where solids are specified by a sequence of operations that create or modify *features*—entities meaningful for a particular application. By exposing the *parameters* (2) that define the positions and dimensions of the features or of the CSG primitives, one creates a *generic solid* that can be instantiated (reused) with different parameter values by the designer to simplify engineering changes or to make new models or by a program to assist shape optimization or tolerance analysis.

Contemporary 3-D graphics capabilities and intuitive GUI support the design activities by providing designers with an interactive environment in which they can; (a) see, after each design step, realistic images of the solids they have designed; (b) control the view parameters and the rendering attributes; (c) graphically select a solids, a primitive, or a feature; (d) edit a selected element by changing its dimensions, position, or orientation; and (e) create or modify solids or features through a variety of operations, which include Booleans, extrusions, sweeps, offsets, fillets, bends, warps, morphs, and various filters.

Applications

When used in the computer-aided design/computer-aided manufacturing (CAD/CAM) industry, solid modeling impacts a variety of design and manufacturing activities. Examples include early sketches, space allocation, detailed design, drafting, visualization, maintenance simulation, usability studies, engineering changes, analysis of tolerances (3), 3-D mark-up, product data management, remote collaboration, Internet-based catalogs, analysis of mechanisms (4), meshing (5), and finite elements analysis, process planning and cutter-path generation for machining (6, 7), assembly and inspection planning, product documentation, and marketing. For example, CAD/CAM applications evaluate various properties of a solid or assembly, such as volumes or contact areas, or assess the feasibility and cost of life-cycle activities, such as manufacture, assembly, or inspection. Ideally, these applications should run concurrently with the design process, to help assess the consequences of design decisions. Analysis algorithms are available for generating displays of solids in many styles and degrees of realism, for kinematic simulation, for the evaluation of mass properties (8), for interference detection (9) in static environments, and so on. On the other hand, problems such as design and planning, which involve synthesis, are much less understood, although progress was made in certain areas, such as feature recognition for machining planning (10), dimensional inspection planning (11, 12), and robot path planning (13). Most of these application algorithms make extensive use of the fundamental queries and constructions described below. For example, mass property calculation for CSG solids typically involves either line/solid classification or CSG-to-octree conversion (14). And feature recognition and accessibility analysis for inspection planning require Boolean operation capabilities. Some applications, such as finite element method (FEM) analysis or computational fluid dynamic (CFD) simulations, require converting the solid into a Cartesian or unstructured mesh of tetrahedra or hexahedra. The automation of the construction of optimal meshes continues to challenge the mesh generation community. Furthermore, the mesh may evolve during simulation (15).

Solids are also used in *medical visualization* and *surgery planning* to represent the desired parts (bone, cartilage, heart, brain, artery) of a human anatomy and to simulate the effect of a contemplated surgical intervention (graft of cartilage, cardiopulmonary connection (16), radiation). The extraction of solids from acquired computed tomography (CT) or magnetic resonance image (MRI) scans is complicated by the errors and low sampling rate of the acquired data set (17). Solids are used in *architecture* and *construction* to model the components, to verify that they fit properly and adhere to construction codes, and to compute their mass and possible deflection under stress (18) or vibration modes. Solids may also be used to represent the empty space and analyze it in terms of aesthetics or acoustics. Finally, solids are used in the *entertainment* and *training* industry represent to the objects in a virtual scene and to detect collisions between moving objects.

Evolution

Research in solid modeling emerged in the 1970s from early exploratory efforts that sought shape representations suitable for machine vision and for the automation of seemingly routine tasks performed by designers and engineers in computer-aided design, manufacturing, construction, and architecture (encapsulated in the CAD/CAM/CAE abbreviation) (19). In particular, early applications of solid modeling focused on producing correct engineering drawings automatically and on cutter-path generation for numerically controlled machining (20). Such two-dimensional (2-D) drawings are now in digital form and are being rapidly replaced with 3-D solid models,

augmented with dimension, tolerance, and other mark-up information.

Solid modeling has evolved to provide a set of fundamental tools for representing a large class of products and processes and for performing on them the geometric calculations required by applications. The ultimate goal was to relieve engineers from the low-level or noncreative tasks in designing products (21); in assessing their manufacturability, assemblability, and other life-cycle characteristics; and in generating all the information necessary to produce them. Engineers/designers should be able to focus on conceptual, high-level design decisions, whereas domain-expert programs should provide advice on the consequences and optimization of design decisions and generate plans for the manufacture and other activities associated with the product's life cycle. The total automation of analysis and manufacturing activities (22), although in principle made possible by solid modeling, remains a research challenge despite of much progress on several fronts.

Solid modeling has rapidly evolved into a large body of knowledge, created by an explosion of research and publications (23–26). The solid modeling technology is implemented in dozens of commercial solid modeling software systems, which serve a multi-billion-dollar market and have significantly increased design productivity, improved product quality, and reduced manufacturing and maintenance costs. Today, solid modeling is an interdisciplinary field that involves a growing number of areas. Its objectives evolved from a deep understanding of the practices and requirements of the targeted application domains. Its formulation and rigor are based on mathematical foundations derived from general and algebraic topology and from Euclidean, differential, and algebraic geometry. The computational aspects of solid modeling deal with compact representations and efficient algorithms, and benefit from recent developments in the field of *Computational Geometry*. Compact representations and efficient processing are essential, because the complexity of industrial models is growing faster than the performance of commercial workstations and the available transmission bandwidth (27). Techniques for modeling and analyzing surfaces and for computing their intersections (28, 29) are fundamental in solid modeling. This area of research, sometimes called *computer-aided geometric design* (CAGD), has strong ties with numerical analysis and differential geometry. GUI techniques also play a crucial role in solid modeling, because they determine the overall usability of the modeler and impact the user's productivity. There have always been strong symbiotic interaction and overlap between the solid modeling community and the computer graphics community. In particular, the complexity of the solid assemblies used in industrial applications has challenged the capabilities of graphics subsystems, motivating research in shape simplification (30, 31), occlusion culling (32), and geometric compression (33). A similar symbiotic relation with computer vision has regained popularity, as some research efforts in vision were model-based (34) and attempted to extract 3-D models from images or video sequences of existing parts or scenes. These efforts are particularly important for solid modeling, because the cost of manually designing solid models of existing objects or scenes far exceeds the other costs (hardware, software, maintenance, and training) associated with solid modeling. Finally, the growing complexity of solid models and the growing need for collaboration, reusability of design, and interoperability of software require expertise in distributed databases, constraint management systems, optimization techniques, object linking standards, and Internet protocols.

Although the low-level geometry processing tools upon which the solid modeling technology still lack in performance, reliability, and generality, much of the innovations in solid modeling are focused on two primary usability issues: (a) Simplify the creation of solid models in a collaborative and often distributed design environment and (b) integrate solid modeling functionality into broader *product data management* (PDM) processes.

Beyond Solid Modeling

The fundamental difference between solid modeling and other geometric modeling paradigms lies in the fact that a solid model is a complete and unambiguous representation of a solid (35) and hence makes it possible to develop practical algorithms for *set membership classification*: i.e., the ability to divide a candidate set C into parts that are in the interior of the solid S, in its exterior, or on its boundary (36). For example, integral properties of a solid may be approximated by classifying randomly sampled points and by adding the properties of points that fall in the interior. Realistic images produced through ray tracing require classifying rays and identifying points where a ray intersects the boundary of the solid. The ability to design and process mathematically precise representations that distinguish among the interior, the boundary, and the exterior of a rigid object being modeled is important to manufacturing or construction applications. Yet it does not address a plethora of characteristics of real objects that are essential for other applications, such as medicine or entertainment, or for manufacturing composite or flexible objects and analyzing their nonuniform properties. To address these challenges, several areas of research extend the scope of solid modeling.

Nonregularized, sometimes called *nonmanifold* (NM) modeling techniques have been proposed (37) to support the representation of structures that combine different materials in a solids (38) and may have lower dimensional entities, such as membranes, sheets, and wires (39). Among other applications, such nonregularized representations provide support for modeling contacts among objects, fractures in objects, anisotropic or composite materials, anatomies of human organs, or the partition of a solid into components made of different materials. Higher dimensional extensions of such nonregularized modelers

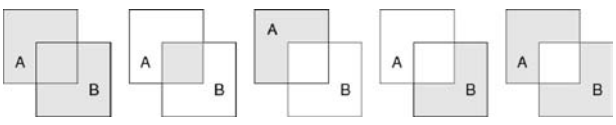

Figure 1. Union A+B, intersection AB, differences A–B and B–A, and symmetric difference (A–B)+(B–A).

support analysis of configuration spaces for robotics and tolerances for manufacturing.

Solid modeling applications that deal with natural or artistic shapes are more in need of easy-to-use shape acquisition, editing, and segmentation tools than of mathematical precision. They often use sampled *free-form* models, which are easy to create and deform (16, 40). Nevertheless, they usually rely on algorithms that assume the represented shapes to be topologically valid *solids*. Many applications represent these solids in terms of a control triangle mesh, which may be generated automatically to interpolate surface samples acquired through a scanning process, created by tessellating implicit surface models (41, 42), edited through user-controlled free-form deformations (43), extracted as a level set of a three-dimensional scalar field produced through simulation, or refined through subdivision. To compare such models, compress them, segment them automatically into features of interest to the application domain, visualize them more clearly, or mesh them for analysis or animation, a global parameterization of these models is desired that minimizes distortion and is aligned with the local curvature lines or features.

Finally, solids and structures may move, deform, break (44) or change state over time (45). These *animations* may either be designed by humans to demonstrate a process, acquired through sensors from real behaviors, or simulated using rules that may, but need not be, based on physical principles. For example, one may use these technologies to model the contractions of a ventricle, the flexing of an aircraft wing, the melting of an iceberg, the sagging of a sofa, or the breaking of a windshield. Many of these animations use a tetrahedron or hexahedron mesh representation of the solids.

To reflect on the importance of these three aspects of solid modeling (nonregularized structures, free-form shapes, and animations), the solid modeling community has extended the name of its major conference from *solid modeling* to *solid and physical modeling* (46).

Organization

Next, we introduce the topological foundations upon which the solid modeling technology is based. Then, we discuss the geometric domain, representation schemes, and numeric accuracy issues. We will explain the fundamental technologies for solid modeling with triangle meshes. We will analyze their extensions to curved boundary representations. Then we will focus on constructive representations and review parameterization techniques. Finally we will discuss morphological and other operations and investigate *human–shape interaction* (HSI) issues.

TOPOLOGICAL DOMAIN

Set Theoretic Boolean Operations

The solid modeling technology manipulates point sets. Set theoretic operators provide the essential tool for defining complex solids or expressing their properties. We will use the following operators. The *complement* of a set S will be denoted !S. Expression A+B, AB, and A–B, respectively, will denote the *union*, *intersection*, and *difference* between set A and set B. There is a correspondence between these Boolean set theoretic operators (*complement*, *union*, *intersection*, and *difference*) and their logical counterparts (*not*, *or*, *and*, and *and not*) because we can use the set builder notation to define sets in terms of logical expressions of predicates (*membership classification*) of the points they contain. Specifically, !S is the set of points that are *not* in S; A+B is the set of all points that belong to A *or* to B; AB is the set of all points that belong to A *and* to B; and A–B is the set of all points that belong to A *and not* to B. The predicate of a point p being in or not in set S is its *membership classification* with respect to S.

Because of this correspondence, set operators inherit the properties of their logical counterparts. For example, !!S=S, !(A+B)=!A!B, and !(AB)=!A+!B, and two operators suffice to define the others. For example, we can express the difference and union in terms of complement and intersection: A–B=A!B and A+B=!(!A!B). Nevertheless, the Boolean operations popular in solid modeling are the union, intersection, and difference. Note that, among the 16 different Boolean combinations of two sets, 8 are unbounded, 3 are trivial, and only 5 (shown in Fig. 1) are of interest for defining bounded solids.

Regularization and Membership Classification

The digital representation and basic algorithms used in solid modeling are based on a few fundamental principles discussed in this section. In the context of solid modeling, the term "solid" distinguishes a subclass of 3-D sets from more general sets, which include arbitrary Boolean combinations of volumes, curves, and surfaces. In particular, a solid is assumed to be *bounded* (i.e., it fits inside a ball of finite radius) and *regular* (i.e., it is the *closure* of its *interior*). The former assumption avoids the need to compute and represent intersection points or curves that are infinitely far from the origin. The latter, proposed in the mid-1970s at the University of Rochester's Production Automation Project (PAP) as a mathematical definition of physical solids (35), has guided the development of correct algorithms for regularized Boolean operations that are guaranteed to produce models of solids that can be manufactured, as opposed to volumes with lower dimensional dangling surfaces, missing boundaries, and cracks.

To better understand what a regular set is and how this restriction helps in solid modeling, consider the following decomposition of space. (Formal definitions for the topological concepts used here may be found in Reference 47. We offer intuitive—although less precise—formulations and combine them to define new terms that identify the various components of the boundary of a set.) We say that a point p is *adjacent* to a set S if all balls of center p and strictly positive radius contain at least one point of S. A set S decomposes space into three parts: a) the *boundary* bS of S, which is the closed, lower dimensional set of points that are adjacent to S and to its complement !S; b) its *interior* iS, which is the open set of point of S that are not in bS; and c) its *exterior* eS, which is open and comprises the remaining points. The boundary bS may be further decomposed (Fig. 2) into four parts: a1) the *skin* sS, which is the set of points

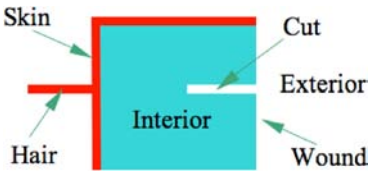

Figure 2. A 2-D set S may be decomposed into its interior iS, its skin sS, and its hair hS. !S may be decomposed into its exterior eS, its wound wS, and its cut cS.

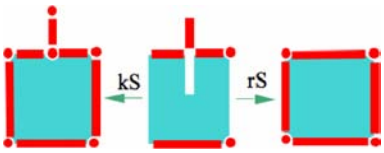

Figure 3. The closure kS (left) of a set S (center) is obtained by adding the cut cS and the wound wS. The regularization rS (right) is obtained by removing (cutting) the hair hS and by adding (filling) the wound wS and the crack cS.

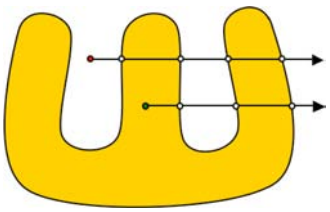

Figure 4. The parity of the number of crossings of bS by a ray from a candidate point establishes its membership.

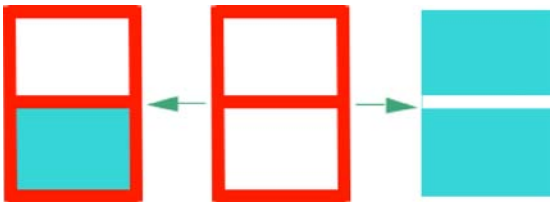

Figure 5. A one-dimensional set (center) may be the boundary of more than one nonregular set: three are shown here.

in S adjacent to both iS and eS; a2) the *wound* wS, which is the set of points in !S adjacent to both iS and eS; a3) the *hair* hS, which is the set of points in S not adjacent to iS; and a4) the *cut* cS, which is the set of points in !S not adjacent to eS. The *closure* kS of S is the union of S with its boundary. The *regularization* rS of S is the closure k(iS) of its interior iS (Fig. 3). We say that S is *regularized* when S=rS.

The *membership classification* (or simply membership) of a point p with respect to a set S may take three values: IN indicates that p is in iS, OUT indicates that p is eS, and ON indicates that p is bS. Regularization, which takes S and returns rS, changes the membership of wS to ON, of hS to OUT, and of cS to IN. Converting a set S to its regularized version rS has several benefits for modeling: 1) the dangling elements (hS and cS) need no longer be represented, because hS is not distinguishable from eS and cS from iS; 2) the boundary bS needs no longer be split into wS and sS; and 3) most importantly, when bS is bounded, S may be represented without ambiguity by its boundary bS. The unambiguous specification (and representation) of a solid by its boundary assumes that S, and hence, iS is bounded. A regularized set S is the union of iS with bS.

One can classify a point p (i.e., establish its membership) with respect to a solid S represented by its boundary as follows (36). If a point p lies on bS, then it is by definition ON S. Otherwise, we must establish its membership with respect to iS. Point p is in iS if a path C that starts at p and goes to infinity crosses bS an *odd* number of times. To simplify the definition—and the corresponding algorithms—we exclude paths that osculate (touch without crossing) bS. With these precautions, the membership of a point p that is not in bS may be consistently established using any such path. The justification (Fig. 4) is that a point infinitely far along the path is OUT, because S is bounded. As that point travels toward p, it toggles its membership between IN and OUT each time it crosses bS. For example, in computer graphics, a ray (straight path) starting at p is used. Note that in the definition of S in terms of bS and in the membership algorithm, no assumption is made as to the *orientation* of bS, nor as to the number of shells of bS or of connected components of S. However, it is implicitly assumed that the boundary representation makes it possible to test whether a point lies on it and to correctly compute the parity of the number of intersections with a ray.

As mentioned, some applications manipulate digital models of physical objects that are not solids but more general structures that divide space into dimensionally homogeneous, connected cells (vertices, edges, faces, and volumes) and group such cells into collections that usually represent *nonregularized* sets. Note that, in general, the boundary bS of such a nonregularized set may not be used to represent S, nor to perform membership classification, because it may be ambiguous (i.e., it may be the boundary of more than one set) (see Fig. 5 for an example). Hence, modeling systems that manipulate such nonregular features must construct, store, and exploit explicit information about the full-dimensional (open) cells and their bounding lower dimensional shells. We will elaborate on such representations later in this article.

The above discussion is important for three reasons: 1) It provides the mathematical foundation for the numerous solid modeling schemes that use a *boundary representation* (BRep); 2) it provides a practical algorithm for membership classification; and 3) it defines which faces should be retained (sS and wS) and which should be discarded (hS and cS) when computing (*boundary evaluation* process) a representation of the boundary bS of a solid S defined by a Boolean, sweep, morph, or other operation. In particular, when a Boolean operation (union, intersection, difference) is followed by a regularization operation, we say that it is a *regularized Boolean*. Note that regularization may be applied after each Boolean (this is done during *incremental boundary evaluation*) or performed on the result produced by combining several solids according to a Boolean expression, which is a constructive solid geometry (CSG) representation of the solid. Both BRep and CSG representations are discussed below. Let us now examine how regularization affects membership classifications. Assume that the

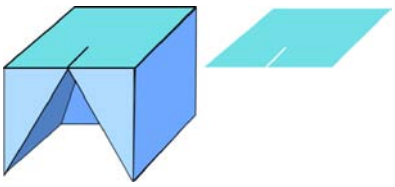

Figure 6. The solid (left) has a nonmanifold edge, which creates a cut in the top face (right).

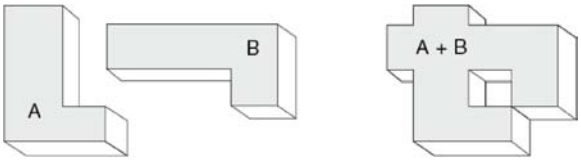

Figure 7. The regularized union A+B of two L-shaped manifold solids (left) creates a nonmanifold solid (right) when the shapes are carefully positioned.

memberships of a point p with respect to solids A and B are known. If at least one of them is not ON, then we can conclude the membership classification of p with respect AB, A+B, and A–B without further information. For example, if p is ON A and IN B, it is ON AB, IN A+B, and OUT of A–B. When p lies ON A and ON B, one must consider the local neighborhood (infinitely small ball) around p, compute its intersection with A and with B, and then test whether the desired Boolean operation on these neighborhoods produces an empty, full, or mixed neighborhood (48).

Boundary Evaluation

Assume that we know the boundary of two solids A and B. How should we compute the boundary of their Boolean combination S (for example, A+B, AB, A–B)? The solid modeling algorithms that perform this computation are all based on the fundamental principle that the boundary of a Boolean combination is a subset of the union bA+bB. Hence, the faces of S are subsets of the faces of A and B. As discussed in the next section, these subsets are either expressed using *solid trimming* (for example, the subsets of bA in B) (49, 50) or *parametric trimming* (for example, the curves that trace the intersection between the boundaries of the two solids).

NonManifold Singularities

The boundary bS of a solid S is composed of a set of pairwise disjoint cells: faces (bounded, connected, and relatively open subsets of a surface), edges (which do not include their endpoints), and vertices. A *face* is a connected portion of the smooth subset of bS. It does not include its bounding edges and vertices and may have cuts (see Fig. 6). Hence, a face F equals its interior iF [relative to the surface (extent) in which it is imbedded] but may not equal the relative interior of its closure. An *edge* is a connected one-dimensional set of points of the nonsmooth subset of bS that are all adjacent to the same set of faces of S. The vertices of bS are the isolated points of bS that are not included in any face or edge.

An edge is *manifold* if it is adjacent to exactly two faces. S is said to be *edge-manifold* if all its edges are manifold. A vertex p of S is *manifold* if the intersection of bS with an infinitely small open ball of positive radius centered at p is homeomorphic to a disk. A solid is *manifold* if all its edges and vertices are.

To reduce the complexity of the underlying data structures, early solid modelers only supported manifold solids, forcing designers to carefully avoid the creation of nonmanifold edges or vertices. Unfortunately, the domain of manifold solids is not closed under many modeling operations. For example, the union A+B of the two manifold solids in Fig. 7 is nonmanifold. Even though one may argue that nonmanifold solids do not correspond to physical models, they occur often in the intermediate stages of design and may be desired as the means of conveying the designers' intent. Hence, many contemporary modelers support the creation and processing of such nonmanifold solids. Nonmanifold solids, which are regularized and hence unambiguously defined by their boundary, should not be confused with the nonregularized sets mentioned earlier (even though some authors use the term nonmanifold models (39) when referring to the latter).

GEOMETRIC DOMAIN AND REPRESENTATION SCHEMES

The choice of representations used by the modeler determines its domain (i.e., which objects can be modeled precisely) and has a strong impact on the complexity and performance of the algorithms that create or process the representations. In the previous section, we have discussed topological restrictions satisfied by solids and further restrictions limiting the domain of some modelers to manifold solids. In this section, we discuss restrictions imposed on the geometry of the bounding faces. These restrictions are linked to the choice of the representation scheme; hence, we discuss them together.

Geometric Domain

A modeler may support several distinct representation schemes and conversion algorithms between them. Various types of representation schemes have been explored for solid modeling. We review the most important 10 here.

Three schemes are *discrete*:

1) A *voxel model* represents a solid as a binary mask indicating which cells (voxels) of a regular three-dimensional lattice are IN the solid. To save storage, voxels with identical mask values may be aggregated hierarchically into an *octree* (51, 52).
2) A *ray model* (53) groups stacks of voxels into columns and for each column indicates which portion lies in the solid. These portions are estimated by computing the intersections of rays (column axis) with the solid.
3) A *slice model* (also called 2.5-D) represents a solid as a stack of slices, each defined by a planar contour and a depth. Note that these three discrete representations produce jaggy approximations of slanted or curved boundaries. A variety of algorithms have been pro-

posed for fitting a triangle mesh that is a better approximation of the original (smooth) solid (54–56).

Two schemes are based on *sampling*:

4) A *triangle mesh* stores a set of samples (points) on the boundary of a solid and a set of triangles that define a piecewise linear (polyhedral) surface that interpolates these samples (called the *vertices* of the mesh). The triangles (*connectivity*) are necessary to ensure that the solid is represented without ambiguity, even though the triangles may often be derived automatically (57, 58) and may be unnecessary for rendering when the sampling of the vertices is sufficiently dense (59). Adjacent coplanar triangles may be grouped into *polygonal faces*.
5) An *implicit surface* model defines the boundary of the solid as an isosurface (zero-crossing) of a scalar field or level set (60), which may be defined by combining individual functions (61) or by interpolating sample values measured or computed at the nodes of a regular lattice.

One scheme is *parametric*:

6) The boundary of the solid is a patchwork of *biparametric patches*, each defined by a mapping M(s,t) from the unit square $[0,1]^2$ to a connected portion of a surface. Typically the *mapping*, which specifies the (x,y,z) coordinates of point M(s,t), is a low-degree polynomial—or a rational polynomial—in s and t. The mappings must be arranged carefully to ensure that the patchwork forms a continuous (watertight) boundary and that it exhibits the desired degree of smoothness (normal, curvature) at the edges and vertices where patches meet.

Two schemes are *trimmed surface* models. They decompose the boundary of the solid into a set of faces. Each face is defined by an extent (host surface) that contains it and by a *trimming model*, which identifies the desired portion of the extent. The extent may be represented (as discussed above) as a parametric patch, by an implicit surface, or even by a triangle mesh. There are two approaches to the trimming model:

7) The *parametric trimming* model represents the trimming information as a set of bounding curves in parameter space (the unit square) of the patch (62). The principal difficulty of this approach lies in the fact that an edge where two patches connect has two representations (one per patch) and that these usually do not define exactly the same 3-D curve, producing a gap or overshoot in the boundary (63) and, hence, an invalid representation.
8) The *solid trimming* (50, 64) defines the face as the intersection of the extent with a solid or with the complement of a solid. This approach avoids the need for computing and representing the intersection curves and provides a powerful link between the trimmed face boundary representation and the CSG model discussed next.

Two schemes are *constructive*:

9) The CSG and the *binary space partitioning* (BSP) (65) representations identify which cells of an arrangement of half-spaces lie in the desired solid. The half-spaces are usually defined by implicit inequalities and may be unbounded. The surfaces that delimit them are usually simple and often restricted to natural quadrics (planes, cylinder, cone, and sphere) and tori (66), although in principle, any solid model could be used as a half-space or as its complement. CSG identifies the cells through a Boolean combination of these half-spaces, such as (A+B)–CD. BSP uses a recursive expression, where, at each step, a half-space is used to further split a portion of space in two parts that are each either declared as IN the solid, as OUT of the solid, or as further refined.
10) A *procedural model* generalizes the CSG and BSP models and represents the solid by a construction process (recipe) (67), which transforms or combines previously defined solids. A procedural model may measure dimensions of specific features and use these dimensions to define new features or patterns of features (68). It may create these features through Boolean operations or through direct boundary tweaking, which must be performed carefully to avoid creating invalid models (69). But it may also invoke other operations that for example *fillet* (70) the concave corners for die design, grow (offset) a solid's boundary (71) to create a crust for layered manufacturing (72), compute the volume swept by a solid cutter for simulation of NC machining (73), or warp the solid model of an artery for surgery planning (74). The challenge with procedural models is that some of the operations they support may produce solids whose boundary cannot be represented exactly in a closed mathematical form suitable for supporting algorithmic queries for downstream applications. For example, Boolean operations between offset solids are typically performed using approximations because intersections between offsets of curved edges cannot be computed exactly.

Computing Intersections

The choice of the geometric domain of a modeler is important because it may affect the accuracy of the analysis results and hence its suitability for specific applications and performance and hence its usability for interactive design and model inspection. The most complex problem in solid modeling is the computation of the *intersections* of the faces that bound solids or primitive half-spaces.

As discussed in the next section, the solution is relatively simple when the solid's boundary is approximated by a triangle *mesh*. Unfortunately, the lack of fidelity of such an *approximation* may invalidate some of the analysis results, unless it is compensated by the use of an exorbitant number of small facets. For example, a cylindrical peg may freely rotate in a cylindrical hole of a slightly larger ra-

dius if both surfaces are modeled using natural quadrics (i.e., cylinders). Using faceted approximations for the peg and the hole may lead to the wrong conclusion that the peg cannot rotate or does not even fit. If we wish to increase accuracy while remaining in the polygonal domain, the mesh may be 1) refined through an iterative subdivision process, which converges to a smooth surface, or 2) regenerated from a CSG model by using a higher tessellation of the primitives. Computing intersections on such refined meshes leads to two problems: 1) the large number of boundary elements (triangles, edges) increases the computational complexity and 2) the results, although acceptable for entertainment and medical applications, may be unacceptable for manufacturing applications, where mathematically precise surface models are needed to ensure precision manufacturing for tight assemblies (where, for example, a peg must fit perfectly in a cylindrical hole and be able to rotate freely) or functional requirements (where the shape of a propeller must have sufficient small curvature derivatives).

Although *trimmed surfaces* are sufficiently flexible to approximate the desired shape with often much fewer faces than triangle meshes, computing their intersections involves elaborate mathematical techniques (28) and algorithms that are significantly slower and less reliable than their counterparts for triangular geometries.

Natural quadric surfaces, which have an implicit equation (e.g., $PQ^2=r^2$ for a sphere) and a biparametric formulation (for example, a cylinder is a family of parallel rays), offer an attractive compromise between polyhedra and trimmed patches, because they provide mathematically exact representations for the majority of faces found in manufactured objects and lead to closed form expressions for their intersection curves and to low-degree polynomial solutions for the computation of the points where three surfaces intersect. Unfortunately, these surfaces cannot model precisely the numerous *fillets and blends* found in most manufactured parts (74, 75). They also cannot model sculptured or free-form surfaces that appear in many objects, especially those that must satisfy esthetic requirements, such as car bodies.

Hence, intersection calculation modules must support intersections between natural quadrics (which include the planar faces of triangle and polyhedral meshes), between biparametric patches, and between surfaces of both types (28).

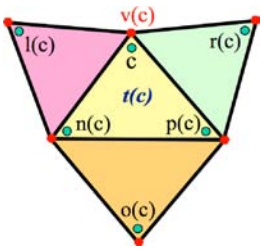

Figure 8. A corner c and operators identifying its vertex v(c), triangle t(c), and neighboring corners.

Numeric Accuracy

The numeric values that are stored to represent the precise shape and position of the surfaces or trimming curves and their intersections are rounded to the nearest value that can be represented in the digital format selected by the developer of the system. The most common formats are *floats*, *doubles*, *integers*, or *rationals* (76). Floats (floating point representations) cover a wider range of values, but their worst-case round-off error is relative (i.e., it grows with the magnitude of the value). Integer numbers, when scaled and offset properly by a judicious choice of units and of the origin, provide a denser and uniform coverage of a desired modeling range, and hence lead to lower and better controlled worst-case round-off errors. Doubles and rationals offer a much higher accuracy than floats and integers but slow down computation and increase storage. Numeric calculations with any of these formats generate and propagate round-off errors. The developers of a modeling system must ensure that these round-off errors do not lead to logical errors, to software crashes, or to wrong design decisions (77). Exact arithmetic packages do not suffer from round-off problems (78) but are usually only effective for polyhedral geometries and significantly slower, unless used with quantized parameters (79).

TRIANGLE MESHES

The triangle meshes (or simply *meshes*) studied in this section are an important category of BReps. They are arguably the most common geometric representation of solids, because they are used as auxiliary representations that approximate curved solids in most solid modelers for rendering and exporting the models to various applications and as primary representations of solids in most virtual reality, animation, entertainment, architecture, and other applications. They facilitate the implementation and accelerate the performance of many algorithms. For example, the GPUs of hardware graphics accelerators have been tuned to render triangles very fast and may be programmed to perform additional computations on triangle meshes (such as Booleans (80), cross sections (81), or interference detection (82)). Triangle meshes may be reconstructed automatically from surface samples (57, 58). Arbitrarily polygonal faces may be easily triangulated (83), and therefore, polyhedra may be represented by triangle meshes. Trimmed biparametric patches are also triangulated for rendering, ensuring that the triangulations of adjacent patches match at their common edge (84) and that the tessellation offers an optimal compromise between accuracy and speed (85). Furthermore, triangle meshes are used for assembly inspection (digital mock-up) (86) and for interactive shape editing (87). Hence, in this section, we focus on techniques for creating and processing solids bounded by triangle meshes.

Initially, we consider a *manifold* triangle mesh M with n_T triangles and n_V vertices. We assume that M is bounded and that the set of all its vertices, edges, and triangles are pairwise disjoint. (As before, we assume that edges do not include their endpoints and that the triangles do not include their bounding edges and vertices.) Hence, each edge of a valid mesh M is bounding exactly two triangles and

each vertex is bounding a single fan of triangles. Sometimes valid polyhedra, and hence valid triangle meshes may be recovered automatically from a set of faces that do not form a watertight surface (88, 89).

M divides its complement !M into two half-spaces: the *inside* i(M) and the *outside* e(M) of M. As explained, i(M) is the set of points from which rays that avoid the edges and vertices of M stab an odd number of triangles of M. Note that M is the boundary of the solid S=M+i(M) and i(M)=iS.

Representations

We explain here an example of a simple data structure for meshes and propose simple versions of the most fundamental solid modeling algorithms that operate on them. We include implementation details to stress the simplicity of these solutions and to help the reader appreciate the challenges of extending them to curved models. Initially, we focus on manifold meshes that are bounding a solid. Then we briefly discuss extensions to nonmanifold and to nonregularized models.

Although a simple enumeration of the triangles of a mesh M suffices to unambiguously define the mesh and hence the solid it bounds, most boundary representation schemes cache additional information to accelerate the traversal and processing of the boundary and combine the description of adjacent faces in order to eliminate the redundant descriptions of their common vertices. These data structures are usually complex (90), because they capture the incidence relations between a face and its bounding edges and vertices, and between an edge and its bounding vertices. Many data structures (91–93) have been studied to achieve desired compromises among 1) the extent of the topological and geometric domain (or coverage) of the modeler; 2) the simplicity, regularity, and compactness of the data structure; and (3) the complexity and efficiency of the algorithms that process the representation. We will discuss the *Corner Table* data structure and the associated operators (94) and algorithms, which have been initially designed to simplify the compression of triangle meshes (33). They may be used to produce efficient and elegant implementations of a broader set of solid modeling tasks, as illustrated below.

Assign to each vertex a different integer (*identifier*) v in [0, n_V–1] and to each triangle a different integer t in [0, n_T–1]. Each triangle t has three *corners*, each one corresponding to a different vertex of t. Assign to each corner of the mesh a different integer c in [0, $3n_T$–1]. Given a corner c, let v(c) denote (the integer identifier of) its *vertex*, t(c) denote its *triangle*, n(p) and p(c) denote the *next* and *previous* corners in t(c), and o(c) denote the *opposite* corner, as shown in Fig. 8. For convenience, we also define l(c) as o(p(c)) and r(c) as o(n(c)). We say that p(c), n(c), o(c), l(c), and r(c) return, respectively, the (identifiers of the) previous, next, opposite, *left*, and *right* neighbors of corner c, g(c) returns the point where vertex v(c) is located.

Point Membership Classification

To classify a point P that is not on M against i(M), we pick a random point O and say that P is in i(M) if it lies in an odd number of tetrahedra that each joint O to a different triangle of M. Consider a triangle (A,B,C). P is in tetrahedron (O,A,B,C) when m(O,A,B,C), m(P,A,B,C), m(O,P,B,C), m(O,A,P,C), and m(O,A,P,C) have the same sign. Let T be the tetrahedron with vertices A, B, C and D. Define

```
float m(A,B,C,D) return((AB×AC)•AD);
```

where AB denotes the vector (B–A). Note that m(A,B,C,D)=0 when T is flat and that it is positive when the vertices of triangle (B,C,D) appear in clockwise order when viewed from A. To avoid dealing with numerical errors, we pick a new perturbed location for point O if we cannot establish the sign of any of these quantities.

Building the Corner Table

The Corner Table stores all the connectivity information used by the corner operators in two arrays of integers: V[c], which contains v(c), and O[c], which contains o(c). Entries for p(c), c, and n(c) are stored as *consecutive triplets* in V and O. Hence, all corner operators may be computed from these using the following procedures:

```
int v(c) {return(V[c]);}
int o(c) {return(O[c]);}
int t(c) {return(int(c/3));}
int n(c) {if ((c%3)==2) return(c-1); else return(c+1);}
int p(c) {return(n(n(c)));}
int l(c) {return(o(p(c)));}
int r(c) {return(o(n(c)));}
```

Furthermore, given a triangle ID t, the ID of its first corner is 3t.

Typically, most file formats for triangle meshes store v(p(c)), v(c), and v(n(c)) as three consecutive integers. Hence, V may be built trivially from these formats. Although O could be computed by

```
for (each corner c) for (each corner b>c) if
((v(n(c))==v(p(b)))&&(v(p(c))==v(n(b))))
{O[c]=b; O[b]=c;}
```

a more efficient approach should be used for meshes with large triangle counts. One such approach, which has linear complexity for typical meshes, first computes the valence (number of incident triangles), valence[v], for each vertex v by simply incrementing valence[v(c)] for each corner c. Then it computes a running valence sum and allocates to each vertex a set of valence[v] consecutive entries in a temporary table C[] starting at C[bin[v]]). Note that C has a total of $3n_T$ entries. The approach stores with each vertex v the index nextC[v] to the first empty entry in the bin. Initially, nextC[v]=bin[v]. Then, for each corner c, it stores c in C[nextC[v(c)]] and increments nextC[v(c)]. At the end of this process, the bin of each vertex v contains the corners incident upon v. The integer IDs of the corresponding corners are stored in C between C[bin[v]] and C[bin[v]+valence[v]]. One can now compute O as follows:

```
for (each vertex v) {
  for (each corner c the bin of v)
   for (each corner b in the bin of v)
    if (v(n(c))==v(p(b)))
     {O[p(c)]=n(b); O[n(b)]=p(c);}}
```

For meshes with a fixed maximum valence (negligible with respect to n_V), this approach has linear cost.

Shells, Volume, and Global Orientation

One may also use the corner operators to identify the *shells* (connected components) of M as follows. Initialize the *shell count* k=0 and, for each triangles t, the *shell number* shell[t]=0. Then perform:

```
{for (each corner c)
if (shell[t(c)]==0) {firstCorner[k++]=c; swirl(c,k);}
```

using

```
void swirl(c,k) {if (shell[t(c)]==0)
{shell[t(c)]=k; swirl(c.l,k); swirl(c.r,k);};}
```

Note that this procedure computes the number k of shells and identifies a corner, firstCorner[s], for each shell s.

Most—although not all—applications assume that the triangles of each shell are consistently oriented, which means that the next corner of each corner is consistently chosen so that for every corner c, n(c)=p(o(c)). This may be easily checked during the swirl and rectified when needed by swapping the values of n(c) and p(c). Note that swapping the orientation of each triangle in a *shell* (connected component) of M, which amounts to swapping the values returned by n(c) and p(c) for each corner c, preserves consistent orientation. Hence each shell has two possible orientations.

One can show that $|m(A,B,C,D)|/6$ is the volume of tetrahedron T with vertices A, B, C, and D. The *volume* of a solid bounded by an oriented shell may be computed as $1/6\sum v(t)$, for all triangles t, using

```
float v(t) {c=3*t;
 return(m(Q,g(c),g(n(c)),g(p(c))));}
```

where Q is any fixed point. To reduce round-off errors, Q may be chosen as the average of the vertices. This formulation may be easily extended to compute the center of mass of the solid and other integral properties. Note that the volume may be positive or negative, depending on the shell orientation.

Now, we would like to assign to each shell a proper orientation and organize the shells, so that we know explicitly how many connected components the solid has, and for each component, how many cavities it has and what other components lie in these cavities. To do this, we first build a *shell containment tree*, which has the universe as root and shells as internal nodes and leaves. Each shell is included in the shell of the parent node. To test whether shell M_i is in shell M_k, we pick any one vertex of M_i and compute its membership with respect to M_k as explained above. The *depth* of a node is its graph distance from the root. Nodes of odd depth represent the outer shells of the connected components of the solid S=M+i(M). Their orientation should be flipped if their volume (computed as explained above) is negative. Their children represent the meshes that bound the holes (cavities) in the component of S. Their orientation should be flipped if their volume is positive. Once the shells are consistently oriented, the volume of each component of S is the sum of the positive volume of its outer shell and of the negative volumes of its cavities.

The corner operators listed above provide constant time access to neighboring elements on M, which speeds up many local and global calculations. For example, the edge common to triangles t(c) and t(o(c)) is flat when m(g(o(c)),g(p(c)),g(c),g(n(c)))=0 and *concave* when m(g(o(c)),g(p(c)),g(c),g(n(c)))>0. Note that the edge between points g(p(c)) and g(n(c)) is implicitly represented by both c and o(c). We say that the edge and c face each other.

Compression

To compress a mesh, several approaches (33, 95) visit the triangles in a depth-first order of a spanning tree and encode the vertices in the order in which they are first encountered by this traversal. The location g(c) of vertex v(c) is estimated using the parallelogram rule as e(c)=g(p(c))+g(n(c))–g(o(c)). Then, the difference vector g(c)–e(c) is encoded. When the coordinates are quantized to 12 bits each and the mesh is reasonably smooth, the difference vectors may be encoded with an average of about 14 bits per vertex by using variable length entropy codes.

The V array of the Corner Table of a small model contains $3n_T$ short integers or $48n_T$ bits. Edgebreaker (33) compresses the entire Corner Table (V and O arrays) to about n_T bits. It can be implemented (94) as

```
void edgebreaker (corner c) {
 if (visitedVertex(v(c)) {
  if (visitedTriangle(t(l(c))) {
   if (visitedTriangle(t(r(c))) {encode('E'); return();}
   else {encode('R'); c=r(c);};}
  else {if (visitedTriangle(t(r(c))){encode('L'); c=l(c);}
  else {encode('S'); edgebreaker(r(c)); c=l(c);};};}
else {encode('C'); c=r(c);}};
```

For each triangle, it encodes a symbol from the set C,L,E,R,S.

If the shell has genus zero (no handles), $n_T=2n_V-4$. Therefore, as the first two triangles need not be encoded, half of the remaining triangles each correspond to a vertex v(c) that has not yet been visited, and hence half of the symbols are a 'C'. If we encode them using one bit, say '0', we can encode the other four symbols using ('100', '101', '110', or '111'), which *guarantees* a compressed size of $2n_T$ bits. If we group symbols in pairs and assign a Huffman code to each pair, the encoded size usually drops to about 1 bit per triangle.

The *clers* string may be decompressed using a variety of simple and fast approaches (33, 95, 96). Both compression and decompression have linear complexity and can process an average complexity model in a fraction of a second. They have been extended to handle meshes with arbitrary topology (97).

Subdivision and Simplification

When a lack of accuracy is acceptable, to improve compression or to accelerate rendering or other applications, a mesh may be coarsened (simplified) by iteratively merging adjacent vertices and removing degenerate triangles (83, 98). Each step *collapses* the edge facing some selected corner c (Fig. 9). Typically, at each step, the next edge to be collapsed is the one that minimizes a bound on the maximum Hausdorff error (83) or an estimate of the quadratic error measure (99) between the original and the resulting simplified mesh. The collapse, which may be implemented as

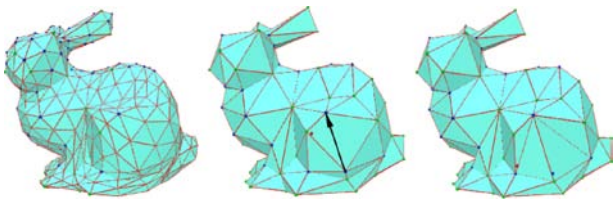

Figure 9. The original mesh (center) may be uniformly refined (left) or simplified (right) by collapsing one of its edges (arrow).

```
{b=p(c), oc=o(c), vnc=v(n(c));
 for (int a=b; a!=n(oc); a=p(r(a))) {V[a]=vnc;};
 V[p(c)]=vnc; V[n(oc)]=vnc;
 O[l(c)]=r(c); O[r(c)]=l(c);
 O[l(oc)]=r(oc); O[r(oc)]=l(oc);}
```

ensures that corners of the two triangles t(c) and t(o(c)) and the vertex v(p(c)) are no longer referenced by any corner of the simplified mesh. When all desired simplification steps have been performed, the triangles and vertices that are not used by the simplified mesh are eliminated from the Corner Table during a simple a posteriori compaction process of the V and O tables and the vertex table.

Inversely, a coarse mesh (which may have been produced through simplification or through coarse sampling) may be refined into a smoother looking mesh. A uniform *refinement* (i.e., *subdivision*) step splits each triangle into four as follows:

```
for (c=0; c<3*nT; c=c+3) {
 V[3*nT+c]=v(c);
 V[n(3*nT+c)]=w(p(c));
 V[p(3*nT+c)]=w(n(c));
 V[6*nT+c]=v(n(c));
 V[n(6*nT+c)]=w(c);
 V[p(6*nT+c)]=w(p(c));
 V[9*nT+c]=v(p(c));
 V[n(9*nT+c)]=w(n(c));
 V[p(9*nT+c)]=w(c);
 V[c]=w(c);
 V[n(c)]=w(n(c));
 V[p(c)]=w(p(c));};
 nT=4*nT;
```

assuming that w(c) returns the index to the new vertex introduced by splitting the edge facing c. The location of the old and new vertices may then be computed using one of several proposed masks (100, 101) to achieve the desired compromise between smoothness and fidelity (Fig. 9, left).

Sharp edges and vertices that may have been chamfered by random sampling may be restored and added to the coarse mesh (102) and preserved as sharp edges during subsequent subdivision steps (103).

Intersections

One of the most challenging problems in solid modeling is the computation of the intersection curves between the boundary of two solids. Such curves may, for example, provide the *trimming model* for the faces of a Boolean combination of the solids. Many approaches have been developed for computing such curves. We include a simple one here and then discuss the challenges of extending it to singular situations and of coping with numeric round-off errors.

Consider five points, A, B, C, P, and Q, where no four are coplanar. Let T be triangle (A,B,C) and E be edge (P,Q). T and E intersect when

```
Boolean hit(A,B,C,P,Q) {
 return((m(P,A,B,C)>0) && (m(P,Q,B,C)>0)
 && (m(P,A,Q,C)>0) && (m(P,A,B,Q)>0) &&
 (m(A,B,C,Q)>0));}
```

is true. One can use *hit* to compute the intersection of triangle meshes M and N as follows, again assuming that no four vertices are coplanar. First, we use

```
{for (each triangle t of M)
 for (each corner c of N) {
  if ((c>o(c))&&
  hit(g(3*t),g(3*t+1),g(3*t+2),g(p(c),g(n(c))))
   W_M+=(t,t(c),t(o(c));}
```

and

```
{for (each triangle t of N)
 for (each corner c of M) {
  if ((c>o(c))&&
  hit(g(3*t),g(3*t+1),g(3*t+2),g(p(c),g(n(c))))
   W_N+=(t,t(c),t(o(c));}
```

to compute the lists W_M and W_N of triplets of triangle indices. Note that the first triangle of each triplet of W_M is a triangle of M and the other two are triangles of N. Similarly, the first triangle of each triplet of W_N is a triangle of N. Also note that each triplet defines a vertex where three triangles intersect (or, equivalently, where an edge of one mesh intersects the triangle of the other mesh). The approximate location of the vertex may be computed as the intersection point between the line through P and Q and the plane through A, B, and C. But this computation involves a *division* and hence numeric round-off. However, the hit test does not require a division and hence may be computed exactly by using a fixed-length integer-arithmetic.

We sort the triplets into *loops* so that two consecutive triplets in a loop share two triangles. Each loop defines a trimming curve (i.e., intersection between M and N). With the non-coplanarity precaution, these loops are manifold and decompose both M and N into faces (subsets of homogeneous membership with respect to the inside of the other mesh). In other words, a face of M, which may be bounded by zero or more trimming curves, is either entirely in i(N) or entirely in e(N). The membership classification of face F of M is defined by the parity of the number of intersections of N with a ray from a point of F, as discussed above. Note that only one ray need to be processed for each *shell* of M, because the membership of one face may be recursively propagated to other faces of the shell: Two adjacent faces have opposite membership.

To produce the mesh bounding the Boolean *intersection* between the solids bounded by M and N, we select the faces of M in N and the faces of N in M. For the union, we select faces of M not in N and faces of N not in M. For a difference, we select faces of M not in N and faces of N in M.

How do we identify and represent faces? We first compute the trimming curves of each shell of M and of N. To produce a triangulation of the faces, we would need to triangulate each triangle that is traversed by a trimming curve. This process is delicate, because a triangle T may be traversed by several trimming curves and more than

once by a single curve. To compute the correct topology of the arrangements of the trimming curves in T, we must order their entry and exit points around the perimeter of T. (This also may be done without numeric error by using fixed length integer arithmetic.) The trimming loops decompose T into subfaces, and we need to triangulate each subface. Now, we can identify the faces of each shell by using a slightly modified version of the *swirl* procedure used above for identifying shells. The modification simply prevents *swirl* from crossing a trimming curve. Hence, when we triangulate the subfaces of T, we must record which of the corners of the triangulations are facing an edge of the trimming curve.

We have pointed out that this approach is free of numeric errors when fixed precision integer arithmetic is used and when the *general position* (non-coplanarity) conditions are met. Because all topological decisions (triangle/edge intersection and order of intersections around the periphery of a triangle) can be derived from the signs of 3×3 determinants, a fixed length arithmetic is sufficient.

Furthermore, the simulation of simplicity (SoS) (104) approach may be used to extend this solution to *singular position* cases where four vertices may be coplanar. SoS produces a globally consistent sign assignment to zero-valued determinants. Although it allows algorithms designed for general position to work with singular position data, it increases the computational cost and produces shapes and topologies that although valid may be incorrect. For example, the union of two cubes, stacked one on top of the other, may produce two components or a single component with overlapping faces that may be thought of as a fracture along a portion of the common face.

The cost of computing all the triplets in the above approach is quadratic, or more precisely proportional to the number of elements in M and N. If we had a starting triplet for each loop, we could trace each loop, with linear (output-sensitive) cost, using signs of 3×3 determinant to identify the next triplet (32). Hence, the main challenge is to devise acceleration techniques for finding all loops. For example, hierarchical or uniform space partitioning may be used, because edges of N in one cell may not intersect triangles of M that are in another disjoint cell. Unfortunately, when testing for interference in tight assemblies or when computing the symmetric difference between two similar solids, a large number of edge/triangle intersections will not be rejected early by this approach and must still be processed.

Topological Extensions

Although more elaborate data structures have been developed for more general polyhedra with polygonal faces that need not be convex and may even have holes, it is often advantageous to triangulate the polygonal faces (105) and use the representations and algorithms for triangles meshes, such as those discussed above. The artificial edges introduced by this triangulation of polygonal faces may be identified when needed using coplanarity tests (as discussed above) or using a marker on the corners that face them.

For simplicity, we have assumed so far that the mesh is manifold. The Corner Table may be extended to represent the boundaries of nonmanifold solids as follows. Consider an edge E with 2k incident triangles. Let c be a corner facing E. Only k–1 of the corners facing E are suitable candidates for o(c) if we want to ensure a consistent orientation. The MatchMaker process (106) computes an optimal assignment of the o() operators so that a manifold mesh could be obtained by replicating some nonmanifold vertices (at most one per connected component of the union of nonmanifold edges and vertices) and by perturbing their location by an infinitely small amount. In practice, the vertices are not perturbed; hence, in this *pseudo-manifold* representation, two vertices with different IDs may be coincident and two different edges may be coincident. Algorithms that assume that all vertices and edges are disjoint need to be adjusted to work on such pseudo-manifold BReps.

Finally, as mentioned in the Introduction, one may wish to support nonregularized sets. Consider a finite *arrangement* of planes. It defines a set of regions (3-cells), faces (2-cells), edges (1-cells), and vertices (0-cells). As before, these cells do not contain their bounding lower dimensional cells and are hence pair wise disjoint. Now, assign a label to each cell. The union of all cells with the same label forms a feature. The arrangement and the labels define a *structured topological complex* (STC) (107), which generalizes the notion of *simplicial complexes*. Various data structures have been proposed for representing the cells of such a complex and operators for traversing it (many are reviewed in Reference 92). Note that such an approach is expensive, because the number of cells in the arrangement grows as the cube of the number of planes. In fact, many cells could be merged with neighbors of identical label using the topological simplifications proposed in Reference 37.

The simplified STC can be compactly represented using a simple and compact extension of the Corner Table constructed as follows. First, triangulate all faces. Then, make two copies (front and back) of each triangle (one with each orientation) and store with each corner c the identifier b(c) of the corresponding corner on the opposite orientation triangle in the B table. Shells may be recovered and arranged into a *shell containment tree* as explained above. Nodes of odd depth represent the outer shells of 3-cells. Nodes of even depth represent the shells that bound their cavities. The *dangling edges* and vertices that are not in these shells must each be assigned to a particular 3-cell (including the infinite outer cell). The original Corner Table operators support the traversal and processing of the shells. The new b() operator provides a tool for moving from a shell of one 3-cell to the shell of an adjacent 3-cell.

Also, the alternation of the o() and b() operators may be used to traverse the triangles that are incident upon a given edge in order around that edge.

CURVED BREPS

In this section, we briefly discuss the challenges of extending to curved BReps the mesh modeling techniques presented above.

Representation

First, consider a deformed version of a triangle mesh, where each edge is possibly curved and where each triangle is a smooth portion of a possibly curved surface. If we use the Corner Table to represent the vertex locations and the connectivity, we need to augment it with a description of the geometry of each edge and of each triangle. Subdivision rules may be applied to refine each triangle and each edge. Hence, the curved elements (edges, faces) may be represented implicitly as the limit of a *subdivision* process applied to a coarse *control triangle mesh*. As an alternative, notice that each shell of a triangle mesh that bounds a solid has an even number of triangles. The triangles may be paired to form *quads*. Each quad may be defined by a biparametric polynomial or rational mapping of the unit square, as discussed above. For example, a patch could be a bicubic Bezier or B-spline patch. The difficulty is to ensure the desired degree of continuity across edges and at vertices. The desired boundary may also be defined *implicitly* (108) as the iso-surface of a smooth three-dimensional scalar field that interpolates samples either using a tetrahedral mesh [A-patches (109)], a global function (radial basis function (110), R-function (111)), or a piecewise fit (moving least-square (112)). Or directly by an implicit equation. For example, a sphere of center C and radius r can be expressed as the sets of points P satisfying $PC^2=r^2$.

The faces of a solid defined as a Boolean combination of curved solids may be subdivided into triangles or quads and represented by approximating parametric patches or may be represented as a *trimmed surface* by a reference to the host surface (original patch) on which they lie, and by trimming loops of curved edges. The edges of a solid typically lie on the intersection curves between two surfaces and sometimes on singular curves (cusps) of a single surface. A simple edge, such as a line segment or a circular arc, may be represented by its type, parameters, and position in space. More complex edges are often approximated by piecewise-polynomial parametric curves, either in 3-D or in the 2-D parameter space of the host surface. Exact, closed-form parametric representations for the intersection of natural quadric surfaces were first derived in the late 1970s at the University of Rochester for the PADL-2 modeler (66). The intersections of these edges with implicit polynomial surfaces can be computed efficiently by substituting the parametric expressions, (x(t),y(t),z(t)), of a point on the curve into the implicit polynomial equation for the surface, f(x,y,z)=0, and solving for t using an efficient numeric polynomial root finder.

For more general surfaces, the trimming loops cannot be computed exactly (as discussed below). Hence, representing them by an approximating curve in three dimensions would not provide a complete trimming model. For example, how would one establish whether a point on a patch lies inside the face defined by a trimming loop if that loop does not exactly lie on the patch? To address this problem, most modeling systems use two separate approximations of the trimming curves, one per patch, and represent them as two-dimensional curves in the parametric domain of the patch. These may be used to perform point-in-face membership classification in the parametric two-dimensional domain, provided that the parameter values of the point are known. Unfortunately, redundant representations may conflict due to numeric round-off errors and cause "cracks" in the boundary (63). An alternative based on solid trimming that avoids these cracks was mentioned earlier and will be discussed in the next section.

Furthermore, trimming loops may be insufficient to define a face unambiguously. For example, a circular edge on a spherical surface is the boundary of two complementary faces. These may be distinguished by storing information about which points in the neighborhood of the edge belong to the face. This neighborhood information can be encoded efficiently, as a single-bit "left" or "right" attribute, in terms of the orientation of the surface normal and the orientation of the curve (37).

Intersections

Let us now discuss the difficulties of adapting the intersections algorithms proposed above for triangle meshes to solids bounded by such curved surface meshes. Let us first look at the problem of computing the intersection between a curved edge and a curved face. Suppose that we are given a curve with parametric equations x=x(u), y=y(u), z=z(u) for the coordinates of the point C(u) on the curve and an implicit surface defined by an algebraic equation f(x,y,z)=0. *Curve/surface intersection* amounts to finding the u-roots of f(x(u),y(u),z(u))=0. Except in very simple cases the solution can only be found numerically, which implies computational cost and accuracy loss.

Unfortunately performing curve/surface intersection for all pairs of faces of one shape and edges of the other does not guarantee that all intersection loops will be detected. Indeed, small intersections loops, which may be of vital importance for assessing the validity of a mechanical assembly, could be missed. Hence, a variety of conservative techniques have been proposed to ensure that no loop is missed (40). For surveys and representative research, see References 28 and 29.

Selective Geometric Complexes

Many contemporary applications of solid modeling require dealing with *nonregularized sets* (such as lower dimensional regions of contacts between solids), or with non homogeneous point sets (such as composite-material aircraft parts and semi conductor circuits consisting of adjacent regions with different properties) (38, 113, 114). Such objects cannot be represented in a traditional solid modeler. Several boundary representation schemes have been proposed for domains that extend beyond solids (92). For example, Weiler's radial-edge represents face-edge and edge-vertex incidence relations and explicitly capture how incident faces are ordered around an edge (39). Such schemes are best analyzed in terms of a decomposition of space into cells of various dimensions (volumes, faces, edges, points) and in terms of their support for selecting arbitrary combinations of such cells. For example, the *selective geometric complex* (SGC), developed by Rossignac and O'Connor (37), provides a general representation for nonregular point sets, which can combine isolated points, edges, faces, and volumes with internal structures and cracks (cuts). An SGC model is

based on a subdivision of Euclidean space into cells of various dimensions that are disjoint, open, connected submanifolds and are "compatible" with all other cells. (Two sets are compatible if they are disjoint or equal.) Each cell is represented by its host manifold (point, curve, surface, or volume) and by the list of its bounding cells. Techniques independent of the dimension of the space have been proposed for computing such subdivisions, for selecting and marking sets of cells that correspond to a given description (such as a regularized Boolean operation between two previously selected sets), and for simplifying the representation through the removal or the merging of cells with identical markings. The SGC representation is capable of modeling sets with internal structures or sets of sets (107). These combine features that are each the union of all cells with identical attributes. Each region may correspond to a mixed-dimensional (i.e., nonregularized) set.

The SGC model does not explicitly store the circular ordering of edges around their common vertex or the circular ordering of faces around their common edge. If desired, this information may be cached in the NAIL (Next cell Around cell In List of incident cells) table (106).

CONSTRUCTIVE SOLID GEOMETRY

Constructive representations capture a process that defines a solid by a sequence of operations that combine modeling primitives or the results of previous constructions. They often capture the user's design intent in a high-level representation that may be easily edited and parameterized. CSG is the most popular constructive representation. Its primitives are typically parameterized solids (such as cylinders, cones, spheres, blocks, tori), volume features suitable for a particular application domain (such as slots or counter-bored holes), more general translational or rotational extrusions of planar regions, or triangle meshes, such as those discussed above. The primitives may be instantiated multiple times (possibly with different parameter values, positions, and orientations) and grouped hierarchically. Primitive instances and groups may be transformed through rigid body motions (which combine rotations and translations) and possibly scaling. The transformed instances may be combined through regularized Boolean operations (35): union, intersection, and difference to form intermediate solids or the final solid. These regularized operations perform the corresponding set theoretic Boolean operations and then transform the result into a solid by applying the topological interior operation followed by the topological closure. In practice, as discussed, regularization removes the hair and cut and merges the wound with the skin.

CSG representations are concise, always valid (i.e., always define a solid or the empty set), and easily parameterized and edited. Many solid modeling algorithms work directly on CSG representations through a divide-and-conquer strategy, where results computed on the leaves are transformed and combined up the tree according to the operations associated with the intermediate nodes. However, CSG representations do not explicitly carry any information on the connectivity or even the existence of the corresponding solid. These topological questions are best addressed through some form of boundary evaluation, where a whole or partial BRep is derived algorithmically from the CSG model.

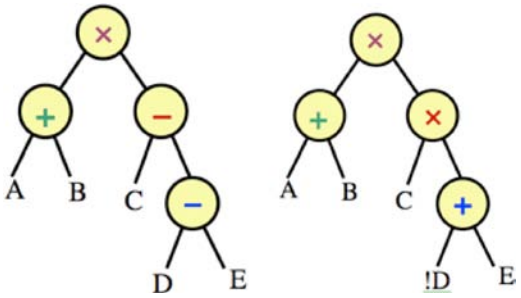

Figure 10. CSG tree for (A+B)(C–(D–E)) and its positive form (A+B)(C(!D+E)).

A blatant example is *null-object detection* (NoD) (115), where, one wishes to quickly establish whether a given CSG model represents the empty set. NoD may be used to establish whether two solids interfere (their intersection is not an null set) and whether two solids are identical (their symmetric difference is the null set).

Boolean Expressions and Positive Form

A CSG solid S is defined as a *regularized* Boolean expression that combines primitive solid instances through union (+), intersection (omitted), and difference (−) operators. Remember that !A denotes the complement of A. Such an expression may be parsed into a *rooted binary tree*: The root represents the desired solid, which may be empty; the leaves represent primitive instances; and the nodes are each associated with a Boolean operation.

To simplify discussion, throughout this section we assume that all CSG expressions have been converted into their *positive form* (Fig. 10), obtained by replacing each difference operator, L–R, by the intersection, L(!R), with the complement, !R, of its right operand R and by propagating the complements to the leaves using de Morgan laws: !!S=S, !(A+B)=!A!B, and !(AB)=!A+!B. Leaves that are complemented in this positive form (as D in Fig. 10) are said to be *negative*. The other ones are said to be *positive*.

The depth of a CSG expression is the maximum number of links that separate a leaf from the root. For example, the depth of the tree in Fig. 10 is 3. The depth of a CSG tree with n leaves is at least $\lceil \log_2(n) \rceil$. The *alternating form* of a CSG tree is obtained by grouping adjacent nodes that have the same operator in the positive form. It no longer is a binary tree. The operators alternate between union and intersection as one goes down an alternate form tree. Note that the depth of the alternate form is usually lower than the depth of the positive form. For example, the depth of the alternate form tree of the example in Fig. 10 is 2.

Point Membership Classification for CSG

Assume that point P does not lie on the boundary of any primitive. It may be classified against a CSG solid S by calling the procedure pmc(P,s), where s is the root-node of the positive form of the CSG tree of S and where pmc() is

defined as

```
boolean pmc (P,n) {
 if (isPrimitive(n))
return(pmcInPrimitive(P,n));
 else {if (operator(n)=='+')
  return(pmc(leftChild(n))||
   pmc(rightChild(n)));
else
 return(pmc(leftChild(n))&&
   pmc(rightChild(n))); }}
```

Note that the recursive calls require a stack depth that is the depth of the CSG tree. A slight variation of this approach that uses the alternate form will reduce the stack depth to the depth of the alternate form tree. The size of the stack is not an issue when points are classified one at a time, but it may become prohibitive when millions of points are classified in parallel against deep CSG trees, which happens when rendering CSG expressions using the GPU (80) to achieve interactive performance. To reduce the footprint (i.e., the number of bits needed to store the intermediate results when computing the membership of a pount), one may expand the CSG expression into a disjunctive form (union of intersections) (116) or simply process the primitives directly off the original tree, as they would appear in the disjunctive form (117). A 2-bit footprint suffices for evaluating disjunctive forms. Unfortunately, the number of terms in the disjunctive form (and hence the associated processing cost) may grow exponentially with the number of primitives. The solution is to convert the CSG tree into its Blist form (118), as shown below.

Membership classification against a natural quadric primitive is simple if the primitive is defined in a natural position (e.g. when the primitive's axes are aligned with the principal axes) and then transformed through a rigid body motion. Classifying the point against the transformed primitive is done by applying the inverse of the transformation to the point, and classifying the result against the primitive in its original position. When the primitive is defined by an algebraic or analytical inequality (for example, a sphere is defined by a second degree inequality), it suffices to substitute the point's coordinates into the inequality and evaluate its sign. More complex primitives may involve intersections of sets defined by inequalities or more general point containment tests.

Special processing based on neighborhood combinations may be necessary for points that lie on boundaries of several primitives (119). A point's *neighborhood* is the intersection of the solid with a small ball around the point. If the neighborhood is full, the point is IN; if it is empty, the point lies OUT; otherwise the point is ON. The complexity involved in computing and testing the neighborhood depends on the nature, number, and orientation of the surfaces involved.

When the primitive faces that contain the point are subsets of a single surface, the neighborhood may be represented by two bits, each indicating whether there is material on the corresponding side of the surface. Combining neighborhoods according to the regularized Boolean operations amounts to combining these bits with the corresponding logical operations (*or* for regularized union, *and* for regularized intersection, *not and* for regularized difference). The initial values of these neighborhood bits for surface points are obtained from the face orientation for each primitive whose boundary contains the point. If the two bits are set, the point is IN. If the two bits are off, the point is OUT. If the bits differ, the point is ON.

For example, when P lies on two or more host surfaces that intersect at a common curve passing through P, a curve neighborhood is used to classify P. A sufficiently small disk around P in the plane orthogonal to the curve is divided by the host surfaces into sectors, analogous to those in a pie chart. Each sector is classified against the primitives, and its classifications are simple logical values, which may be combined according to the Boolean expression. If all sectors are full, the point—and in fact the edge-segment containing it—lies in the solid. If all sectors are empty, the point is out. Otherwise the point lies on the solid. The most delicate computation in this process is the segmentation of the curve neighborhood, because it involves computing a circular order of surfaces around their common edge. The process may be numerically unreliable and mathematically challenging, if the surfaces are nonplanar and are not simple quadrics, especially when they have identical tangent planes and curvature measures at P. When all surfaces are planar and are represented by fixed-precision numbers, the neighborhood evaluation may be done exactly and efficiently (79) using fixed-length arithmetic.

Curve Membership Classification

A simple two-step strategy can be used to classify a line or curve C with respect to a CSG solid S. In the first step, the curve is segmented at places where it reaches, crosses, or leaves the boundary of any primitive. Then, the classification of each segment is inferred from the membership of its midpoint, which is computed as discussed above.

The first step requires computing curve/surface intersections. It is usually performed by substituting a parametric formulation C(t) of the curve into an implicit equation of the surface and finding the roots or through an iterative process. Sorting these t-values defines the endpoints of the segments of C.

Active Zone

Consider the *path* from the root of the positive form of the CSG tree of a solid S to an arbitrary primitive A. The *i-nodes* of A are the children of intersection nodes of the path that are not in the path. The *u-nodes* of A are the children of union nodes of the path that are not in the path. The *I-zone* I of A is the intersection of the *universe* W with all i-nodes. The *U-zone* U is the union of all u-nodes. The *active zone* (9) Z=WI–U of A is therefore the intersection of the universe W with the i-nodes and with the complements of the u-nodes of A. Note that the CSG expression of the active zone of each primitive may be derived trivially from the CSG tree by a recursive traversal (120). For example, in (A+B)(C(!D+E)) of Fig. 10, primitive A has one u-node, B, and one i-node, C(!D+E). Its active zone is Z=!BC(!D+E). Primitive E has two i-nodes, A+B and C, and one u-node, !D. Its active zone in is (A+B)CD.

Active zones have many applications, including CSG-to-BRep conversion, NoD Detection, rendering from CSG

(50), and interference detection between CSG solids (9). In particular, changes to a primitive A out of its active zone Z will not affect the CSG solid S. For instance, in our example, changing E in !D will have no affect on D–E. Changes of E in D will affect D–E but will affect C(D–E) only if they are in C.

Constructive Solid Trimming

The boundary of a CSG solid S is the union of the trimmed boundaries of its primitives, where the trimming solid for a primitive A is its active zone Z. Note that the formulation of the active zone needs to be adjusted (120) for trimming faces where the boundaries of several primitives overlap (ON/ON cases). By using the Blist form (118) of the CSG expression of the active zone, discussed below, this approach, called *constructive solid trimming* (CST), can be implemented in hardware (50) to provide real-time feedback during the editing of CSG models. For instance, assume that CST(X,T) takes as argument a primitive X and the CSG expression of a trimming volume T and renders the portion of the boundary of X in T. In the CSG tree of (A+B)(C(!D+E)), we can render the contribution of A as CST(A, !BC(!D+E)). The contribution of E is CST(E,(A+B)CD). This formulation may be used to render CSG solids in real time on graphics hardware and to highlight (in a different color) the contribution of any given primitive to the boundary of a CSG solid or the portion of a surface defined by a trimming solid represented by a CSG tree.

Blist Form

The Blist form (121) of a Boolean expression is a particular case of the reduced function graph (RFG) and of the ordered binary decision diagram (OBDD) (122) studied in logic synthesis. These are acyclic binary decision graphs, which may be constructed through Shannon's Expansion (123). The size (number of nodes) of RFGs may be exponential in the number n of primitives and depends on their order (124). Minimizing it is NP-hard. In contrast, Blist expressions have exactly n nodes and have linear construction and optimization costs, because they treat each leaf of the tree as a different primitive. Although this may not be acceptable for logic synthesis, it is appropriate for CSG rendering. Indeed, if a primitive appears several times in a CSG expression, each instance usually has a different position and hence must be processed as a different primitive during rendering.

Consider a switch A (Fig. 11a). When current is applied to the input node at the left of the triangle, if the switch is up (i.e., A is true), current will flow to the upper right exit node (Fig. 11b). When current is applied and A is false, the switch is down and current flows to the lower output node (Fig. 11c). Hence, the top output represents A and the bottom !A (Fig. 11d). We can then wire two such these switches to model a union, intersection, or difference between two primitives (Fig. 11, right).

When A is true, then current will exit from A and reach directly the top right output node of the combined circuit, regardless of the value of B. If however A is false, current will flow from its bottom output node to B. If in that case B is true, then current will flow to its upper output node.

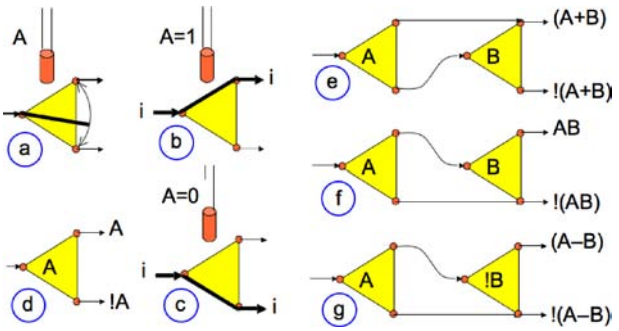

Figure 11. Blist represents each primitive as a switch (left). Two such switches may be wired to represent A+B, AB, and A-B, which is A!B (right).

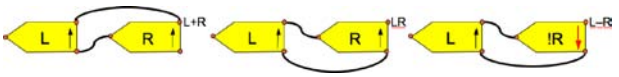

Figure 12. Blist circuits for subexpressions L and R can be combined to model Boolean operations.

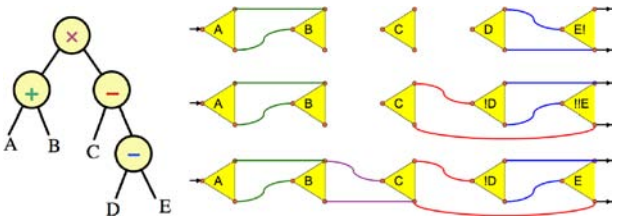

Figure 13. The expression (A+B)(C–(D–E)) may be represented by the CSG tree (left). We first wire (A+B) and (D–E) as shown top right. Note that E is negative. Then we wire (C–(D–E)). Note that the wiring of (D–E) is inverted and its primitives complemented. Now E is positive again. Finally, we wire the two expressions together as an intersection (bottom right).

Assume now that we have already built such Blist circuits for two CSG subexpressions, L and R. We can wire them (Fig. 12) to model L+R, LR, and L!R. The wiring for the union and intersection operators is identical to those for individual primitives. The wiring for L–R is the wiring for the intersection, because L–R=L!R, but to obtain the complement !R of R (right), we need to flip all its wires and complement the primitives according to the de Morgan laws.

To better understand the process of converting a CSG expression into its Blist form, consider the example of Fig. 13.

A simple two-pass linear-cost algorithm for extracting the Blist of a CSG tree is presented in Reference 50. With each primitive X of the CSG tree, it associates three labels: a label X.n assigned to the primitive, the label X.t of the primitive reached by the top wire of the triangle of X, and the label X.f of the primitive reached by the bottom wire.

To classify a point P against a Blist, we simply follow the wiring (labels). If we reach a primitive X, we test P against X. If the membership of P with respect to X is IN (*true*), we go to primitive X.t; otherwise, we go to X.f. For example, when X is A in Fig. 14, X.n= "A", X.t= "t", X.f= "B". This circuit represents the different paths that one may take to classify a candidate P against the Blist, depending on its

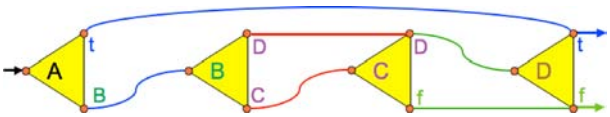

Figure 14. Blist of A+(B+C)D with X.n, X.t, and X.f labels.

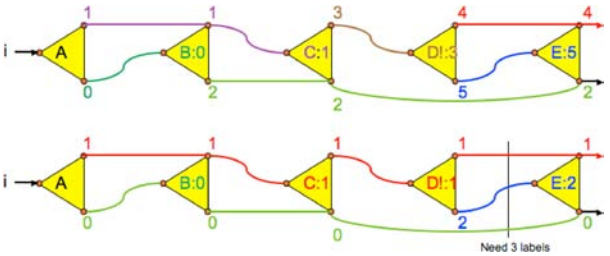

Figure 15. Top: Integer Blist labels for (A+B)(C–(D–E)). Bottom: Reusing labels reduces the number of labels from 5 to 3.

classifications against the primitives. For instance, if P∉A, P∈B, P∉C, P∈D, we would leave A by the bottom wire to B, leave B by the top wire directly to D, skipping C, and leave the whole circuit by the top wire of D. Note that the special labels, "t" and "f", of the exit wire stand for *true* and *false* and indicate the ultimate membership of P.

When using a Blist form to evaluate a Boolean expression, the *footprint* is the maximum number of bits needed to store a label. Although the size of the footprint is not a concern during a sequential evaluation of one or more Boolean expressions, it becomes important when evaluating a large number of such expressions in parallel. For example, in References 50 and 80 hundreds of thousands of Boolean expressions are evaluated in parallel, one per pixel, at each frame. The footprint for each pixel must be stored in seven stencil bits. Hence, to support hardware-assisted rendering of complex CSG solids (which may have hundreds or thousands primitives), we must reduce the number of labels used, so that they can all be encoded using different combinations of these seven bits.

First, we convert the labels to consecutive positive integers. As we do so, we keep track of the active and free integer labels (80). When a primitive (or the final result of *true* or *false*) is referenced for the first time, we give it the smallest free integer label. When a primitive with label L is reached during the Blist traversal, its label becomes available for subsequent primitives in the Blist. Reusing labels significantly reduces the number of integers used and, hence, the number of bits needed to represent each label (Fig. 15).

As the intersection and union operators are commutative, one may swap (pivot) their left and right arguments to produce equivalent Blists. This flexibility may be exploited to further reduce the number of labels needed. A pivoting strategy that makes the tree left-heavy (80), combined with a linear optimization (125), reduces the storage requirement for CSG membership evaluation to at most $\lceil \log_2 j \rceil$ bits, where $j = \lceil \log_2 (2n/3 + 2) \rceil$. This saving is substantial over the recursive evaluation, which requires a stack of at least $\lceil \log_2(n) \rceil$ bits. For example, the Blist for (A+((B+((C+((D+E)+F))+G))+H))+(I+(J+(K+L)M)N) uses five labels. The Blist for the left-heavy tree D+E+F+C+G+B+H+A+(((K+L)M+J)N+I) uses four labels. The Blist for the optimized tree (K+L)+JM+IN+(A+(H+(B+(G+(C+(F+(D+E)))))))) uses only three labels. For example, a 2-bit Blist footprint suffices for CSG trees with up to 21 leaves and 4 bits suffice for up to 98,301 leaves.

Boundary Evaluation

Consider a solid S defined by a regularized Boolean operation on two argument solids A and B, which are both defined in CSG. The boundary of S may be constructed by computing the intersection curves of the boundaries of A and B and by using them to trim the desired portions. For example, if S=A+B, we discard the portions of the boundary of A inside B and vice versa. This process is called *boundary merging*. If the boundaries of A and B are not available because one or both are CSG expressions, they may be derived recursively through an *incremental boundary evaluation* by merging boundaries up the tree, starting at the primitives.

The BRep of S can also be obtained directly from its CSG by a nonincremental boundary evaluation process. We describe briefly one such nonincremental boundary evaluation algorithm. Typically, faces of CSG solids are represented in terms of their host surface and their bounding edges. To compute the edges of a CSG solid S, we apply the generate-and-test paradigm. First, compute the intersection curves between all pairs of surfaces that support primitive faces. Then partition these curves into subsets that are in IN S, OUT of S, or ON S by using curve/solid membership classification algorithms. Segments with a curve neighborhood that is neither empty nor full form the *edges* of the solid. The details of this process are presented in Reference 79 for CSG models composed of polyhedral primitives. By keeping track of the association between curves and surfaces, and between vertices and segments, a full BRep may be inferred from the result of the curve classification process. Edges are chained into loops, which are used to trim the solid's faces. The representation of the loops may be simplified by merging adjacent curve segments, when no more than two segments share a vertex. Other algorithms for boundary evaluation and merging typically also use the generate-and-test paradigm but may be articulated around vertices, edges, or faces (see Reference 48 for an example and Reference 119 for further references). Boolean operation algorithms are among the most complex in solid modeling. They also are among the most difficult to implement reliably, in the presence of round-off errors.

Representation Conversion

Boundary evaluation is an important example of *representation conversion*, from CSG to BRep. The inverse process of computing semi-algebraic expressions for solids represented by their boundary is also very important, because it provides algorithms for maintaining consistency in multi-representation systems. When a BRep is edited, for example, to create or adjust a feature, the modifications must be reflected on the associated CSG. The 2-D case is fairly well understood (126).

Other representation conversion algorithms are useful as well. For example, point membership classification for points on a regular grid can be used to construct a spatial enumeration approximation for a solid, which in turn facilitates the computation of the solid's mass and moments of inertia (14) and of interferences. As another example, conversion into a cell decomposition in which all the cells are "slices" perpendicular to a given direction is needed to drive rapid prototyping machines (127). It can be accomplished by classifying a set of parallel planes with respect to a solid (81). The portions of the planes inside the solid are the desired slices.

Efficiency Enhancements

Set membership classification and CSG to BRep conversion algorithms perform a very large amount of computation when compared with their output sizes. Many of these algorithms are based on the *generate-and-test paradigm* and spend much of their time generating, testing, and rejecting. Performance-enhancement methods play a crucial role in eliminating a large fraction of unnecessary tests. In essence, these methods ensure that entities are compared only when there is a chance that they may interact.

Two of the widely used efficiency-enhancement techniques are plane sweep algorithms from computational geometry (which generalize the earlier scan line algorithms developed in computer graphics) and grid-based spatial directories. A plane sweep algorithm maintains a list of active entities that can potentially interact and updates the list incrementally, as the plane advances in its sweep of the whole space. Only active entities need to be compared. A spatial directory decomposes the space into cells (either of constant size or arranged hierarchically) and associates with each cell the entities that intersect it. Only those entities associated with the same cell are compared and processed.

When classifying a candidate set X (point, curve, primitive's boundary) against a CSG expression S representing a solid or the active zone of a primitive in a CSG tree, one may prune S to eliminate *redundant* primitives as follows. Let bound(X) be a simple set (ball, axis aligned box, linear half-space) containing X. Let A be a positive primitive (or the complement of a negative primitive) in the positive form of S. If $A \cap \text{bound}(X) = \emptyset$, we can replace A by $\emptyset$ in S. If $\text{bound}(X) \subset A$, we can replace A by the universe Ω in S. Then, we can usually perform further simplifications up the positive form tree using the following substitutions: $\emptyset \cap B \rightarrow \emptyset$, $\emptyset \cup B \rightarrow B$, $\Omega \cap B \rightarrow B$, $\Omega \cup B \rightarrow \Omega$, where B is any leaf or node.

Constructive Nonregularized Geometry

Extensions of the Boolean operations to *nonregularized* sets, to sets with internal structures, and to sets of dimension larger than three are important for many applications. A constructive model for creating sets of sets from higher level input, and for querying the existence and nature of intersections or adjacency relations between regions was developed by Rossignac and Requicha in their *constructive nonregularized geometry* (CNRG) model (128). Users of applications can instantiate primitives and specify how they should be combined to create a hyperset that is the union of mutually disjoint regions. CNRG regions correspond to expressions involving nonregularized Boolean and topological operations on primitive regions. Rossignac's STC (107) add to the CNRG representation the capability of creating and interrogating simultaneously several decompositions of the three-dimensional space. For example, the same arrangement may be decomposed into volume and surface features important to manufacturing, into design features and their intersections, into functional components, or into finite elements for analysis. These decompositions are compatible, in that a cell belongs to a unique set in each decomposition. The user or the application program can manipulate primitive regions and their intersections in a uniform manner, independently of the representation or approximation of these entities in a particular modeler.

PARAMETERS, CONSTRAINTS, AND FEATURES

Regardless of the representation scheme used, building and editing a model for a complicated solid is nontrivial. Finding the size and position parameters that are needed to ensure that geometric entities are in desired relationships often is quite difficult (2). And so is ensuring that such relationships are preserved when an object is edited. In addition, the primitives provided by the representation methods discussed earlier tend to be relatively low level and not directly connected with the application domain. The representational facilities discussed in the following subsections address these problems.

Parametric Models

The size and position parameters used to instantiate the primitives needed to represent an object provide a natural parameterization for the object. However, there is no guarantee that a change of parameter values will produce an object that is valid and consistent with the designer's intent. The first of these problems can be solved easily by using a CSG-based parameterization, which ensures that an instance of a parametric solid model is always valid. The second problem is more pernicious. Some design constraints may be expressed by substituting the parameters of the primitives or of the transformations by symbolic parameter expressions. This approach was first demonstrated in the 1970s with the PADL-2 solid modeling system (66) and is now in widespread use. In addition to symbolic expressions, Rossignac proposed to link CSG parameters to procedures specified by the user in terms of geometric constraints (129). Each constraint corresponds to a transformation that brings the host surface of a primitive face into a specified relationship (contact, distance, angle) with the host surface of a primitive not affected by the transformation. These approaches rely on the user for producing and sorting a sequence of steps that evaluate symbolic expressions or that compute transformations to achieve the desired effects (68). The user's actions are akin to the writing of a macro that takes some input parameters and produces the desired solid. The macro defines a family of solids, also called a "parametric solid model." The user is responsible for designing the correct macro, ensuring that executing such a sequence for reasonable values of the input param-

eters produces a solid that meets the designer's intent. This is not always easy to achieve, because the required symbolic expressions may be hard to find, and a transformation may violate a previously achieved constraint. For example, one can use Rossignac's CSG constraints to first specify that a set S of CSG primitives (that typically form a feature) should be translated in some given direction until one of them, say, cylinder A, becomes tangent to another cylinder B not in the set S. Instead of tangency, one may for instance specify the distance between the two cylinders. Assume now that the designer also wishes to achieve another tangency or distance relation between a cylinder C of S and another cylinder D not in S. The second constraint can be achieved without violating the first one by either rotating S around the axis of B or translating it along that axis until the second constraint is satisfied. Rossignac has provided a variety of closed-form solutions for computing the corresponding rotations or translations for the natural quadric surfaces. Note however that, in this approach, the designer is responsible for defining an order of transformations, each specified by a set S of primitives, by the two surfaces (one in S and one not) on which the constraint is defined, and by additional parameters defining and constraining the authorized rigid motion.

VARIATIONAL GEOMETRY

In contrast, the variational geometry approach does not require the user to define an order for constraint-achieving operations, nor even to define all the constraints. A user can specify symbolic expressions that define relations between two or more parameters. In addition, the system infers automatically bidirectional constraints from all singular situations (such as parallelism, orthogonality, or tangency) that are detected on a nominal model. A constraint solver (130) adjusts the model to meet all the constraints simultaneously. This process may involve numeric iterations to solve the corresponding system of simultaneous, nonlinear equations. Because the constraints, such as edge dimensions or angles between faces, are typically expressed in terms of boundary entities, and because it is difficult to relate these to the input parameters of a CSG model (131), variational geometry is typically used in conjunction with a parameterized boundary representation. The variational geometry approach is popular for 2-D drafting and for designing solids that are extruded from 2-D contours, but its application to more general 3D shapes still suffers from several drawbacks. Performance problems are due to the large number of nonlinear equations in many variables that must be solved simultaneously. A small change in one parameter may lead the iterative techniques to converge to a local minimum that is significantly different from the previous configuration, and surprise or confuse the user. In an over-constrained situation, a user will have trouble deciding which constraints to relax for the system to converge to a solution. Finally, users may create invalid boundary representations, because no practical techniques exist for computing the bounds on parameter values for which the model remains valid (132). One solution is to let the user constrain both the dimensions and the topology (133).

Features

Features provide a higher level and domain-targeted vocabulary for specifying shape-creating operations, and for identifying the model's elements from which the parameters of symbolic expressions or manufacturing plans are to be derived. Models may be constructed by a sequence of operations that create additive or subtractive volumetric features. The nature of these features may vary widely with the application domain. *Volumetric features* may be viewed as higher level parameterized CSG primitives that are relevant to a specific domain. For example, dove-tail slots, profiled pins, blends, or chamfered holes are useful features for machined parts. Their creation sequence and parameters can be captured in a CSG representation with union and difference operations and feature leaves. However, the geometry of a feature created by one operation may be partially or totally obliterated by a subsequent operation (129). Consequently, these design features cannot be used directly for analysis or other computations without a verification step, or conversion into a standard (i.e., non-feature-based) model.

A feature-based representation can be converted into a BRep via a general-purpose CSG-to-Boundary conversion. However, many systems provide the user with immediate feedback based on direct modification of the boundary. This is fast but not without danger. When tweaking the parameters of one feature, the faces that bound the feature may intersect faces of other features in unanticipated ways. Also, if the volume of an additive feature overlaps the volume of a subtractive feature, precedence information is needed to decide whether to keep or remove the intersection of the two features. This amounts to using a CSG-like structure for evaluating the boundary of the resulting solid.

Because feature faces may be altered, split into several connected components, or even destroyed by the creation of other features, it is important to provide mechanisms for connecting the feature entities with the corresponding faces of the resulting solid. Furthermore, the user or an automatic feature-extraction process may identify collections of faces or volumes in the solid or in its complement as features that are important for further design activities or for downstream applications, but that do not correspond to a single-feature creation operation. For example, adding two ribs to a solid may create a slot feature, which is a more appropriate abstraction for manufacturing process planning than the ribs. Techniques developed by Requicha and his students at the University of Southern California address issues of automatic feature conversion and dependencies between converted and original features (10).

In essence, the input (or design) features are converted either manually or automatically into other, application-dependent features. The challenge is to capture the results of these conversions in a form that *persists* when the parameters of the model are changed. Otherwise, all user interactions or annotations with the converted features are lost and must be re-entered manually after each parameter modification or engineering change to the geometry of the model. The difficulty of this challenge may be illustrated by considering two versions, S and S', of the same CSG model, although with different parameter values. Which face F' of

S' corresponds to a given face F of S? Because the boundary of a CSG solid is a subset of the boundaries of its primitives, F may be associated with the faces of CSG primitives whose intersection with F is two-dimensional and F' may be recovered as the contributions to S' of the corresponding primitive faces, given the CSG parameters for S'. This approach suffers from three difficulties: 1) There may be several primitive faces in S that overlap with F, 2) some of these faces may not be responsible for contributing F, and 3) F may only be one connected component of the contribution of a set of primitive faces in the CSG model of S. The first two difficulties have been addressed by Rossignac using an extension of the *active zone* (9), which provides a simple CSG expression for the region of space where the boundary of a CSG primitive contributes to the boundary of the solid. The third difficulty may be addressed by using Boolean or topological filters to distinguish one connected component of the boundary of a solid from another (68). Except for limited situations, no reliable and automatic technique is currently available for deriving such filters.

MORPHOLOGICAL TRANSFORMATIONS AND ANALYSIS

A Boolean operation always returns a solid whose boundary is a subset of the union of the boundaries of the arguments. Several transformations and operations that create new surfaces have been considered for extending the capabilities of CSG systems. However, many of these operations are difficult or impossible to integrate in the divide-and-conquer paradigm for CSG and in some CSG-to-boundary conversion algorithms, because simple calculations, such as point-containment, may not be easily obtained by combining the results of similar calculations on CSG primitives.

Warps

Simple nonlinear transformations may twist an object by applying to each point in space a 2-D rotation around the z-axis with an angle that is a function of the z-coordinate (134) or may bend an object by interpreting the Cartesian x and y coordinates of a user-defined local coordinate system as the radius and angle in a cylindrical coordinate system. More complex free-form deformations have been proposed that, for example, deform space by using a control mesh (40), screw-motions between a grab and a release pose (87), or a family of screw-motions controlled by a ribbon (16) (Fig. 16). These deformations are usually applied to the vertices of triangle meshes or of control meshes of curved surfaces.

Minkowski Sums and Morphs

The Minkowski sum $A \oplus B$ of two solids A and B is the result of sweeping one solid over the other. Mathematically, it is defined by $a+b$, $a \in A$, $b \in B$, where point $a+b$ corresponds to the translation of point a by the vector from the origin to point b. Kaul and Rossignac used linear Minkowski combinations $C(t)=(1-t)A \oplus tB$ to construct parameterized shapes that smoothly interpolate any two polyhedra A and B (Fig. 17). They further expanded this approach to the weighted Minkowski averages of a larger number of polyhedra (135). The user selects the shapes and orientations of the argument polyhedra. The modeling system computes the parameterized shape and animates its evolution in real time as the user explores the range of values of the interpolation parameters (weights). For example, one may animate a solid morph that corresponds to a Bezier curve in the space of polyhedra. Such tools are important for the interactive exploration of new shapes in design, for the simulation of some manufacturing processes, and for the creation of animations. Such morphs may be combined with rigid motions (Fig. 17).

Minkowski sums also play a crucial role in robotics for collision avoidance (13) and in accessibility and visibility analysis (11, 12).

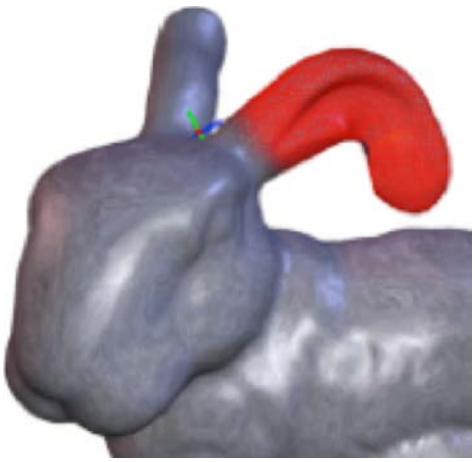

Figure 16. The red ear was selected by the user and bent using Bender (16).

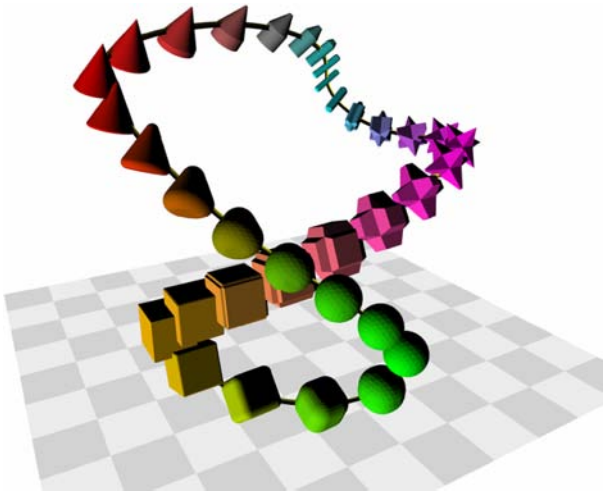

Figure 17. A solid is moving along a smooth polyscrew while morphing from one shape to the next (136).

Grow-Shrink Combinations and Tightening

Minkowski sums or differences with a ball define *growing* and *shrinking* operations on solids (Fig. 18). For instance, when B is a ball of radius r, $S \oplus B$ is the *grown solid* $S \uparrow r$

Figure 18. The original shape (center), its grown version, (right) or shrunk version (left) (137).

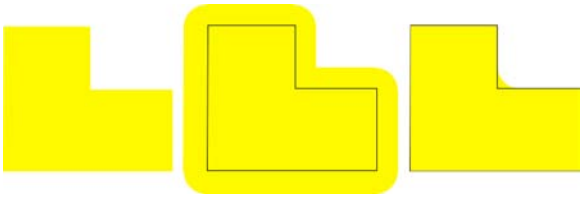

Figure 19. The 2-D shape (left) may be filleted (right) by first expanding it (center) and then shrinking the result.

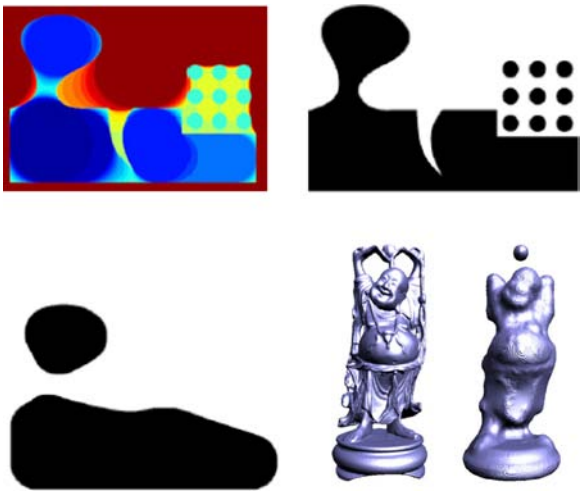

Figure 20. The stability map (top left) indicates the maximum radius of a ball that can reach the point without intersecting the shape. Tightening fills in cracks and removes constrictions and small components (top right and bottom left). Tightening of a 3-D shape (bottom right).

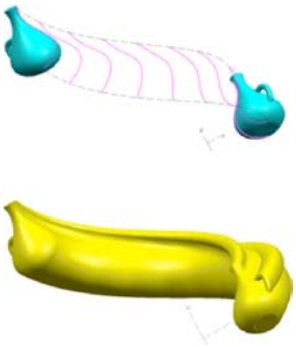

Figure 21. Two control poses (top) define a screw motion. The volume swept by the solid is shown (bottom).

defined as the union of S with all points at a distance less than or equal to r from S. The shrunk solid S↓r is the difference between S and the set of points at distance less than r from the boundary of S. The grown version of the boundary bS of S is a tolerance zone E=(bS)↑r, which has been used by Requicha to define the mathematical meaning of tolerance specifications (136).

Combinations of growing and shrinking operations on solids (71) produce constant radius fillets and rounds. Specifically, $F_r(S)=S\uparrow r\downarrow r$ is the set not reachable by a ball of radius r that is disjoint from S. Hence the fillet operation F_r add constant radius fillets along the concave edges of S (see Fig. 19 for a 2-D illustration). Similarly, the round operation produces $R_r(S)=S\downarrow r\uparrow r$, which is the set reachable by a ball of radius r that in S. These operations always produce valid solids and may be combined with Boolean operations to limit their "blending" or "filleting" effect to the desired sets of edges.

Note that R and F operations may be combined in an attempt to round or fillet both the concave and the convex edges. Unfortunately this approach does not always work. In fact $R_r(F_r(S))$ tends to increase the volume but may leave some sharp concave edges, whereas $F_r(R_r(S))$ tends to decrease the volume but may leave some sharp convex edges.

The *r-mortar*, $M_r(S)$, of a solid S is $(bS)\uparrow r\downarrow r$ or, equivalently, $F_r(bS)$. It is a small subset of the tolerance zone E. The $R_r(F_r(S))$ and $F_r(R_r(S))$ combinations only affect S in its mortar $M_r(S)$. In fact, one or the other combination may be selected independently for each connected component of the mortar so as to minimize volume changes (138). The *stability* of a point P of space with respect to a set S is min r: $P\in M_r(S)$. It may be used for a multiscale analysis of how a shape S is imbedded in space (Fig. 20, left).

The *r-tightening* $T_r(S)$ of S (139) is obtained by tightening (i.e., reducing perimeter length in 2-D or surface area in 3-D) bS, while keeping it in $bS+M_r(S)$. Tightening provides a powerful solid modeling operator for smoothing a solid by imposing a constraint on the curvature (Fig. 20). Although some applications may require that the topology of the solid be preserved during simplification (140), one may want to use tightening to *simplify topology* to remove noise. Tightening has several advantages over other smoothing operators (141) that affect the surface outside of the mortar and deform the boundary without consideration as to its immersion in the surroundings space.

Sweeps

The volume V *swept* by S during a motion M is important in machining simulation. It is the infinite *union* of all instances of S@M(t) of S at poses M(t) produced during the motion M. (An infinite *intersection* produces an *unsweep* (142), which is also useful for design and analysis.) The computation of the boundary of V constructs candidate faces as sweeps of characteristic curves on S, where velocity is orthogonal to the surface normal. Unfortunately, in general, these curves change their shape with time. Hence, to simplify the problem of computing bV (73), one may approximate each motion with a polyscrew (piecewise helical) motion (Fig. 21). Polyscrews defined by a few control poses may be smoothened (Fig. 17) with C^2 continuity (143).

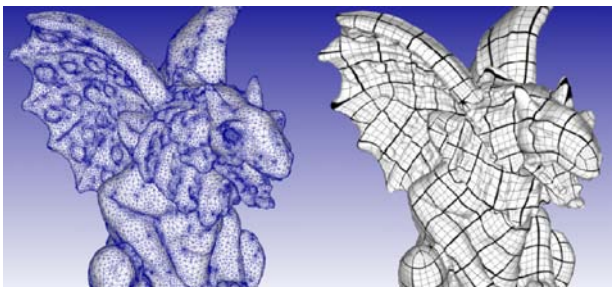

Figure 22. Original (left) and periodic quadrangulation (right) (146) (Courtesy Bruno Levy).

Resampling and Parameterization

As an alternative to the refinement and simplification operations discussed earlier, a mesh may be *resampled*. Resampling strategies may favor the regularity of vertex positions or the alignment of the edges with the directions of principal curvature.

Most approaches focus on triangle meshes. For example, to improve compression, vertices may be placed at the tips of the isosceles Edgebreaker (type 'C') triangles (102) or along uniformly spaced X, Y, or Z-rays (144). Yet quadrilateral meshes may be preferred for surface PDE simulations, especially fluid dynamics, and are best suited for defining Catmull–Clark subdivision surfaces. Manifold triangle meshes may be quadrangulated (145) using Laplacian eigenfunctions, the natural harmonics of the surface, which distribute their extrema evenly across a mesh and connect them via gradient flow into a quadrangular base mesh. An iterative relaxation algorithm simultaneously refines this initial complex to produce a globally smooth parameterization of the surface. From this, one can construct a well-shaped quadrilateral mesh with few extraordinary vertices. The *periodic global parameterization* method (146), also generates a (u,v) coordinate system aligned with the principal direction of curvatures (Fig. 22) but uses periodic variables that makes it possible to compute a parameterization that *winds* around the protrusions of the object.

HUMAN-SHAPE INTERACTION

The skills required to use a solid modeler impact the users' effectiveness. Early solid modeling systems were reserved to CAD experts in the aerospace and automotive industries. Considerable R&D efforts have improved the ease-of-use for nonexperts and the productivity of expert designers. Indeed, labor is the dominant cost of solid modeling, and many professionals involved in the design cycle are not CAD experts. Furthermore, new easy-to-use "light-weight" solid modelers are making inroads in nontraditional areas (such as electronic components or entertainment) where accessibility to nonspecialists and rapid design cycles are more important than precision. Furthermore, a complex 3-D database created by designers would be of considerable value to others employees, customers, or suppliers, who do not have the skills necessary to use a complex solid modeler. To fulfill this need, many vendors are now offering intuitive 3-D browsers that support the review and annotation of 3-D models of complex assemblies of solids. These browsers support communication in product data management activities, help illustrate maintenance manuals, or provide interactive virtual-reality experiences for marketing, styling, or ergonomic analysis. In fact, we expect that future solid modelers will be architected from interchangeable (third party) design and analysis components controlled from such a browser.

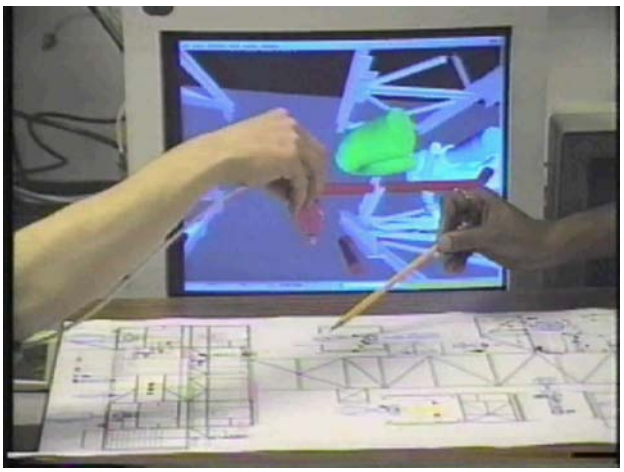

Figure 23. Rossignac's tangible camera for collaborative assembly inspection.

Advances in GUI are numerous. To illustrate their benefits we mention two examples where the use of 3-D input devices has considerably simplified the interaction between designer and scene and hence increases productivity and ease-of-use. A 6 degrees-of-freedom (DoF) magnetic tracker was used by Rossignac and colleagues at IBM Research to provide an intuitive interface for manipulating the *view* of, say, a large model of a power plant, ship, or airplane for digital mock-up inspections. In this setting (Fig. 23), while the team is watching a blueprint of the plant and a screen showing a 3-D view of it, one of the team members is manipulating the *tangible camera* (tracker) over the blueprint as if it was a small camera filming a 3-D model made to scale and positioned above the blue print. This dual visualization, which combines the traditional blueprint with an easier to disambiguate 3-D view, encourages team interaction, because others see exactly what is displayed on the screen (the context is offered by the relative position of the camera with respect to the blueprint) may point to the blueprint or even annotate it. More recently, Rossignac and students have developed a two-handed Human–Shape Interaction paradigm (Fig. 24), which uses two haptic 3-D trackers through which the user can either grab and manipulate objects, paint on them, or warp them to explore new shapes or plan heart surgeries (74).

CONCLUSIONS

Solid modeling technology has significantly outgrown its original scope of computer-aided mechanical design and manufacturing automation. It plays an important role in may domains, including medical imaging and

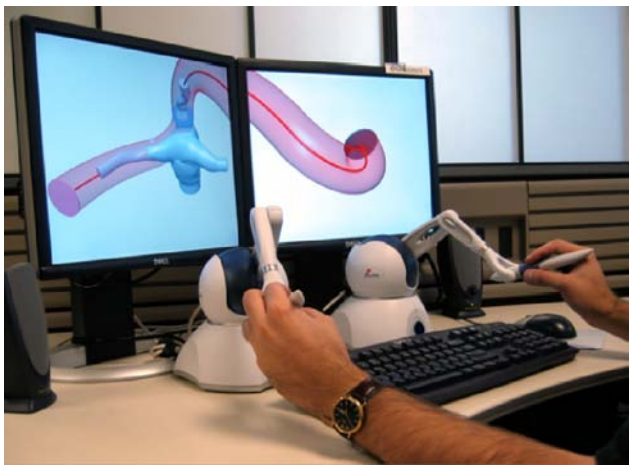

Figure 24. Rossignac's two-handed haptic interface for Human–Shape interaction.

therapy planning, architecture and construction, animation, and digital video production for entertainment and advertising.

The maturity of the solid modeling theory and technology has fostered an explosion in the field's scientific literature and in the deployment of commercial solid modelers. The dominant cost of embracing the solid modeling technology within an industrial sector has shifted over the years from hardware, to software, to labor. Today, industrial strength solid modeling tools are supported on inexpensive personal computers, software price ceased being an issue, and the progress of user-friendly graphics interfaces has considerably reduced training costs. As the theoretical understanding of solid modeling and efficient algorithms for the fundamental operations have begun to percolate toward commercial systems, research efforts are focused on making nonexpert users more productive.

The modeling task is labor intensive. For instance, the design of a new aircraft engine requires 200 person years. Although the solid modeling activity is only a small part of this cost, much of the current research attempts to make designers more effective, by supporting higher level design automation and reusability. Significant progress was recently achieved on data compatibility between different solid modelers and on the support of constraints and features encapsulated into "smart" objects that adapt their shape and dimensions to the context in which they are used.

The exploitation of the resulting models has been so far primarily restricted to designers. Spreading the access to a larger population will reduce the cost of downstream applications (such as manufacturing, documentation, and marketing) and will improve communication through out the enterprise, its suppliers, and its customers.

Total automation of a wide range of applications—an original motivation of solid modeling—has proven harder than originally expected, especially when automatic synthesis or planning is required.

BIBLIOGRAPHY

1. Lee, S. H. Feature-based Multiresolution Modeling of Solids. *ACM Trans. Graph*. 2005, **24**(4),pp 1417–1441.
2. Shah, J., Mantyla, M. *Parametric and Feature Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons: New York, 1995.
3. Requicha, A. A. G. Mathematical Definition of Tolerance Specifications. *ASME Manuf. Rev*. 1993, **6**,pp 269–274.
4. Joskowicz, L., Sacks, E. HIPAIR: Interactive Mechanism Analysis and Design Using Configuration Spaces. *Proc. of the 11th Annual Symposium on Computational Geometry SCG '95*; ACM Press: New York, 1995, pp 443–444.
5. Cutler, B., Dorsey, J., McMillan, L. Simplification and Improvement of Tetrahedral Models for Simulation. *Proc. of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing SGP '04*, ACM Press, 2004, 71,pp 93–102.
6. Voelcker, H., Hunt, W.The Role of Solid Modeling in Machine-Process Modeling and NC Verification. SAE Technical Paper No810195, Feb. 1981.
7. Elber, G. Cohen, E. Tool Path Generation for Freeform Surface Models.J. Rossignac, J. Turner, G. Allen, Eds. *ACM Symposium on Solid Modeling and Applications. SMA '93*. ACM Press, 1993, pp 419–428.
8. Gonzalez-Ochoa, C. McCammon, S. Peters, J. Computing Moments of Objects Enclosed by Piecewise Polynomial Surfaces. *ACM Trans. Graph*. 1998, **17**(3)pp 143–157.
9. Rossignac, J., Voelcker, H. Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection and Shading Algorithms. *ACM Trans. Grap*. 1989, **8**,pp 51–87.
10. Han, J.-H. Requicha, A. A. G. Integration of Feature Based Design and Feature Recognition. *Computer-Aided Design*. 1997, **29**(5),pp 393–403.
11. Spyridi, A. J., Requicha, A. A. G. Accessibility Analysis for the Automatic Inspection of Mechanical Parts by Coordinate Measuring Machines *Proc. IEEE Int'l Conf. on Robotics & Automation*, Cincinnati, OH, 1990, pp 1284–1289.
12. Spyridi, A. J., Requicha, A. A. G. Automatic Programming of Coordinate Measuring Machines *Proc. IEEE Int'l Conf. on Robotics & Automation*. San Diego, CA, May 8–13, 1994, pp 1107–1112.
13. Latombe, J. *Robot Motion Planning*, Kluwer: Boston, 1991.
14. Lee, Y. T. Requicha, A. A. G. Algorithms for Computing the Volume and other Integral Properties of Solids: I—Known Methods and Open Issues and II—A Family of Algorithms Based on Representation Conversion and Cellular Approximation. *Commun. ACM*, 1982, **25**(9),pp 635–650.
15. Feldman, B., O'Brien, J., Klingner, B. Animating Gases with Hybrid Meshes. *ACM SIGGRAPH 2005*. ACM Press, 2005, pp 904–909.
16. Llamas, I., Powell, A., Rossignac, J., Shaw, C. Bender: A Virtual Ribbon for Deforming 3D Shapes in Biomedical and Styling Applications. *ACM Symposium on Solid and Physical Modeling (SPM)*.June 2005.
17. Vivodtzev, F., Bonneau, G., Linsen, L., Hamann, B., Joy, K., Olshausen, B. Hierarchical Isosurface Segmentation Based on Discrete Curvature. *Eurographics Symposium on Data Visualization*. 2003, **40**,pp 249–258.
18. Smith, J., Hodgins, J., Oppenheim, I., Witkin, A. Creating Models of Truss Structures with Optimization. *Conference on Computer Graphics and Interactive Techniques SIGGRAPH '02*. ACM Press: New York, 2002, pp 295–301.

19. Faux, I. D., Pratt, M. J. *Computational Geometry for Design and Manufacture*. Halsted Press: New York, 1979.
20. Requicha, A. A. G. Voelcker, H. B. Solid Modelling: A Historical Summary and Contemporary Assessment. *IEEE Comput. Graph. Applicat.* 1982, **2**,pp 9–24.
21. Requicha, A. A. G. Geometric Reasoning for Intelligent Manufacturing. *Commun. ACM*. 1996, **39**,pp 71–76.
22. Spyridi, A. J., Requicha, A. A. G. Automatic Planning for Dimensional Inspection. *ASME Manufact. Rev*. 1993, **6**,pp 314–319.
23. Requicha, A. A. G. Solid Modelling: A 1988 Update. InRavani, B.; Ed., *CAD Based Programming for Sensory Robots*. Springer Verlag: New York, 1988, pp 3–22.
24. Requicha, A. A. G., Voelcker, H. B. Solid Modelling: Current Status and Research Directions. *IEEE Comput. Graph. Applicat*. 1983, **3**,pp 25–37.
25. Requicha, A. A. G., Rossignac, J. R. Solid Modeling and Beyond. *IEEE Comput. Graph. Applicat*. 1992, **12**,pp 31–44.
26. Hoffmann, C., Rossignac, J. A Road Map To Solid Modeling. *IEEE Trans. Vis. Comput. Graph*. 1996, **2**(1),pp 3–10.
27. Rossignac, J. Shape Complexity. *Visual Comput*. 2005.
28. Patrikalakis, N. Surface-to-Surface Intersections. *IEEE Comput. Graph. Applicat*. 1993, **13**pp 89–95.
29. Krishnan, S., Manocha, D. An Efficient Surface Intersection Algorithm Based on Lower-dimensional Formulation. *ACM Trans. Graphics*. 1997, **16**(1),pp 74–106.
30. Rossignac, J., Borrel, P. Multi-Resolution 3D Approximations for Rendering Complex Scenes, In *Geometric Modeling in Computer Graphics*, Falcidieno, B., Kunii, T. L., Eds., Springer Verlag: New York, 1993.
31. Cignoni, P., Montani, C., Scopigno, R. A Comparison of Mesh Simplification Algorithms. *Comput. Graph*. 1998, **22**(1),pp 37–54.
32. Navazo, I., Rossignac, J., Jou, J., Shariff, R. ShieldTester: Cell-to-Cell Visibility Test for Surface Occluders. *Proc. of Eurographics*.September 2003.
33. Rossignac, J. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Trans. Vis. Comput. Graph*. 1999, **5**,pp 47–61.
34. Besl, P. J., Jain, R. C. Three-dimensional Object Recognition. *ACM Comput. Surv*. 1985, **17**(1),pp 75–145.
35. Requicha, A. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM Comput. Surv*. 1980, **12**,pp 437–464.
36. Tilove, R. Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE Trans. on Comput*. 1980, **C-29**,pp 874–883.
37. Rossignac, J., O'Connor, M. SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries. In *Geometric Modeling for Product Engineering*, Wosny, M.;Turner, J.;Preiss, K.; Eds., North-Holland: Amsterdam, 1989, pp 145–180.
38. Kumar, V., Dutta, D. An Approach to Modeling Multi-material Objects. *Proc. 4th ACM Symposium on Solid Modeling and Applications*. ACM Press: New York, 1997, pp 336–345.
39. Weiler, K. Non-Manifold Geometric Boundary Modeling. *ACM Siggraph, Tutorial on Advanced Solid Modeling*. Anaheim, CA, July 1987.
40. Sedeberg, T., Parry, S. Free-Form Deformation of Solid Geometric Models *ACM Comput. Graph. (Proc. Siggraph)*. 1986, **20**,pp 151–160.
41. Bloomenthal, J., Wyvill, B. *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc.: San Francisco, CA, 1997.
42. Guthe, M., Balázs, A., Klein, R. GPU-based Trimming and Tessellation of NURBS and T-Spline Surfaces. *ACM Trans. Graphics*. 2005, **24**(3),pp 1016–1023.
43. Müller, M. Dorsey, J., McMillan, L., Jagnow, R., Cutler, B. Stable Real-time Deformations. *Proc 2002 ACM Siggraph / Eurographics Symposium on Computer Animation SCA '02*. ACM Press: New York, 2002, pp 49–54.
44. O'Brien, J., Bargteil, A., Hodgins, J. Graphical Modeling and Animation of Ductile Fracture. *Proc. SIGGRAPH*. ACM Press: New York, 2002, pp 291–294.
45. Melek, Z. Keyser, J. Bending Burning Matches and Crumpling Burning Paper; *ACM SIGGRAPH*; ACM Press: New York, 2006, p 131.
46. Kobbelt, L., Shapiro, V. *Proc. ACM Symposium on Solid and Physical Modeling*. ACM Press: New York, 2005.
47. Alexandroff, P. *Elementary Concepts of Topology*, Dover Publications: New York, 1961.
48. Mantyla, M. Boolean Operations of 2-manifold Through Vertex Neighborhood Classification. *ACM Trans. Graph.* 1986, **5**(1)pp 1–29.
49. Rossignac, J., Szymczak, A. Wrap&Zip Decompression of the Connectivity of Triangle Meshes Compressed with Edgebreaker. *J. Computat. Geom. Theory Applicat*. 1999, **14**,pp 119–135.
50. Hable, J. Rossignac, J. CST: Constructive Solid Trimming for Rendering BReps and CSG. *IEEE Trans. on Vis. Comput. Graph.* 13(5), Sept/Oct 2007. Available from the GVU Center at Georgia Tech. www.gvu.gatech.edu/research/techreports.html. as GVU Report GIT-GVU-06-16.
51. Brunet, P., Navazo, I. Solid Representation and Operation Using Extended Octrees. *ACM Trans. Graphics (TOG)*. 1990, **9**(2),pp 170–197.
52. Samet, H. *Applications of Spatial Data Structures*. Addison-Wesley: Reading, MA, 1990.
53. Ellis, J. L., Kedem, G., Lyerly, T. C., Thielman, D. G., Marisa, R. J., Menon, J. P., Voelcker, H. B. The RayCasting Engine and Ray Representations. *ACM Symposium on Solid Modeling Foundations and CAD / CAM Applications*. 1991, pp 255–267.
54. Andujar, C., Brunet, P., Chica, A., Rossignac, J., Navazo, I., Vinacua, A. *Computing Maximal Tiles and Applications to Impostor-Based Simplification*, Eurographics,September 2004.
55. Barequet, G., Goodrich, M., Levi-Steiner, A., Steiner, D. Contour Interpolation by Straight Skeletons. *Graphical Models*, 2004, **66**(4),pp 245–260.
56. Nonato, L. G., Cuadros-Vargas, A. J., Minghim, R., De Oliveira, M. F. Beta-Connection: Generating a Family of Models from Planar Cross Sections. *ACM Trans. Graph*. 2005, **4**,pp 1239–1258.
57. Amenta, N., Choi, S., Kolluri, R. K. The Power Crust, *Proc. of the 6th ACM Symposium on Solid Modeling and Applications*,May 2001, pp 249–266.
58. Dey, T., Goswami, S. Tight Cocone: A Water-tight Surface Reconstructor. *Proc. 8th ACM Sympos: In Solid Modeling Applications*. 2003, pp 127–134.Journal version in *J. Computing Infor. Sci. Eng*. 2003, **30**,pp 302–307.

59. Alexa, M., Gross, M., Pauly, M., Pfister, H., Stamminger, M., Zwicker, M. Point-based Computer Graphics, *Proc. of the Conference on SIGGRAPH 2004 course notes*; 2004.
60. Museth, K., Breen, D., Whitaker, R., Barr, A. Level Set Surface Editing Operators. *Proc. ACM SIGGRAPH*. ACM Press: New York, 2002, pp 330–338.
61. Pasko, G., Pasko, A., Kunii, T. Bounded Blending for Function-Based Shape Modeling. *IEEE Comput. Graph*. 2005, **2**,pp 36–45.
62. Farin, G. *Curves and Surfaces for Computer-Aided Geometric Design*, 2nd ed., Computer Science and Scientific Computing series, Academic Press: New York, 1990.
63. Kumar, S. Preventing Cracks in Surface Triangulations. *Proc. Chimera 98: 4th Symposium on Overset Composite Grid & Solution Technology*; 1998, pp 40–47.
64. Schmitt, B., Pasko, G., Pasko, A., Kunii, T. Rendering Trimmed Implicit Surfaces and Curves. *Proc. of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, Stellenbosch, South Africa, 2004.
65. Naylor, B., Amanatides, J., Thibault, W. Merging BSP Trees Yields Polyhedral Set Operations,*ACM Comput. Graph. SIGGRAPH '90*. 1990, **24**pp 115–124.
66. Brown, C. PADL-2: A Technical Summary. *IEEE Computer Graphics Applications*. 1982, **2**(2)pp 69–84.
67. Hayes, E., Sevy, J., Regli, W. Representation of Temporal Change in Solid Models; *Proc. of the 6th ACM Symposium on Solid Modeling and Applications*.D. C. Anderson Ed., SMA '01. ACM Press: New York, 2001, pp 317–318.
68. Rossignac, J., Borrel, P., Nackman, L. Interactive Design with Sequences of Parameterized Transformations. *Proc. 2nd Eurographics Workshop on Intelligent CAD Systems: Implementation Issues*. Veldhoven, The Netherlands, 1988, pp 95–127.
69. Raghothama, S., Shapiro, V. Boundary Representation Deformation in Parametric Solid Modeling. *ACM Trans. Graph*. 1998, **17**,pp 259–286.
70. Rossignac, J., Requicha, A. Constant-Radius Blending in Solid Modeling. *ASME Comput. Mech. Eng. (CIME)*. 1984, **3**,pp 65–73.
71. Rossignac, J. Requicha, A. Offsetting Operations in Solid Modelling. *Comput.-Aid. Geomet. Design*. 1986, **3**,pp 129–148.
72. McMains, S. Layered Manufacturing Technologies. *Commun. ACM*, 2005, **48**,pp 50–56.
73. Rossignac, J., Kim, J., Song, S., Suh, K., Joung, C.Boundary of the Volume Swept by a Free-Form Solid in Screw Motion. GVU Report GIT-GVU-06-19, 2006.
74. Rossignac, J., Pekkan, K., Whited, B., Kanter, K., Sharma, S., Yoganathan, A.,Surgem: Next Generation CAD Tools for Interactive Patient-Specific Surgical Planning and Hemodynamic Analysis. GVU Report GIT-GVU-06-15.
75. Middleditch, A. E., Sears, K. H. Blend Surfaces for Set Theoretic Volume Modelling Systems. *Proc of the 12th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH '85*. ACM Press: New York, 1985, pp 161–170.
76. A. Ralston, E. Reilly, Eds. *Encyclopedia of Computer Science and Engineering*, 2nd ed., van Nostrand Reinhold Co.: New York, 1983, pp 97–102.
77. Hoffmann, C. *Geometric and Solid Modeling*, Morgan Kaufmann: San Mateo, CA, 1989.
78. Agrawal, A., Requicha, A. A. G. A Paradigm for the Robust Design of Algorithms for Geometric Modeling; *Proc. Eurographics '94. Computer Graphics Forum*, 1994, 13(3),pp 33–44.
79. Banerjee, R., Rossignac, J. Topologically Exact Evaluation of Polyhedra Defined in CSG with Loose Primitives. *Computers Graphics Forum*. 1996, **15**(4),pp 205–217.
80. Hable, J. Rossignac, J. Blister: GPU-based Rendering of Boolean Combinations of Free-form Triangulated Shapes. *ACM Trans. Graphics*. 2005, **24**(3),pp 1024–1031.
81. Rossignac, J., Megahed, A., Schneider, B. O. Interactive Inspection of Solids: Cross-Sections and Interferences. *Proc. ACM Siggraph, ACM Comput. Graph*. 1992, **26**,pp 353–360.
82. Hadap, S., Eberle, D., Volino, P., Lin, M., Redon, S., Ericson, C. Collision Detection and Proximity Queries. *ACM SIGGRAPH 2004 Course Notes, SIGGRAPH '04*. ACM Press: New York, 2004.
83. Ronfard, R. Rossignac, J. Full-Range Approximations of Triangulated Polyhedra. *Comput. Graph. Forum Proc. of Eurograph*. 1996, pp C-67.
84. Rockwood, A., Heaton, K., Davis, T. Real-Time Rendering of Trimmed Surfaces. *Proc. ACM SIGGRAPH*, 1989, pp 107–117.
85. Kumar, S., Manocha, D. Efficient Rendering of Trimmed NURBS Surfaces. *Computer-Aided Design*. 1995, **27**(7),pp 509–521.
86. Schneider, B. O., Borrel, P., Menon, J., Mittleman, J., Rossignac, J. BRUSH as a Walkthrough System for Architectural Models. In *Rendering Techniques*, Eurographics Workshop on Rendering, Springer-Verlag: New York, 1995, pp 389–399.
87. Llamas, I., Kim, B., Gargus, J., Rossignac, J., Shaw, C. D. Twister: A Space-warp Operator for the Two-handed Editing of 3D shapes. *ACM Trans. Graphics*. 2003, **22**(3),pp 663–668.
88. Murali, T. M., Funkhouser, T. A. Consistent Solid and Boundary Representations from Arbitrary Polygonal Data. *Proc. 1997 Symposium on Interactive 3D Graphics*. ACM Press: Providence, RI, April 1997, pp 155–162.
89. Bischoff, S., Pavic, D., Kobbelt, L. Automatic Restoration of Polygon Models. *ACM Trans. Graph*. 2005, **24**(4),pp 1332–1352.
90. Lopes, H., Tavares, G. Structural Operators for Modeling 3-manifolds; *Proc. ACM Symposium on Solid Modeling and Applications SMA*: ACM Press, 1997, pp 10–18.
91. Baumgart, B.Winged Edge Polyhedron Representation, AIM-79. Stanford University Report STAN-CS-320, 1972.
92. Rossignac, J. Through the Cracks of the Solid Modeling Milestone. In *From Object Modelling to Advanced Visualization*.Coquillart, S., Strasser, W., Stucki, P., Eds., Springer Verlag: New York, 1994, pp 1–75.
93. Kallmann, M., Thalmann, D. Star-vertices: A Compact Representation for Planar Meshes with Adjacency Information. *J. Graphics Tools*. 2001, **6**(1),pp 7–18.
94. Rossignac, J., Safonova, A., Szymczak, A. Edgebreaker on a Corner Table: A Simple Technique for Representing and Compressing Triangulated Surfaces. *Hierarchical Geometrical Methods Scientific Vis*. 2003, pp 41–50.
95. Taubin, G., Rossignac, J.Geometric Compression through Topological Surgery, IBM Research Report RC-20340, January 1996. http://www.watson.ibm.com:8080/PS/7990.ps.gz.

96. Isenburg, M., Snoeyink, J. Spirale Reversi: Reverse Decoding of the Edgebreaker Encoding; *Canadian Conference on Computational Geometry 2000*:August 2000, pp 247–256.
97. Lewiner, T., Lopes, H., Rossignac, J., Wilson-Vieira1, A.Efficient Edgebreaker for Surfaces of Arbitrary Topology; SIBGRAPI/SIACG 2004.
98. Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., Hubner, R. Levels of Detail for 3D Graphics Morgan Kaufmann: San Mateo, CA, 2002.
99. Garland, M. Heckbert, P. Surface Simplification Using Quadric Error Metrics. *Proc. ACM SIGGRAPH'97*. 1997, pp 209–216.
100. Warren, J., Weimer, H. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann: San Francisco, CA, 2001.
101. Botsch, M., Pauly, M., Rössl, C., Bischoll, S., Kobbelt, L. Geometric Modeling Based on Triangle Meshes. *Course Notes, ACM SIGGRAPH 2006*. ACM Press: New York, 2006.
102. (102) Attene, M., Falcidieno, B., Spagnuolo, M., Rossignac, J. SwingWrapper: Retiling Triangle Meshes for Better Compression. *ACM Trans. Graphics* 2003, **22**(4),pp 982–996.
103. Attene, M., Falcidino, B., Spagnuolo, M., Rossignac, J. Sharpen&Bend: Recovering Curved Edges in Triangle Meshes Produced by Feature-insensitive Sampling. *IEEE Trans. Visualization Computer Graphics (TVCG)*, 2005, **11**(3),pp 181–192.
104. Edelsbrunner, H., Mücke, E. P. Simulation of Simplicity: A technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Trans. Graph*. 1990, **9**(1),pp 66–104.
105. Ronfard, R., Rossignac, J. Triangulating multiply-connected polygons: A simple, yet efficient algorithm. *Computer Graphics Forum, Proc. Eurographics*,Vol 13, No 3, pp C281–C292,Sept 1994.
106. Rossignac, J., Cardoze, D. Matchmaker: Manifold BReps for Non-manifold r-sets *Proc. of the ACM Symposium on Solid Modeling*. 1999, pp 31–41.
107. Rossignac, J. Structured Topological Complexes: A Feature-Based API For Non-Manifold Topologies. *Proc. of the ACM Symposium on Solid Modeing 97*.Hoffmann, C., Bronsvort, W., Eds., ACM Press: New York, 1997, pp 1–9.
108. Lodha, S., Franke, R. Scattered Data Techniques for Surfaces. *Proc. of the Conference on Scientific Visualization*.June 9–13, 1997, p 181.
109. Bajaj, C., Chen, J., Xu, G. Modeling with Cubic A-patches. *ACM Trans. Graphics (TOG)*. 1995, **14**(2),pp 103–133.
110. Reuter, P., Tobor, I., Schlick, C., Dedieu, S. Point-Based Modelling and Rendering using Radial Basis Functions. *Proc. of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, 2003.
111. Schmitt, B., Pasko, A., Christophe, S. Constructive Modeling of FRep Solids Using Spline Volumes *Proc. of the Sixth ACM Symposium on Solid Modeling and Applications*. 2001, pp 321–322.
112. Fleishman, S., Cohen-Or, D., Silva, C. Robust Moving Least-squares Fitting with Sharp Features. *ACM Trans. Graphics (TOG)*. 2005, **24**(3).
113. Adzhiev, V., Kartasheva, E., Kunii, T., Pasko, A., Schmitt, B. Cellular-functional modeling of Heterogeneous Objects; *Proc. of the 7th ACM Symposium on Solid Modeling and Application*; Saarbrücken, Germany, June 17–21, 2002.
114. Bowyer, A., Cameron, S., Jared, G., Martin, R., Middleditch, A., Sabin, M., Woodwark, J. *Introducing Djinn: A Geometric Interface for Solid Modeling*, Information Geometers Ltd.: 1995.
115. Tilove, R. A Null-Object Detection Algorithm for Constructive Solid Geometry. *Commun. ACM*. 1984, **27**,pp 684–694.
116. Goldfeather, J., Molnar, S., Turk, G., Fuchs, H. Near Real-time CSG Rendering Using Tree Normalization and Geometric Pruning. *IEEE Comput.r Graph. Applicat*. 1989, **9**(3),pp 20–28.
117. Rossignac, J. Processing Disjunctive Forms Directly from CSG Graphs. *Proc. of CSG 94: Set-theoretic Solid Modelling Techniques and Applications, Information Geometers*. Winchester, UK, 1994, pp 55–70.
118. Rossignac, J.Blist: A Boolean List Formulation of CSG Trees. GVU Report GIT-GVU-99-04, 1998.
119. Requicha, A. A. G., Voelcker, H. B. Boolean Operations in Solid Modelling: Boundary Evaluation and Merging Algorithms. *Proc. IEEE*. 1985, **73**,pp 30–44.
120. Rossignac, J. CSG Formulations for Identifying and for Trimming Faces of CSG Models. *CSG'96: Set-Theoretic Solid Modeling Techniques and Applications, Information Geometers*, Woodwark, J., Ed., 1996, pp 1–14.
121. Rossignac, J.BLIST: A Boolean List Formulation of CSG Trees. Technical Report GIT-GVU-99-04, GVU Center, Georgia Institute of Technology. 1999. http://www.cc.gatech.edu/gvu/reports/1999.
122. Bryant, R. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. *Proc. IEEE/ACM international Conference on Computer-Aided Design*. 1995, pp 236–243.
123. Yang, B., O'Hallaron, D. Parallel Breadth-First BDD Construction. *Proc. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1997, pp 145–156.
124. Payne, H., Meisel, W. An Algorithm for Constructing Optimal Binary Decision Trees. *IEEE Trans. Comput*. 1977, **26**,pp 905–916.
125. Rossignac, J. Optimized Blist Form (OBF). Technical Report GIT-GVU-07-10, GUV Center, Georgia Institute of Technology. May 2007.
126. Shapiro, V., Vossler, D. Construction and Optimization of CSG Representations. *Comput.-Aid. Design*. 1991, **23**pp 4–20.
127. McMains, S., Séquin, C. A Coherent Sweep Plane Slicer for Layered Manufacturing; *Proc. of the fifth ACM Symposium on solid Modeling and Applications*;June 8–11, 1999, pp 285–295.
128. Rossignac, J., Requicha, A. Constructive Non-Regularized Geometry. *Comput.-Aided Design*. 1991, **23**,pp 21–32.
129. Rossignac, J. Constraints in Constructive Solid Geometry. *Proc. ACM Workshop on Interactive 3D Graphics* ACM Press: Chapel Hill, NC, 1986, pp 93–110.
130. Durand, C., Hoffmann, C. A Systematic Framework for Solving Geometric Constraints Analytically. *J. Symb. Comput*. 2000, **30**(5),pp 493–519.
131. Rossignac, J. Issues on Feature-Based Editing and Interrogation of Solid Models. *Comput. Graph*. 1990, **14**,pp 149–172.
132. Raghothama, S., Shapiro, V. Boundary Representation Deformation in Parametric Solid Modeling. *ACM Trans. Graph*. 1998, **17**,pp 259–286.
133. van der Meiden, H., Bronsvoort, W. Solving Topological Constraints for Declarative Families of Objects; *Proc. ACM Symposium on Solid and Physical Modeling*;June 6–8, 2006.

134. Barr, A. Local and Global Deformations of Solid Primitives; *Proc. Siggraph'84, Computer Graphics*, 1984, **18**(3),pp 21–30.

135. Rossignac, J., Kaul, A. AGRELs and BIPs: Metamorphosis as a Bezier Curve in the Space of Polyhedra. *Comput. Graph. Forum*. 1994, **13**,pp C179–C184.

136. Requicha, A. A. G. Toward a Theory of Geometric Tolerancing. *Int. J. Robotics Res*. 1983, **2**,pp 45–60.

137. Chen, Y., Wang, H., Rosen, D., Rossignac, J. Filleting and Rounding Using a Point-based Method. *ASME Design Engineering Technical Conferences, DETC05/DAC-85408*.September 2005.

138. Williams, J., Rossignac, J. Mason: Morphological Simplification. *Graph. Models*. 2005, **67**,pp 285–303.

139. Williams, J., Rossignac, J. Tightening: Curvature-Limiting Morphological Simplification. *ACM Symposium on Solid and Physical Modeling (SPM)*, June 2005.

140. Sakkalis, T., Peters, T. Ambient Isotopic Approximations for Surface Reconstruction and Interval Solids *Proc. of the Eighth ACM Symposium on Solid Modeling and Applications*.June 16–20, 2003.

141. Taubin, G. A Signal Processing Approach to Fair Surface Design *Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*.September 1995, pp 351–358.

142. Ilies, H., Shapiro, V. UNSWEEP: Formulation and Computational Properties; *Proc. of the 4th ACM Symposium on Solid Modeling and Applications* 1997, pp 155–167.

143. Powell, A., Rossignac, J. ScrewBender: Smoothing Piecewise Helical Motions. *IEEE Comput. Graph. Applicat*. In press.

144. Szymczak, A., Rossignac, J., King, D. Piecewise Regular Meshes: Construction and Compression. *Graph. Models*. 2002, **64**,pp 183–198.

145. Dong, S., Bremer, P-T., Garland, M., Pascucci, V., Hart, J. C. Spectral Surface Quadrangulation. *ACM Trans. Graphics Proc. SIGGRAPH*, 2006.

146. Ray, N., Li, W. C., Lévy, B., Sheffer, A., Alliez, P. Periodic Global Parameterization. *ACM Trans. Graph*. 2006, **4**,pp 1460–1485.

Reading List

Rossignac, J., Requicha, A. Depth Buffering Display Techniques for Constructive Solid Geometry *IEEE Comput. Graph. Applicat*. 1986, **6**,pp 29–39.

Sederberg, T. W., Meyers, R. J. Loop Detection in Surface Patch Intersections. *Comput.-Aid. Geomet. Design*. 1988, **5**,pp 161–171.

JAREK R. ROSSIGNAC
Georgia Institute of Technology