Solid Modeling

Jarek R. Rossignac
GVU Center, College of Computing,
Georgia Institute of Technology, Atlanta

Aristides A. G. Requicha
Computer Science Department
University of Southern California at Los Angeles

1 Introduction

A solid model is a digital representation of the geometry of an existing or envisioned physical object. Solid models are used in many industries, from entertainment to health care. They play a major role in the discrete-part manufacturing industries, where precise models of parts and assemblies are created using solid modeling software or more general computer-aided design (CAD) systems. The design process is usually incremental. Designers may specify points, curves, and surfaces, and stitch them together to define electronic representations of the boundary of the object. Alternatively, they may select models of simple shapes, such as blocks or cylinders, specify their dimensions, position, and orientation, and combine them using union, intersection, or difference operators. The resulting representation is an *unambiguous,* complete, and detailed digital approximation of the geometry of an object or of an assembly of objects (such as a car engine or an entire airplane). Interactive three-dimensional (3D) graphic supports the design activities by providing designers with: (1) easy to understand images of their design, (2) efficient facilities for graphically selecting or editing features of the part being designed, and (3) immediate feedback, which helps them perform and check each design step.

Early applications of solid modeling focused on producing correct engineering drawings automatically and on cutter-path generation for numerically controlled machining [Requicha82, Requicha96]. Today, the engineering drawings still provide a link to traditional, non-electronic manufacturing or archival processes, but are being rapidly replaced with electronic file transfers. Solid modeling has evolved to provide the 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 is to relieve engineers from all of the low-level or non-creative tasks in designing products; in assessing their manufacturability, assemblability, and other life-cycle characteristics; and in generating all the information necessary to produce them. Engineers should be able to focus on conceptual, high-level design decisions, while domain-expert application programs should provide advice on the consequences 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 (see for example [Spyridi93]), although in principle made possible by informationally complete solid models, remains a research challenge in spite 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 [Requicha88, Requicha92]. 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 efficient data structures and algorithms, and benefit from recent developments in the field of *computational geometry.* Efficient processing is essential, because the complexity of industrial models is growing faster than the performance of commercial workstations. Techniques for modeling and analyzing surfaces and for computing their intersections are important in solid modeling. This area of research, sometimes called *computer aided geometric design,* has strong ties with numerical analysis and differential geometry. *Graphic user-interface* (GUI) techniques also play a crucial role in solid modeling, since they determine the overall usability of the modeler and impact the user's productivity. There have always been strong symbiotic links and overlaps between the solid modeling community and the *computer graphics* community. Solid modeling interfaces are based on efficient three-dimensional (3D) graphics techniques, whereas research in 3D graphics focuses on fast or photo-realistic rendering of complex scenes, often composed of solid models, and on realistic or artistic animations of non-rigid objects. A similar symbiotic relation with *computer vision* is regaining popularity, as many research efforts in vision are model-based and attempt to extract 3D 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.*

Research in solid modeling emerged in the 1970's 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 (currently encapsulated in the CAD/CAM/CAE abbreviation).

Solid modeling impacts a great variety of design and manufacturing activities. Examples include early sketches, design decisions, space allocation negotiations, detailed design, drafting, interactive visualization of assemblies, maintenance-process simulation, usability studies, engineering changes, reusability of design components, analysis of tolerances [Requicha93], 3D mark-up and product data management, remote collaboration, internet-based catalogs of parts, electronic interaction with suppliers, analysis (e.g., mechanism analysis or finite elements), process planning and cutter-path generation for machining, assembly and inspection planning, product documentation, and marketing.

2 Solid modeling systems

A solid modeling system, often called a *solid modeler*, is a computer program that provides facilities for storing and manipulating data structures that represent the geometry of individual objects or assemblies. These representations can be created either by a human through a graphic user interface (GUI), or specified by software applications via an application programming interface (API). In a typical interactive application, a user selects and manipulates modeling primitives (parameterized instances of simple geometric shapes, such as a cylinder or a line) and invokes modeling operations that combine or transform these primitives into more elaborate representations.

A modeler's GUI generates 3D graphic feedback to a user by immediately displaying selected portions of the objects being designed. In addition, it provides facilities for selecting and for graphically editing the displayed entities. Users of modern modelers can describe objects in terms of *features*, which are higher-level entities meaningful for their applications, can use dimensions and other *constraints* to help in sizing and positioning geometric entities, and can also *parameterize* the objects so as to create object families.

The choice of representations used by the modeler determines its domain (i.e., which objects can be modeled), and has a strong impact on the complexity and performance of the algorithms that create or process the representations. A modeler may support several distinct representation schemes. Consistency between representations in different schemes is typically enforced by representation conversion algorithms.

Application programs invoked by the users analyze the models and generate information on the object's properties (e.g., its moments of inertia, or its deflection under load, calculated by finite element methods), or on processes needed by life-cycle activities such as manufacture, assembly or inspection. In a modern engineering environment, these applications should run concurrently with the design process, to help assess the consequences of design decisions, and provide guidance to the designers. The architecture of a typical solid modeler is illustrated Figure 1.

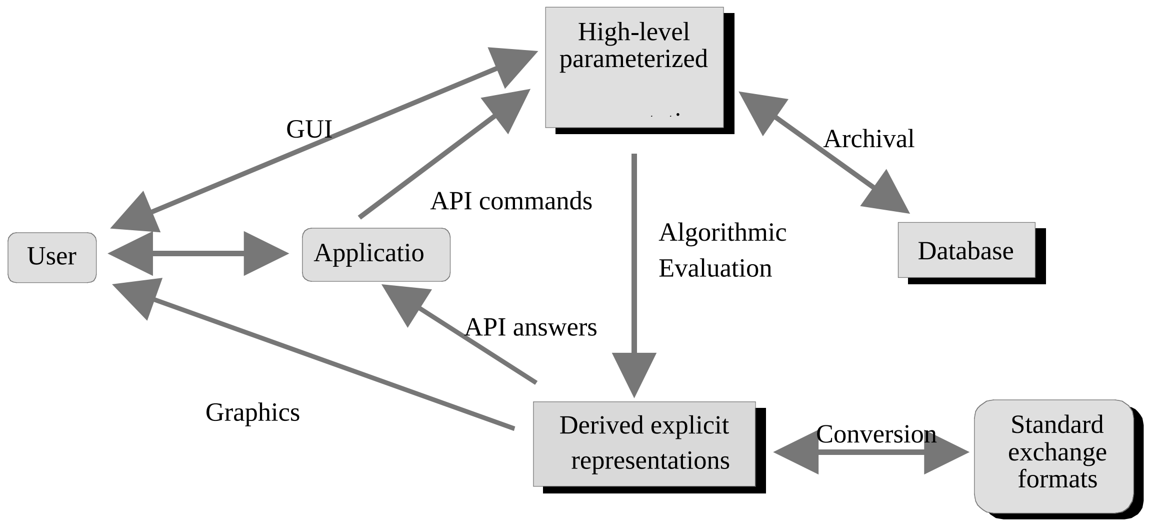

Figure 1: A high-level parameterized representation of the design steps and of the model's features and constraints may be created interactively by the user through a design interface (GUI) or by an application through a programming interface (API). This constructive representation may be archived for future editing and reuse and may be automatically converted into one or several derived representations, which explicitly list the faces of the solid's boundary or the 3D cells covered by the solid. The derived representations are suitable for providing realtime graphic feedback to the designer and directly support application queries. They may also be exported to other applications or CAD systems by converting them to a standard file format.

3 Mathematical foundations

The modeling process is the result of a sequence of abstractions and approximations (idealization, surface approximation, and digitization) presented in this section.

3.1 Idealization

First the physical shape is abstracted into a perfect and homogeneous 3D point set, ignoring internal structures and boundary imperfections. (Modeling of non-homogeneous objects will be discussed briefly later in this article.)

Mathematically, a solid is a subset of 3D Euclidean space. Precise definitions were developed in the mid 1970's at the University of Rochester's Production Automation Project (PAP) and have guided the development of data structures and algorithms that support high-level design operations and guarantee that valid representations of solids are produced. *R-sets*, which are *bounded*, *regular*, *semi-algebraic* sets, were proposed as mathematical models for physical solids [Requicha80]. A set is bounded if it has finite extent, i.e., if it can be enclosed by a ball of a finite radius. A regular set is homogeneously three-dimensional, and therefore has no dangling faces or edges. Mathematically, a set is regular if it equals the topological closure of its interior. A semi-algebraic set is the result of combining through set operations (union, difference, intersection and complement) a finite number of half-spaces, each defined by an algebraic inequality of the form $\{(x,y,z) \mid f(x,y,z) \leq 0\}$, where f is a polynomial. Note that this definition does not preclude sets that are bounded by free-form parametric patches, although implicitizing (i.e., converting them) into semi-algebraic form is computationally hard and produces inequalities of very high degree. (Semi-algebraic sets also play an important role in robotics, constraint satisfaction, and theorem proving.) More precise definitions for these topological terms may be found in [Alexandroff61].

Some modelers cater only to r-sets that are *manifolds*. A manifold model has a boundary whose edges are shared by precisely two faces, and whose vertices are adjacent to a set of faces that forms a single cone with apex at the vertex. Unfortunately, the domain of manifold r-sets is not closed under some of the fundamental modeling operations. For example, the union of the two L-shaped manifold solids in Figure 2 is a non-manifold solid. Hence, users of such restricted systems must carefully avoid creating non-manifold shapes.

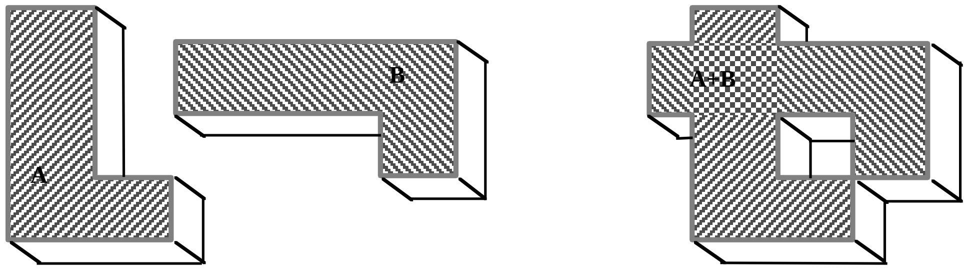

Figure 2: The two L-shaped solids A and B (left), are positioned so that an edge of A coincides with an edge of B. Their union, A+B, is a non-manifold solid (right).

3.2 Surface Approximation

In a second idealization stage, the boundary of the shape is approximated by a relatively small number of faces, each face being a subset of a surface of a type supported by the modeling system. Much of the variation between solid modeling technologies stems from the different choices of the primitive surfaces they support. These surfaces essentially determine the geometric domain of a system, i.e., the objects that can be modeled exactly.

On one hand, planar face primitives, i.e., triangles or more general polygons, provide a poor approximation of the real shape, unless used in large quantities to define fine tessellations of highly curved geometries. Although algorithms for dealing with individual triangles are simple and relatively robust, compact data structures for representing very large numbers of triangles and efficient algorithms for processing or rendering them are complex to develop and to maintain.

On the other hand, a single parametric free-form surface patch [Farin90] which may be smoothly stitched with other patches in a Non-Uniform Rational B-spline Surface (NURBS), may provide a better fit to the desired geometry than a thousand triangles. However, detecting and computing intersections of such free-form surfaces involves elaborate mathematical techniques and algorithms that are significantly slower and less reliable than their counterparts for triangular geometries.

Natural quadric surfaces (plane, cylinder, cone, sphere) offer an attractive compromise, 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 3 surfaces intersect. However, these surfaces cannot model precisely the numerous fillets and blends found in most manufactured parts [Rossignac84]. 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.

The choice of the geometric domain of a modeler may affect the accuracy of the analysis results. For example, a cylindrical pin may freely rotate in a cylindrical hole of a slightly larger radius, if both surfaces are modeled using natural quadrics. Using faceted approximations for the pin and the hole may lead to the wrong conclusion that the pin cannot rotate or doesn't even fit.

3.3 Digitization

In a third approximation stage, the numeric parameters that define the precise shape and position of the surfaces and their intersections are rounded to the nearest value representable in the digital format selected by the developer of the system. The most common formats are floating point and integer or rational [Ralston83]. Floating point representations cover a wider range of values, but their worst case round-off error grows with the distance to the origin. Integer numbers, when scaled and offset properly by a judicious choice of units and of the origin, provide a much denser and uniform coverage of a desired modeling range, and hence lead to lower and better-controlled round-off errors. In practice, floating point numbers are favored because they do not require prior knowledge of the range and can be used in a uniform way to represent the parameters of the model and the results of intermediate calculations. However, floating point calculations 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. Exact arithmetic packages do not suffer from round-off problems, but are significantly slower and usually only effective for polyhedral geometries (see discussions in [Banerjee96, Agrawal94]).

4 Representations

This section reviews the most common schemes for representing 3D objects: boundary representations, constructive solid geometry, and spatial decompositions. Their extensions beyond the domain of solid models are also briefly discussed.

4.1 Boundary representations

A physical object, modeled mathematically by an r-set, is unambiguously defined by its boundary. Therefore, a solid may be represented by a set of non-overlapping faces, whose union approximates the solid's boundary. Such a scheme is called a *boundary representation,* or BRep for short. Unfortunately, an arbitrary set of faces does not necessarily correspond to the boundary of a solid. In fact, invalid BReps created by designers or by incorrect algorithms that implemented higher-level modeling operations plagued the early versions of many solid modelers.

4.2 Data structures

Although a simple enumeration of a solid's faces suffices to unambiguously define the solid, most boundary representation schemes store 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 often capture the incidence relations between a face and its bounding edges and vertices, and between an edge and its bounding vertices. Many data structures have been studied to achieve desired compromises between (1) the 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.

As an example, we describe with the help of Figure 3 a simple data structure for manifold polyhedral objects. It is a simplified version of the winged-edge representation introduced in [Baumgart72]. The vertices of the object are stored in a table with the associated coordinates. Edges are represented by references to vertices and to adjacent edges. Note that this data structure defines the geometry of the edges and of the loops which bound the faces, but does not explicitly capture the relations between the different loops of the same face. For example, the face on the right of Figure 3 has an outer loop that encloses a smaller, triangular, inner loop of edges. Some boundary representations have separate nodes for faces, and store explicitly the face and loop relationships. Alternatively, faces with holes may be converted into simply-connected faces by introducing artificial "bridge" edges that each merge two loops of a face.

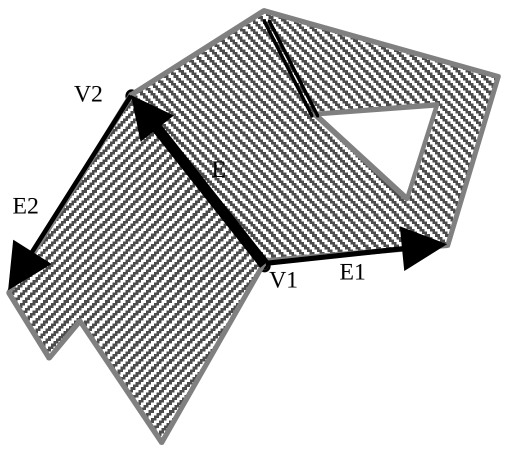

Figure 3: Edge E points to its starting and ending vertices V1 and V2. The order of vertex-references defines an orientation for the edge (arrow). The node that corresponds to E also contains references to edges E1 and E2. E1 is the next edge around V1 for the face on the right of E, and E2 is the next edge around V2 for the face on the left of E. The terms "next edge", "on the left", and "on the right" are unambiguously defined by considering the outward-pointing normals to the faces and the orientation of E. The bridge edge which links the outer loop of the right face with its inner loop is indicated by the double line segment through the face.

The faces in a BRep of a curved object may be represented as parametric patches, which are the images of a unit square or of a triangle by a polynomial mapping from a 2D parameter space into the 3D modeling space. Alternatively, a face may be represented as a *trimmed surface* by a reference to the supporting surface on which it lies, and by its boundary in that surface. The supporting surface may itself be a larger parametric patch or an implicit surface. The boundary is usually represented by a set of edges, which may be arranged into loops.

The edges of a solid typically lie on the intersection curves between two surfaces, and sometimes on singular curves 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 3D, or in the 2D parameter space of the host surface. The latter case leads to redundant representations because each edge typically belongs to two surfaces. Redundant representations may conflict due to numeric round-off errors, and cause "cracks" in the boundary. 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 [Brown82]. 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.

Edge 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.

4.3 Domain extensions for BReps

Many contemporary applications of solid modeling require dealing with non-regular 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 material properties). Such objects cannot be represented in a traditional solid modeler. Several boundary representation schemes have been proposed for domains that extend beyond solids [Rossignac92b]. For example, Weiler's radial-edge structure was centered around entities that represent face-edge and edge-vertex incidence relations to explicitly capture how faces are ordered around an edge [Weiler87]. Schemes that extend beyond solids 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 [Rossignac89b], provides a general representation for non-regular point sets, which combine isolated points, edges, faces, and volumes with internal structures and cracks (isolated faces, edges, or vertices removed from the interior of the volume). An SGC model is based on a subdivision of Euclidean space into cells of various dimensions that are disjoint, open, connected sub-manifolds and are "compatible" with all the solids, faces, edges, and vertices of the primitive shapes and with all their intersections. (Two sets are compatible if they are disjoint or equal, or if one is included in the other.) Each cell is represented by its supporting manifold (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. These combine mutually disjoint regions, where each region is the union of all the cells with identical markings. Each region may correspond to a mixed-dimensional (i.e. non regularized) set.

4.4 Constructive solid geometry

Constructive representations capture a construction process which defines the solid by a sequence of operations that instantiate or 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.

4.4.1 CSG representations

Constructive Solid Geometry (CSG) is the most popular constructive representation. Its primitives are parameterized solids, which may be simple shapes (such as cylinders, cones, blocks) or more complex features suitable for a particular application domain (such as slots or counter-bored holes). 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) or scaling.

The transformed instances may be combined through regularized Boolean operations: union, intersection, and difference. These regularized operations perform the corresponding set theoretic Boolean operations, and then transform the result into an r-set by applying the topological interior operation followed by the topological closure. They always return valid (although possibly empty) solids. Although other Boolean operations may be offered, these three are convenient and sufficient, because amongst the 16 different Boolean combinations of two sets, A and B, 8 are unbounded, 3 are trivial, and only 5 are useful for solid modeling: the union A+B, the intersection AB, the differences A-B and B-A, and the symmetric difference, (A-B)+(B-A), as shown in Figure 4.

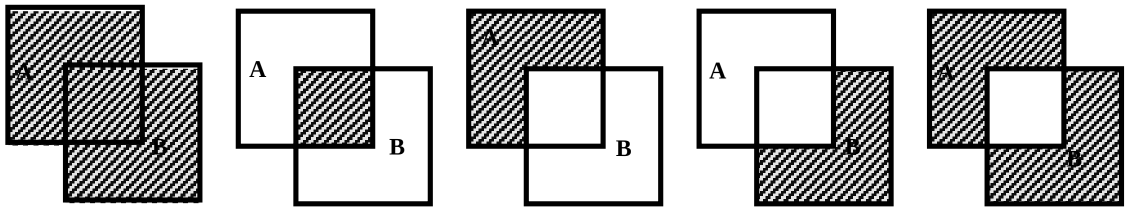

Figure 4: The five non-trivial Boolean combinations of two sets (from left to right): A+B={a: a∈A or a∈B}, AB={a: a∈A and a∈B}, A-B={a: a∈A and a∉B}, B-A={a: a∉A and a∈B}, and the symmetric difference (A-B)+(B-A). The notation "ikS" refers to the interior of the closure of S and corresponds to the regularization of the results of the standard set theoretic Boolean operations.

Figure 5 illustrates how a simple syntax may be used to specify a solid in CSG. Parsing such a syntax, yields a rooted graph, whose leaves represent primitive instances and whose internal nodes represent transformations or Boolean operations that produce solids. The root represents the solid corresponding to the CSG graph.

CSG representations are concise, always valid in the r-set modeling domain, 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.

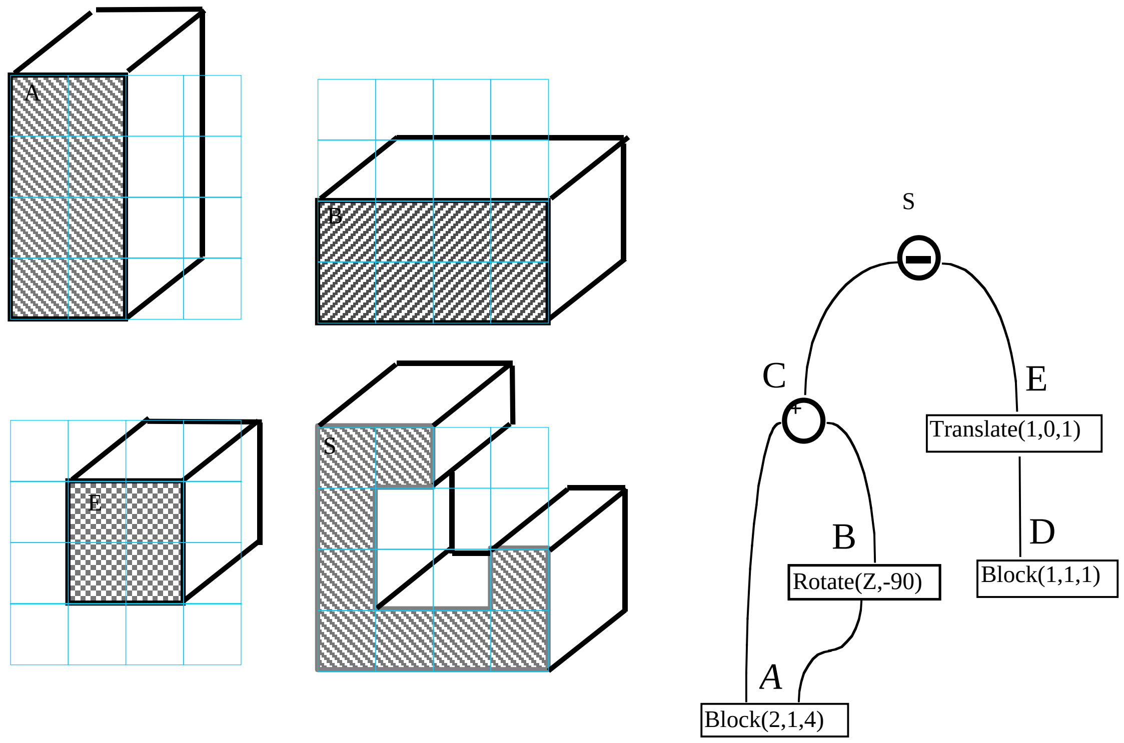

Figure 5: The instances A, B, and E are shown (left) superimposed on the same reference grid. The solid S was specified by the following sequence of commands:
A=Block(2,1,4);
B=Rotated(A,Z-axis,-90)
C=A+B
D=Block(1,1,1);
E=Translated(D,1,0,1);
S=C-E;
The corresponding CSG graph (right) has 2 leaf primitives, 2 transformation nodes, and 2 regularized Boolean operation nodes.

4.4.2 Bending, twisting, blending, and Minkowski operations

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

Simple non-linear transformations may *twist* an object by applying to each point in space a 2D rotation around the z-axis with an angle that is a function of the z-coordinate [Barr84]. 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 in [Sedeberg86].

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⊕tB to construct parameterized shapes that smoothly interpolate any two polyhedra A and B. They further expanded this approach to the weighted Minkowski averages of a larger number of polyhedra [Rossignac94]. 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 which 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.

Minkowski sums also play a crucial role in robotics for collision avoidance [Latombe91], and in accessibility and visibility analysis [Spyridi90, Spyridi94] Requicha used Minkowski operations to define the mathematical meaning of tolerance specifications [Requicha83] Minkowski sums or differences with a ball define growing and shrinking operations on solids. For instance, when B is a ball of radius *r*, A⊕B is a solid defined as the union of A with all points at a distance less or equal to r from A. The Minkowki difference, is the regularized difference between A and the set of points at a distance less than or equal to r from the complement of A. Combinations of growing and shrinking operations on solids were used by Rossignac and Requicha [Rossignac86] to produce constant radius fillets and blends (see Figure 6 for a 2D illustration). 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.

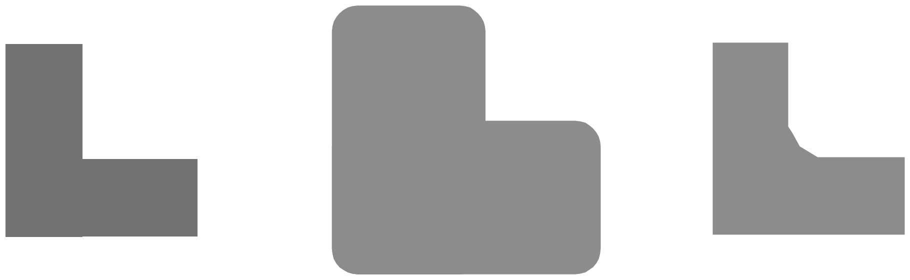

Figure 6: The 2D shape (left) may be filleted (right) by first expanding it via a Minkowski sum with a disk of radius r (center) and then by shrinking the result via a Minkowski difference with the same disk. Subtracting the original shape from the filleted version yields solid fillets that may be easily trimmed before being added to the original solid.

4.4.3 CSG extensions to non-regularized sets

Extensions of the Boolean operations to non-regularized solids, to r-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 Non-Regularized Geometry (CNRG) model [Rossignac91]. Users or 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 non-regularized Boolean and topological operations on primitive regions. Rossignac's Structured Topological Complexes (STC) add to the CNRG representation the capability of creating and interrogating simultaneously several decompositions of the three-dimensional space [Rossignac97]. For example, the same space 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.

4.5 Spatial decomposition approximations of solids

Solids may be represented either exactly or approximately by a variety of space decomposition schemes. The entire 3D space, or just the set that corresponds to the solid, is partitioned into non-overlapping 3D regions called *cells*.

Spatial decomposition schemes may differ in the restrictions they impose on cells. These may be polyhedral or bounded by curved surfaces. Cells are usually connected. Furthermore, many decomposition schemes do not support cells with holes or handles Cells may be further restricted to be convex, or even to be tetrahedra, or axis-aligned rectangular blocks, or regularly-spaced cubes.

4.5.1 Regular decompositions

Regular decompositions approximate solids by stacks of constant-thickness slices, by prismatic columns of square cross-sections and parallel axes, or by regularly-spaced arrangements of cubes, which may also be organized into a hierarchical structure, called an *octree* [Samet90]. Conceptually, an octree may be constructed from a non-hierarchical decomposition by replacing 2x2x2 arrangements of neighboring cubes that are all in the solid or all in its complement by a single cell representing their union. This process is repeated recursively so as to reduce the total number of cells needed to represent the solid. Although the error allowed by these discretized representations may be significant, especially when their resolution is limited by storage concerns, they are popular, because their simplicity is well-suited to parallel algorithms and hardware support.

4.5.2 *Boundary space partition trees*

Among irregularly-spaced arrangements of cells, the Boundary Space Partition tree (BSP), introduced at the University of North Carolina at Chapel Hill [Naylor90] is one of the most interesting. A BSP recursively splits space into a binary tree, whose leaves correspond to convex polyhedra. A selection of these leaf-cells defines the desired solid. BSP trees were designed originally to speed up hidden-surface elimination in graphics. Although the z-buffer hardware available today on most graphics systems [Rossignac86b] makes BSP trees obsolete for conventional hardware-assisted rendering, BSP trees are still important for low-end software graphics systems (such as video-games), as an auxiliary data structure for computing shadows and other properties, and as an efficient modeling tool for supporting certain fundamental queries on solids. A BSP representation of a polyhedral solid may be obtained by selecting a face of the solid and using its supporting plane to split the solid into two parts. The face is encoded as the root of the BSP tree and the process is repeated to construct its two children nodes, as the BSP trees for the two parts of the solid.

5 Algorithms and Applications

Algorithms for constructing representations of solids, for converting between them, or for processing these representations in various applications build upon a few fundamental procedures that implement geometric queries and constructions. In the following sub-sections we describe some of these fundamental procedures, and then discuss briefly their use in application algorithms and efficiency issues. Our goal is to provide a high-level overview of typical computations involved in solid modeling. We neither attempt to present the most efficient algorithms nor the (many) details required for their successful implementation.

5.1 Fundamental Queries and Operations

The most fundamental query is point membership, which tests whether a point is contained in a given set or not. Point membership classification may be generalized to sets of points such as curves or surface patches. This generalization, called set membership classification, is useful for many applications. Other important algorithms deal with Boolean operations, representation conversion, and intersections. Algorithms and representations are closely intertwined. We illustrate their interdependency by discussing a few algorithms that compute the same mathematical functions (or properties) but operate on different representations.

5.1.1 *Point membership classification for BReps*

The most popular technique for point membership classification (PMC) with respect to solids represented by their boundaries is ray-casting. Construct a half-line L starting from the candidate point X to infinity and then count the parity of the number of times L crosses a face of S. Even parity of the number of line/face intersections implies that X lies outside of the solid. Odd parity implies that X lies inside. (A point on a face is by definition on S.) The treatment of special cases when L hits an edge or a vertex or when it is tangent to one of the supporting surfaces may be avoided by randomly choosing candidate lines L until one is found that does not run into any singular situation.

For polyhedral models, each face F of S is checked against L as follows. First compute the intersection I of L with the plane P supporting F. (I is always an isolated point since L was chosen not to be tangent to any of the supporting planes.) Then perform the PMC of I against F in the two-dimensional space of plane P. If I lies in F, then the parity bit is toggled for the PMC of X against S, and the next face of S is considered. (The parity bit is initially set to 0 indicating that X is out of S if no intersection of L with a face is discovered.)

To test whether I lies in F, cast a half-line R in P from I in a random direction that avoids the vertices of F and that is not parallel to any edges of F. Then count the parity of the number of intersections of R with the edges of F.

The complexity of this process grows with the number of faces in S. Consider for example the case of a simple sphere S crudely approximated with 288 flat faces with a 15 degree angle between two adjacent rectangular faces. Point membership classification against S requires 282 line-plane intersection calculations in 3D, and over a thousand line-line intersection calculations in 2D. Auxiliary data structures that accelerate PMC [Hoffmann89, Horn89] may help when the associated construction costs are amortized over a large number of candidate points.

PMC against BRep models involving a larger number of curved surfaces is typically performed using a variation of the ray-casting approach outlined above, which requires intersecting half-lines with surfaces in 3D, and intersecting lines with face-bounding curves in the 2D parametric spaces of the curved surfaces.

Although many subtleties were left out, this discussion shows that even the most basic computations in a solid modeler are not trivial.

5.1.2 *Point membership classification for CSG*

Divide and conquer is the natural design paradigm for CSG algorithms. The standard point membership classification algorithm for CSG representations traverses the CSG tree by recursive descent. When the recursion reaches the primitives, the point to be classified is tested against them.

Point/primitive classification is simple if the primitives are 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.

Once point/primitive classification results are known, they are combined through the original Boolean expression that defines the CSG graph. Special processing based on neighborhood combinations may be necessary for points that lie on boundaries of several primitives [Requicha85]. A point's neighborhood is the intersection of the solid with a small ball around the point. If the neighborhood is full, the point lies inside the solid, if it is empty, the point lies outside, otherwise, the point is on the boundary of the solid. 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 inside. If the two bits are off, the point is outside. If the bits differ, the points lies on a face of the solid. Figure 7 illustrates these cases in a 2D example.

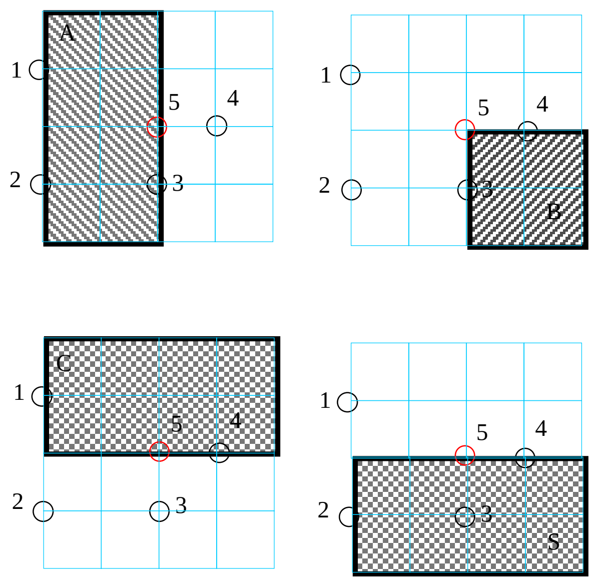

Figure 7: The solid S (bottom right) was defined in CSG by S=(A+B)-C. Points 1-4 may be evaluated using a 2-bit neighborhood. The neighborhoods of point 1 with respect to the primitives A, B and C are 01, 00, and 01, if we assign the first bit to the left side of the vertical line through the point and the second bit to the right side of that line. Combining each bit through the logical expression "(01 `OR` 00) `AND NOT` 01" yields 00 and thus point 1 lies out of S. The primitive neighborhoods for points 2 are 01, 00, and 00 and their combination yields 01. Indeed point 2 lies on S. The primitive neighborhoods for point 3 are 10, 01, and 00, and their combination is 11, making point 3 an interior point of S. The primitive neighborhoods for point 4 are 00, 01, and 10 (assigning the first bit to the side above the horizontal line and the second bit to the side below the line). Their logical combination yields 01, and hence the point lies on S. Evaluating the neighborhood of point 5 requires considering four sectors defined by the horizontal and vertical lines passing through the point. Assigning one bit to each of these sectors in clockwise order starting from the upper-right sector yields for 0011, 0100, 1001 for the three primitives and 0110 for S. Consequently, point 5 lies on S.

When the primitive faces containing point P lie on two or more supporting 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 supporting 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 points 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 non-planar and are not simple quadrics.

5.1.3 Curve membership classification

The notion of membership classification can be generalized as follows. Given a solid S and a candidate set X, set membership classification partitions X into three subsets: X_i, the subset X in the interior of S, X_b, the subset of X on the boundary of S, and X_o, the subset of X in the interior of the complement of S [Tilove80]. Membership classification for curves and surfaces with respect to solids is used in many geometric algorithms.

A simple two-step strategy can be used to classify a line or curve X with respect to a solid S represented either by a BRep or CSG. (There are other, more efficient approaches to curve/solid classification.) In the first step, the curve is segmented into subsets that have uniform classification with respect to S. Then, the classification for each segment is determined, for example by classifying its mid point using the PMC algorithms discussed earlier. Which PMC algorithm to use depends on whether the solid is represented by a BRep or CSG. The first step also depends on the solid's representation. If the solid is given by a BRep, we compute the isolated intersection points of X with all the faces, edges, and vertices of S and represent them in terms of the corresponding parameter values of some parameterization of X. (Cases where X is contained in a surface that supports a face of S or overlaps with a curve that supports an edge of S need not be processed.)

Sorting these parameter values defines segments of X that have uniform classification with respect to S. (The classification of X can only change at the computed intersection points.)

If the solid is represented in CSG, in the first step we intersect the curve with the faces, edges and vertices of all the *primitives*, and sort the parameter values of the intersection points to create the initial segmentation.

5.1.4 Boolean Operations

Consider a solid S defined by a regularized Boolean operation on two argument solids A and B, both defined in CSG. The boundary of S may be computed by subdividing the boundaries of A and B at their intersection and by discarding the appropriate 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, 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 non *incremental boundary evaluation*, process.

We describe briefly a non-incremental boundary evaluation algorithm, to illustrate the issues. Typically, faces of CSG solids are represented in terms of their supporting 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 the interior of S, in its complement, or on its boundary 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 illustrated in [Banerjee96] for CSG models composed of polyhedral primitives. First a tentative set of lines if generated as the intersection of each possible pair of non-parallel planes taken from the set **P** of planes that support the faces of the primitives of the CSG definition of **S**. Each line L is classified against **S** as follows. First, separate the planes of **P** into three sets: **C**, the planes containing **L**; **I**, the planes intersecting **L** at a single point, and **O** the planes disjoint from **L**. Then use the intersections with planes of **I** to split **L** into segments. Split the neighborhood of **L** into sectors defined by the planes of **C**. The classification of each sector for each segment may be represented by a single bit. Because S and its primitives are finite, the classification of any given sector of a semi-infinite segments of L with respect to the primitives and nodes of the CSG tree of S is readily known and may be stored by associating a bit with each node. Moving from one sector to an adjacent sector in the same segment or in an adjacent segment corresponds to the traversal of a single supporting plane, and therefore alters the classification bit of a single primitives or of several primitives that have faces supported by this plane. Changing this classification bit and updating the CSG tree may or may not alter the classification bit of S. By traversing all these sectors from neighbor to neighbor and by storing the associated classification bits with the sectors, we compute the neighborhood of each segment. Edges with full or empty neighborhood are discarded. Edges with identical neighborhoods are merged.

By keeping track of the association between curves and surfaces, and between vertices and segments, a full BRep may be inferred form 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. Alternatively, the faces of CSG solids may be represented as the intersection between a supporting surface and a 3D trimming volume, whose CSG expression may be easily derived from the CSG of the entire solid [Rossignac96].

Other algorithms for boundary evaluation and merging typically use also the generate and test paradigm, but may be articulated around vertices, edges or faces (see [Mantyla86] for an example and [Requicha85] 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.

5.1.5 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, the modifications must be reflected on the associated CSG. The 2D case is fairly well understood [Shapiro91]. Algorithms for converting incomplete polyhedral boundary representations into a BSP tree have been proposed in [Murali97]. These algorithms are very useful for disambiguating models represented by incomplete boundaries where faces-face adjacency may have been perturbed by numeric approximation.

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 [Lee82]. Conversion from CSG or BRep to octree format is even more useful for mass-property calculations, and can be done also through membership classification algorithms.

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. It can be accomplished by classifying a set of parallel planes with respect to a solid. The portions of the planes inside the solid are the desired slices.

5.1.6 Curve and Surface Intersections

The curve/solid classification algorithms discussed earlier are an example of computations that require intersecting a curve with surfaces of composite or primitive solids. Boolean operation algorithms also need to intersect surfaces with other surfaces. Curve and surface intersections arise in many other geometric algorithms.

Suppose that we are given a curve with parametric equations x=x(u), y=y(u), z=z(u), plus an algebraic surface with implicit equation f(x,y,z)=0. Curve/surface intersection amounts to solving for the roots of the system of four equations: x=x(u), y=y(u), z=z(u), and f(x,y,z)=0.

This can be done by substituting the parametric equations in the implicit equation to obtain g(u)=f(x(u), y(u), z(u))=0 and solving this equation for u. Except in very simple cases the solution can only be found numerically.

This simple example shows that intersection problems essentially consist of root finding. Much is known about this area, but the problem of designing algorithms that are *both* robust and efficient is still largely open. For surveys and representative research see [Patrikalakis93, Krishnan97].

5.2 Applications

Applications that involve *analysis* of models are relatively well developed. Algorithms are available for generating displays of solids in many styles and degrees of realism, for kinematic simulation, for the evaluation of mass properties, for collision detection in static environments, and so on. For example, graphics applications are flourishing in the entertainment industry. On the other hand, problems such as design and planning, which involve *synthesis*, are much less understood. We speculate that the difficulties arise because synthesis algorithms must reason about space, which is notoriously hard to do computationally.

Nevertheless, applications such as feature recognition for machining planning [Vandenbrande93, Han97] dimensional inspection planning [Spyridi90, Spyridi94] and robot path planning [Latombe91] are getting close to industrial usability.

Most of these application algorithms use extensively the fundamental queries and constructions described above. For example, mass property calculation for CSG solids typically involves either line/solid classification or CSG to octree conversion. And feature recognition and accessibility analysis for inspection planning require Boolean operation capabilities.

A growing number of solid modeling applications focus not on the design of the geometry of the individual components, but on their exploitation for the interactive design and inspection of manufacturing processes (assembly plans, NC cutter path generation) and of spatial arrangements (manufacturing cell layout, accessibility to control instruments in aircrafts, or urban planing.

5.3 Efficiency Enhancements

Set membership classification and CSG to BRep conversion algorithms perform a very large amount of computation when compared to 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 good 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.

6 Parameters, Constraints and Features

Regardless of the representation scheme used, building and editing a model for a complicated solid is non-trivial. Finding the size and position parameters that are needed to ensure that geometric entities are in desired relationships often is quite difficult. 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.

6.1 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 of the 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 1970's with the PADL-2 solid modeling system [Brown82], 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 [Rossignac86b]. Each constraint corresponds to a transformation that brings the host surface of a primitive face into a specified relationship 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 [Rossignac88b]. 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 parameters 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.

6.2 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 bi-directional constraints from all singular situations (such as parallelism, orthogonality or tangency) that are detected on a nominal model. A constraint solver 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, variational geometry is typically used in conjunction with a parameterized boundary representation. The variational geometry approach is popular for 2D drafting and for designing solids that are extruded from 2D 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.

6.3 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 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 [Rossignac90]. 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 amount 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 [Vandenbrande93, Han97].

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* [Rossignac89] invented by Rossignac and Voelcker, 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. Except for limited situations, no automatic technique is currently available for deriving such filters.

7 User interfaces

A solid modeler's user interface defines the skills required to use the modeler and has a major impact on the users' effectiveness. Early solid modeling systems were reserved to CAD experts in the aerospace and automotive industries and were focused on providing designers with powerful and robust design operations and analysis. Although much progress is still needed on these fronts, considerable research and development efforts have been recently re-focused on improving the ease-of-use for non-experts 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 non-traditional areas (such as electronic components or entertainment) where accessibility to non-specialists and rapid design cycles are more important than precision.

Very often a complex 3D 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 3D browsers that support the review and annotation of 3D 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 design and analysis components controlled from such a browser. The remainder of this section discusses performance and ease-of-use issues related to interactions with highly complex models.

7.1 Graphics

Solid modelers may be rendered in different styles. Photo-realistic images of complex scenes require extensive computation and are typically used at the final illustration stage of finished models. Simpler shaded images are generally used to support interactive manipulation during design and analysis.

7.1.1 Triangulation

Solid models are typically approximated by triangles (or sometimes convex polyhedra) before rendering, because graphics libraries and hardware graphics accelerators have been tuned to render such simple shape-primitives very fast. Arbitrary polyhedral faces must be decomposed into triangles. Curved surfaces must be approximated by triangular tesselations, which may automatically adapt their resolution so as to guarantee the desired accuracy. Triangulating trimmed surfaces requires combining the 3D surface tessellation with the triangulation of the face representation in the 2D parameter space of the surface. The implementation of such triangulations must ensure that the triangulations of any two adjacent faces do not generate gaps, which would arise if the two triangulations were to produce different piecewise linear approximation of the common edge of the two faces.

7.1.2 Graphics acceleration and compression

Simple scenes in 3D video-games may only need a few hundred textured polygons, but solid models used for mechanical CAD, scientific, geo-science, and medical applications involve scenes with millions of faces, sometimes grouped to form the boundaries of polyhedral solids of widely varying complexity. The successful exploitation of such large volumes of scientific and engineering 3D data hinges on users' ability to access the data through phone lines or network connections and to manipulate significant portions of these 3D models interactively on the screen for scientific discovery, teaching, collaborative design, or engineering analysis. Currently available high-end graphics hardware is often insufficient to render the models at sufficient frame rates to support direct manipulation and interactive camera control.

The complexity (triangle count) of industrial and sometimes even entertainment models significantly exceeds what can be rendered on a mid-range or low-end workstation at interactive rates (15 frames per second or more). Software techniques, which eliminate unnecessary or unessential rendering steps, lead to dramatic performance improvements and hence reduce hardware costs for graphics. Many of these techniques require not only complex algorithmic pre-processing. Furthermore, it is often necessary to trade the accuracy of the images for performance during interactive view manipulation. The following are examples of such techniques.

Triangles are often ordered to form long triangle strips which reduce by 3 the number of times the same vertex is processed by the graphics library. Each new triangle is defined by combining the next vertex of the strip with the previous two vertices.

Triangles may be grouped by orientation and entire groups of back-facing triangles may be readily rejected without further considerations. Objects or triangles that do not intersect the viewing frustum or are hidden behind large occluding objects may also be quickly identified and need not be processed for rendering.

The remaining (potentially visible) triangles may still be too numerous. Solids that are far from the viewer (i.e. whose projection on the screen is small) may be rendered using "impostors" which resemble the original solid but have significantly fewer triangles. Numerous simplification algorithms have been developed for producing series of approximating models with decreasing levels-of-detail (LOD). The simplest and most efficient approach clusters vertices by comparing their truncated coordinates and eliminates triangles with more than one vertex in the same cluster [Rossignac93]. Many slower and more complicated, although more effective simplification techniques have been proposed. For example, the edges of a triangulated boundary may be collapsed in an order which introduces the least amount of error [Ronfard96], until the desired triangle count or the maximum allowable error have been reached. Collapsing an edge also collapses the two triangles incident upon the edge.

In many industrial applications, complex 3D models approximated by millions of triangles, must be archived, downloaded using limited bandwidth connections, and finally transmitted to the rendering subsystem at each frame (or stored on the graphics board's limited local memory). Representing each triangle by the floating point coordinates of its three vertices requires 26 bytes, to which one must often add rendering attributes, such as texture coordinates or vertex normals for smooth shading. This count may be reduced by more than a factor of two if one were to use triangle strips. Separating the description of the vertex locations from the description of the triangle-vertex incidence eliminates the need to store or send the same vertex twice, but requires storing for each triangle, the references to its supporting vertices. In its non-compressed form, each one of these references may involve a number of bits equal to the log of the total number of vertices.

A lossy compression to the vertex coordinates, which combines vertex quantization and entropy coding, reduces by three or more the number of bits that represent the location of the vertices [Deering95]. Triangle-vertex incidence relations may be encoded with less than 2 bits per triangle [Taubin96]. An alternative, which does not require random access to all the vertices, generalizes the notion of triangle strips to a stack-buffer of 16 vertices [Deering95]. Although this approach requires on average about 13 bits per triangle to store the triangle-vertex incidence and requires sending about 15% of the vertices more than once, it is particularly effective for interfacing to a graphics subsystem with limited on-board memory.

7.2 3D input

The human-machine communication during the design activity involves mostly the selection and entry of commands and parameter values and increasingly the direct manipulation of the model on the screen to control the position of the camera or of a particular group of solids in an assembly. Although this manipulation of 3D positions was traditionally performed with a 2D mouse, new techniques are being explored for using two-hand interfaces [Zeleznik97] or immersive virtual reality environments. A natural alternative uses pen-and-tablet 2D input and generates solid 3D models from drawings [Grimstead95]. These approaches have to deal with a lack of information that implies ambiguity.

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

The recent 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 towards commercial systems, research efforts are focused on making non-expert 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. Progress is still inhibited by the network bandwidth for accessing highly complex assembly models, by the limited graphics performance of personal computers, and by unnatural 3D manipulation interfaces. Recent progress on these fronts are reviewed in [Rossignac97b].

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.

Current trends point away from closed end-user modelers and towards flexible object-oriented libraries supporting a broader range of geometric computations. Access to these libraries may be provided from easy-to-use 3D browsers.

9 Bibliography

[Agrawal94] A. Agrawal and A. A. G. Requicha, "A paradigm for the robust design of algorithms for geometric modeling", Computer Graphics Forum, Vol. 13, No. 3, pp. 33-44, September 1994. (Proc. Eurographics '94.)

[Alexandroff61] P. Alexandroff, Elementary Concepts of Topology, Dover Publications, New York, NY, 1961.

[Banerjee96] R. Banerjee and J. Rossignac, Topologically exact evaluation of polyhedra defined in CSG with loose primitives, to appear in Computers Graphics Forum, Vol. 15, No. 4, pp. 205-217, 1996.

[Barr84] A. Barr, Local and global deformations of solid primitives, Proc. Siggraph'84, Computer Graphics, Vol. 18, No. 3, pp. 21-30, July 1984.

[Baumgart72] B. Baumgart, Winged Edge Polyhedron Representation, AIM-79, Stanford University Report STAN-CS-320, 1972.

[Bowyer95] A. Bowyer, S. Cameron, G. Jared, R. Martin, A. Middleditch, M. Sabin, J. Woodwark, *Introducing Djinn: A Geometric Interface for Solid Modeling, Information* Geometers Ltd, 1995.

[Brown82] C. Brown, PADL-2: A Technical Summary, IEEE Computer Graphics and Applications, 2(2):69-84, March 1982.

[Deering95] M. Deering,Geometry Compression, Computer Graphics, Proceedings Siggraph'95, 13-20, Augiust 1995.

[Farin90] G. Farin, *Curves and Surfaces for Computer-Aided Geometric Design,* Second edition, Computer Science and Scientific Computing series, Academic Press, 1990.

[Grimstead95] I.J. Grimstead and R.R. Martin, Creating solid models from single 2D sketches, Proc. Third Symposium on Solid Modeling and Applications, ACM Press, pp. 323-337, May 17-19, 1995.

[Han97] J.-H. Han and A. A. G. Requicha, Integration of feature based design and feature recognition, Computer-Aided Design, Vol. 29, No. 5, pp. 393-403, May 1997.

[Hoffmann89] C. Hoffmann. Geometric and Solid Modeling. Morgan Kaufmann, San Mateo, CA, 1989.

[Horn89] W. Horn and D. Taylor, A theorem to determine the spatial containment of a point in a planar polyhedron, Computer Vision, Graphics, and Image Processing, 45:106-116, 1989.

[Krishnan97] S. Krishnan and D. Manocha, An efficient surface intersection algorithm based on lower-dimensional formulation, ACM Transactions on Graphics, Vol. 16, No. 1, pp. 74-106, January 1997.

[Latombe91] J. Latombe, Robot Motion Planning, Kluwer, Boston, 1991.

[Lee82] Y. T. Lee and A. A. G. Requicha, 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. of the ACM, Vol. 25, No. 9, pp. 635-650, September 1982.

[Mantyla86] M. Mantyla, Boolean Operations of 2-manifold Through Vertex Neighborhood Classification, ACM Trans. on Graphics, 5(1):1-29, 1986.

[Murali97] T.M. Murali and T.A. Funkhouser, Consistent solid and boundary representations from arbitrary polygonal data, Proc. 1997 Symposium on Interactive 3D Graphics, ACM Press, pp. 155-162, Providence, R.I., April 1997.

[Naylor90] B. Naylor, J. Amanatides and W. Thibault, Merging BSP Trees Yields Polyhedral Set Operations, ACM Computer Graphics SIGGRAPH '90, 24(4):115-124, August 1990.

[Patrikalakis93] N.M. Patrikalakis, Surface-to-surface intersections, IEEE Computer Graphics and Applications, Vol. 13, No. 1, pp. 89-95, 1993.

[Requicha80] Representation of Rigid Solids: Theory, Methods, and Systems, A.A.G. Requicha, ACM Computing Syrveys, 12(4), 437:464, Dec 1980.

[Ralston83] A. Ralston and E. Reilly, Editors, Encyclopedia od Computer Science and Engineering, second Edition, van Nostrand Reinhold Co., New York, pp .97-102, 1983.

[Requicha82] A. A. G. Requicha and H. B. Voelcker, Solid modelling: a historical summary and contemporary assessment", IEEE Computer Graphics and Applications, Vol. 2, No. 2, pp. 9-24, March 1982.

[Requicha83] A. A. G. Requicha, Toward a theory of geometric tolerancing, Int. J. of Robotics Research, Vol. 2, No. 2, pp. 45-60, Winter 1983.

[Requicha83b] A. A. G. Requicha and H. B. Voelcker, "Solid modelling: current status and research directions", IEEE Computer Graphics and Applications, Vol. 3, No. 7, pp. 25-37, October 1983.

[Requicha85] A. A. G. and H. B. Voelcker, "Boolean operations in solid modelling: boundary evaluation and merging algorithms", Proc. IEEE, Vol. 73, No. 1, pp. 30-44, January 1985.

[Requicha88] A. A. G. Requicha, "Solid modelling: a 1988 update", in B. Ravani, Ed., CAD Based Programming for Sensory Robots. New York: Springer Verlag, 1988, pp. 3-22.

[Requicha92] A. A. G. Requicha and J. R. Rossignac, "Solid modeling and beyond", IEEE Computer Graphics & Applications, (Special issue on CAGD) Vol. 12, No. 5, pp. 31-44, September 1992.

[Requicha93] A. A. G. Requicha, "Mathematical definition of tolerance specifications", ASME Manufacturing Review, Vol. 6, No. 4, pp. 269-274, December 1993.

[Requicha96] A. A. G. Requicha, "Geometric reasoning for intelligent manufacturing", Commun. ACM, Special Issue on Computer Science in Manufacturing, Vol. 39, No. 2, pp. 71-76, February 1996.

[Ronfard96] R. Ronfard and J. Rossignac, Full-range approximations of triangulated polyhedra, Computer Graphics Forum, (Proc. Eurographics'96), pp. C-67, Vol. 15, No. 3, August 1996.

[Rossignac84] J. Rossignac and A. Requicha, Constant-Radius Blending in Solid Modeling, ASME Computers in Mechanical Engineering (CIME), Vol. 3, pp. 65-73, 1984.

[Rossignac86] J. Rossignac and A. Requicha, Offsetting Operations in Solid Modelling, Computer-Aided Geometric Design, Vol. 3, pp. 129-148, 1986.

[Rossignac86b] J. Rossignac and A. Requicha, Depth Buffering Display Techniques for Constructive Solid Geometry, IEEE Computer Graphics and Applications, 6(9):29-39, September 1986.

[Rossignac86b] J. Rossignac, Constraints in Constructive Solid Geometry, Proc. ACM Workshop on Interactive 3D Graphics, ACM Press, pp. 93-110, Chapel Hill, 1986.

[Rossignac88b] J. Rossignac, P. Borrel, and L. Nackman, Interactive Design with Sequences of Parameterized Transformations, Proc. 2nd Eurographics Workshop on Intelligent CAD Systems: Implementation Issues, April 11-15, Veldhoven, The Netherlands, pp. 95-127, 1988.

[Rossignac89] J. Rossignac and H. Voelcker, Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection and Shading Algorithms, ACM Transactions on Graphics, Vol. 8, pp. 51-87, 1989.

[Rossignac89b] J. Rossignac and M. O'Connor, SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries, in *Geometric Modeling for Product Engineering*, Eds. M. Wosny, J. Turner, K. Preiss, North-Holland, pp. 145-180, 1989.

[Rossignac90] J. Rossignac, Issues on feature-based editing and interrogation of solid models, Computers&Graphics, Vol. 14, No. 2, pp. 149-172, 1990.

[Rossignac91] J. Rossignac, and A. Requicha, Constructive Non-Regularized Geometry, Computer-Aided Design, Vol. 23, No. 1, pp. 21-32, Jan./Feb. 1991.

[Rossignac92b] J. Rossignac, Through the cracks of the solid modeling milestone, in *From object modelling to advanced visualization*, Eds. S. Coquillart, W. Strasser, P. Stucki, Springer Verlag, pp. 1-75, 1994.

[Rossignac93] J. Rossignac and P. Borrel, Multi-resolution 3D approximations for rendering complex scenes, pp. 455-465, in Geometric Modeling in Computer Graphics, Springer Verlag, Eds. B. Falcidieno and T.L. Kunii, Genova, Italy, June 28-July 2, 1993.

[Rossignac94] J. Rossignac and A. Kaul, AGRELs and BIPs: Metamorphosis as a Bezier curve in the space of polyhedra, Computer Graphics Forum, Vol 13, No 3, pp. C179-C184, Sept 1994.

[Rossignac96] J. Rossignac, CSG formulations for identifying and for trimming faces of CSG models, CSG'96: Set-theoretic solid modeling techniques and applications, Information Geometers, Ed. John Woodwark, pp.1-14, 1996.

[Rossignac97] J. Rossignac, Structured Topological Complexes: A feature-based API for non-manifold topologies", Proceedings of the ACM Symposium on Solid Modeing 97, C. Hoffmann and W. Bronsvort, Editors, ACM Press, pp. 1-9, 1997.

[Rossignac97b] J. Rossignac, The 3D revolution: CAD access for all, International Conference on Shape Modeling and Applications, Aizu-Wakamatsu, Japan, IEEE Computer Society Press, pp. 64-70, 3-6 March 1997.

[Sedeberg86] T. Sedeberg and S. Parry, Free-form deformation of solid geometric models, ACM Computer Graphics (Proc. Siggraph), 20(4):151-160, Dallas, August 1986.

[Spyridi93] A. J. Spyridi and A. A. G. Requicha, "Automatic planning for dimensional inspection", ASME Manufacturing Review, Vol. 6, No. 4, pp. 314-319, December 1993.

[Taubin96] G. Taubin and J. Rossignac, Geometric Compression through Topological Surgery, IBM Research Report RC-20340. January 1996. (http://www.watson.ibm.com:8080/PS/7990.ps.gz).

[Samet90] Hanan Samet, *Applications of Spatial Data Structures*, Reading, MA, Addison-Wesley, 1990

[Shapiro91] V. Shapiro and D. Vossler. Construction and Optimization of CSG Representations, Computer-Aided Design, 23(1):4-20, January/February 1991.

[Spyridi90] A. J. Spyridi and A. A. G. Requicha, "Accessibility analysis for the automatic inspection of mechanical parts by coordinate measuring machines", Proc. IEEE Int'l Conf. on Robotics & Automation, Cincinnati, OH, pp. 1284-1289, May 13 - 18, 1990.

[Spyridi94] A. J. Spyridi and A. A. G. Requicha, "Automatic programming of coordinate measuring machines", Proc. IEEE Int'l Conf. on Robotics & Automation, S. Diego, CA, pp. 1107-1112, May 8 - 13, 1994.

[Tilove80] R. Tilove, Set Membership Classification: A Unified Approach to Geometric Intersection Problems, IEEE Trans. on Computers,, C-29(10):874-883, October 1980.

[Vandenbrande93] J. H. Vandenbrande and A. A. G. Requicha, “Spatial reasoning for the automatic recognition of machinable features in solid models”, IEEE Trans. Pattern Analysis & Machine Intelligence, Vol. 15, No. 10, pp. 1269-1285, December 1993.

[Weiler87] Non-Manifold Geometric Boundary Modeling, K. Weiler, ACM Siggraph, Tutorial on Advanced Solid Modeling, Anaheim, California, July 1987.

[Zeleznik97] R.C. Zeleznik, A.S. Forsberg, and P. Strauss, Two pointer input for 3D interaction, Proc. 1997 Symposium on Interactive 3D Graphics, ACM Press, pp. 115-120, Providence, R.I., April 1997.

10 Further reading

M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988. (Detailed presentation of som solid representations and of algorithms for regularized Boolean operations.)

A. Bowyer, S. Cameron, G. Jared, R. Martin, A. Middleditch, M. Sabin, J. Woodwark, *Introducing Djinn: A Geometric Interface for Solid Modeling, Information* Geometers Ltd, 1995. (Definition of a set-theoretic API independent of the representation scheme used.)

J. Rossignac and J. E. Turner, Editors, Proc. ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM Press, Austin, Texas, 1991. (About 40 papers devoted to solid modeling.)

J. Rossignac, J. Turner, and G. E. Allen, Editors, Second ACM Symposium on Solid Modeling and Applications, ACM Press, Montreal, Canada, 1993. (About 40 papers devoted to solid modeling.)

C. Hoffmann and J. Rossignac, Editors, Proc. Third Symposium on Solid Modeling and Applications, ACM Press, Salt Lake City, Utah, 1995. (About 40 papers devoted to solid modeling.)

J. Rossignac, Simplification and Compression of 3D Scenes, Tutorial, Eurographics’97, Budapest, Hungary, September 1997. (Overview of advances in polyhedral simplification and compression.)

Representing Solids and Geometric Structures, in Geometry and Optimization Techniques for Structural Design, pp. 1-44, Eds. S. Kodiyalam, M. Saxena, Computational Mechanics Publications, Southhampton, 1993.