57 SOLID MODELING

Christoph M. Hoffmann and Vadim Shapiro

INTRODUCTION

The objective of solid modeling is to represent, manipulate, and reason about the 3D shape of solid physical objects, by computer.

As an application-oriented field, the scope of solid modeling, as well as the relative emphasis on its parts, change over time. Since its inception, [RV82, Bra75], in the 1970s, the focus on modeling shape alone has continued to expand to integrate widening domains of physical properties. Monographs of the subject appeared in the late 1980s and include [Chi88, Hof89, Män88]. The expansion was and continues to be driven by major applications that include manufacturing, architecture, construction, computer vision, materials science, medicine, biological engineering, graphics, and virtual reality. With the integration of new applications, and with the availability of inexpensive, powerful computing platforms, the relative importance of specific shape representations can change. Along with such shifts, new modeling and analysis techniques are added. As a result, the field draws on very diverse technologies, including numerical analysis, symbolic algebraic computation, approximation theory, point set and algebraic topology, differential geometry, algebraic geometry, and computational geometry.

In this chapter, we first review the major representations of solids in Section 57.1. They include constructive solid geometry, boundary representation, spatial subdivisions of various types, and medial surface representations. With changing scope and focus, different representations enter and exit center stage. For instance, polygonal meshes and voxel-based representations became relatively marginal in the late 1990s, only to gain renewed interest and importance recently with the advent of 3D printing, because they allow representing a structured interior of modeled shapes. Procedural and declarative representations are becoming more popular as geometric programming methods are increasingly used for creating complex solid models and assemblies in architecture and additive manufacturing.

For decades modeled solids were generally assumed to have a homogeneous interior. This fact reflected common manufacturing applications and manufacturing processes that separated the production of raw materials from subsequent processing of those materials into parts, assemblies, etc. With the advent of additive manufacturing, that is, machinery that builds physical objects by additive processes, laying down material layer by layer, there is growing interest in building solids that have a complex interior structure. An efficient and comprehensive representation of heterogeneous interior has not yet emerged. Along with explorations of printer technologies and of what can or should be built there is a wide-ranging, diverse body of research, that we will comment on in Section 57.1.7, tracing this development.

Next, major layers of abstraction in a typical solid modeling system are characterized in Section 57.2. The lowest level of abstraction comprises a substratum of basic service algorithms. At an intermediate level of abstraction, there are algorithms for larger, more conceptual operations. Many functions and operations have been devised over the years and at varying levels of abstraction. Note that this layer can be further sub-structured into a variety of levels. Finally, a yet higher level of abstraction is offered by constraint- and feature-based designs: the constraints and the feature parameters, index instance designs. The resulting generic design, prior to valuating parameters and constraints, defines families of shape instances, and ventures into territory that is only partially mastered, mathematically and computationally.

The rich infrastructure of solid modeling representations and operations can be accessed in several ways:

1. A graphical user interface (GUI) presents the capabilities, usually targeting specific application areas. This is by far the most common type of access.
2. An application programming interface (API) presents the infrastructure much like any other software library. Most systems have an API, but the vendor API does not necessarily expose every system functionality.
3. A current trend in system architecture is a shift toward modularized confederations of plug-compatible functional components. Using a plug-and-play perspective, components with a specific functionality collaborate in the system. This can be done in two ways: (a) design and analysis data is exchanged and/or shared by the system components, or (b) components query each other, so accumulating needed information to do their work.

The first style of plug-and-play requires that the data is understood by the cooperating components. If the components come from different vendors, understanding the exchanged data has to overcome the absence of a mathematically sound semantics of the data. Approach (3a) thus favors systems implemented by the same vendor who may use an idiosyncratic data format but can deliver a consistent interpretation. Approach (3b) has an object-oriented character. Moreover, it takes a page from *Geometric Dimensioning and Tolerancing*; e.g., [Sri03]. This recent work seeks to abstract the role of a solid model in applications, reducing the interface to a set of queries so as to avoid representation translation/interpretation altogether. All this will be discussed in Section 57.3. Open problems are gathered in Section 57.4.

57.1 MAJOR REPRESENTATION SCHEMATA

GLOSSARY

Solid representation: Any representation allowing a deterministic, algorithmic point membership test.

Constructive solid geometry (CSG): The solid is represented as union, intersection, and difference of primitive solids that are positioned in space by rigid-body transformations.

Boundary representation (Brep): The solid surface is represented as a quilt of vertices, edges, and faces.

Mesh representation: A boundary representation whose faces are planar polygons. Topological information may be reduced or implicit.

Spatial subdivision: The solid is decomposed into a set of primitive volumes with nonintersecting interior, for instance voxels.

A solid representation must allow the unambiguous, algorithmic determination of point membership: given any point $p = (x, y, z)$, there must be an algorithm that determines whether the point is inside, outside, or on the surface of the solid; [Req77]. Moreover, restrictions are placed on the topology of the solid and its embedding, excluding, for example, fractal solids.

These restrictions are eminently reasonable. Increasingly, however, solid modeling systems depart from this strict notion of solid and permit representing a mixture of solids, surfaces, curves, and points, for example, in surface modeling in graphics via "particle systems." The additional geometric structures are useful for certain design processes, for interfacing with applications such as meshing solid volumes, and for abstracting solid features, to name a few. More than that, solid models used to partition space into three point sets: points exterior to the solid, points contained in the solid, and points on the boundary, separating inside from outside. This is changing with a growing interest in representing structured solid interiors.

GEOMETRIC COVERAGE

The range and geometric representation of solid surfaces is referred to as *geometric coverage*. Polyhedral modeling restricts to planes. Classical CSG allows only planes, cones, cylinders, spheres, and tori. Experimental modelers have been built allowing arbitrary algebraic halfspaces. SGDL used implicit algebraic surfaces of degree up to 4.

Most commercial and many research modelers use B-splines (uniform or nonuniform, integral or rational) or Bézier surfaces. The properties and algorithmic treatment of these surfaces are studied by computer-aided geometric design. See Chapter 56, as well as the monographs and surveys [Far88, Hos92, HL93].

57.1.1 CONSTRUCTIVE SOLID GEOMETRY

GLOSSARY

Primitive solids: Traditionally: block, sphere, cylinder, cone, and torus. More general primitives are possible.

Sweep: Volume covered by moving a solid or a closed contour in space.

Extrusion: Sweep along a straight line segment.

Revolution: Circular sweep.

Regularized Boolean operation: The closure of the interior of a set-theoretic union, intersection, or difference.

Algebraic halfspace: Points such that $f(x, y, z) \leq 0$ where f is an irreducible polynomial.

Irreducible polynomial: Polynomial that cannot be factored over the complex numbers.

Constructive Solid Geometry is a special case because it is not only a particular representation of solid shapes, but is also a methodology for composing primitive solids to represent complex solid shapes. Visual representation of a CSG solid, moreover, soon was done converting the CSG representation into boundary representation for faster rendering. In this section, we restrict to the representational aspect of CSG.

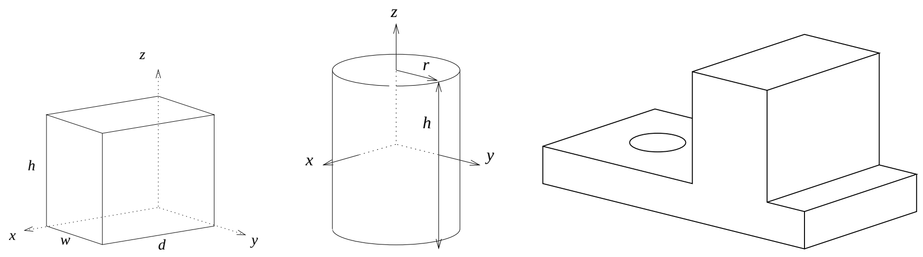
Classical Constructive Solid Geometry (CSG) represents a solid as a set-theoretic Boolean expression of *primitive* solid objects, of a simpler structure [RV77]. The traditional CSG primitives are block, sphere, cylinder, cone, and torus. The traditional operations are regularized union, intersection, and difference. A regularized set operation is obtained by taking the closure of the interior of the set-theoretic result.

Each solid has a default coordinate system that can be changed with a rigid body transformation. A Boolean operation identifies the two coordinate systems of the solids to be combined and makes it the default coordinate system of the resulting solid.

Primitives, and solids obtained from them with the CSG operations, are represented by expressions that can be conceptualized as expression trees. The leaves are the primitives, the internal tree nodes are rigid-body transformations and regularized Boolean operations.

Both the surface and the interior of the final solid are thereby defined, albeit implicitly. The CSG representation is valid if the primitives are valid.

FIGURE 57.1.1
Left and middle: CSG primitives $\texttt{block}(w, d, h)$ *and* $\texttt{cylinder}(r, h)$ *with default coordinate systems. Right: T-bracket as union of two blocks minus a cylinder.*



As an example, consider Figure 57.1.1. Using the coordinate system conventions shown, the CSG representation of the bracket is the expression

$$\begin{aligned}\texttt{block}(8,3,1) \cup^* \texttt{move}(\texttt{block}(2,2.5,3),(0,4.5,1))\\ -^* \texttt{move}(\texttt{cylinder}(0.75,1),(1.5,1.5,-0.5))\end{aligned}$$

where the * indicates a regularized operation. (See also Figure 42.4.1.)

Algorithmic infrastructure operations the system can perform include classifying points, curves, and surfaces with respect to a solid; eliminating redundancies in the CSG expressions; and rendering solids visually. A CSG solid can be rendered by ray tracing directly, or by converting the solid to a boundary representation and rendering the resulting data structure.

More general primitives may be added if they support the CSG operations. Examples include the volume covered by sweeping a solid along a space curve, or by sweeping a planar contour bounding an area. Defining a sweep is delicate, requiring many parameters to be exactly defined; for instance consider the operation of blending discussed in Section 57.2.2. However, simple cases are widely used. They include extrusion, i.e., sweep along a straight line; and revolution, i.e., a sweep about an axis. The evaluation of general sweeps can be accomplished by a number of methods.

CSG representations define unambiguously the surface and homogeneous interior of solids, provided the primitives do, a fact that is straightforward. In contrast, boundary representations must satisfy both local and global properties if they are to partition space in a like manner [Wei86].

57.1.2 BOUNDARY REPRESENTATION

In boundary representation (Brep), the surface of a solid is explicitly represented, and the interior is implicitly represented. As before, the interior is assumed to be homogeneous. The solid surface is represented as a quilt of faces, edges, and vertices; [Bra75]. A distinction is drawn between the topological entities, vertex, edge, and face, related to each other by incidence and adjacency, and the geometric location and shape of these entities. See also Figure 57.1.2. For example, when polyhedra are represented, the faces are polygons described geometrically by a face equation plus a description of the polygon boundary.

Face equations are often parametric, with nonuniform rational B-splines the dominant form. They can also be implicit algebraic, perhaps of limited maximum degree. The SGDL modeler used implicit algebraic surface patches of degree 4 or less.

Geometrically, the entities in a Brep are not permitted to intersect anywhere except in edges and vertices that are explicitly represented in the topology data structure. In addition to the classification operations mentioned for CSG, Boolean union, intersection, and difference operations are usually implemented for Brep systems. Both regularized and nonregularized Boolean operations may occur. Early systems maintained Brep and CSG representations in parallel; [RS00].

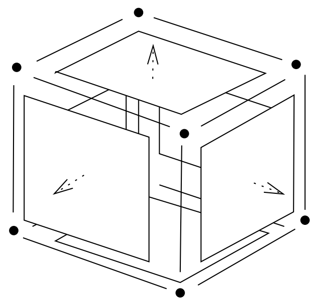


FIGURE 57.1.2
Topological entities of a box. Adjacency and incidence are recorded in Brep. Dotted arrows indicate face orientation.

Different Brep schemata appear in the literature, divided into two major families. One family restricts the solid surfaces to oriented manifolds. Here, every edge is incident to two faces, and every vertex is the apex of a single cone of incident edges and faces. The second family of Brep schemata allows oriented nonmanifolds in which edges are adjacent to an even number of faces. When these faces are ordered radially around the common edge, consecutive face pairs alternatingly bound solid interior and exterior. See Figure 57.1.3 for examples.

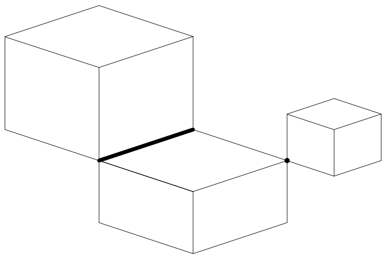


FIGURE 57.1.3
A nonmanifold solid without dangling or interior faces, edges, and vertices; the nonmanifold edges and vertices are drawn with a thicker pen.

More general nonmanifold Breps are used in systems that combine surface modeling with solid modeling [Tak92]. In such representation schemata, a solid may have interior (two-sided) faces, dangling edges, and so on. Solid modelers often integrate surface modeling capabilities.

The topology may be restricted in other ways. For instance, the interior of a face may be required to be homeomorphic to a disk, and edges to have two distinct vertices. In that case, the Brep of a cylinder would have at least four faces, two planar and two curved. This may be desirable because of the geometric surface representation, or may be intended to simplify the algorithms operating on solids.

All such topological representations may be viewed as instances of a chain complex, a concept commonly used in algebraic topology that involves representing a sequence of boundary relationships between the spaces of k-chains constructed over cellular decomposition of a solid. When the boundary operators are represented by sparse incidence matrices, many boundary representations and algorithms reduce to problems in linear algebra [DPS14].

57.1.3 MESH REPRESENTATIONS

GLOSSARY

Mesh boundary representation: A simplified boundary representation in which faces are planar, often polygons with few vertices.

Triangulated surface representation: A boundary mesh representation where all faces are triangles.

STL file format: A triangle mesh specification widely used in 3D printing.

Mesh representations of solids restrict to planar faces. Some topological information is usually given, for instance by using vertex identifiers when defining faces. Which side of a face is to the outside of the solid could be decided using a right-hand rule and restricting to convex polygons, or be based on locally outward pointing vertex normals; see, e.g., [CMS98]. Mesh and octree representations are treated in [BN90, Hof95, Sam89a, Sam89b, TWM85], including the associated conversion problems.

The STL file format, used in 3D printing, restricts to triangles and uses vertex normals; e.g., [RW91]. Vertex coordinates are local to the triangle defined. Consequently no adjacencies are explicit and all topology has to be reconstructed/inferred.

57.1.4 SPATIAL SUBDIVISION REPRESENTATIONS

GLOSSARY

Boundary conforming subdivision: Spatial subdivision of a solid that represents the boundary of the solid exactly or to within a given tolerance.

Boundary approximating subdivision: Spatial subdivision that represents the boundary of the solid only approximately to within the size of the subdivision.

Regular subdivision: A subdivision whose cells are congruent. Grids are regular subdivisions.

Voxel/voxelized subdivision: A regular subdivision whose cells are cubes.

Irregular subdivision: A subdivision with noncongruent cells.

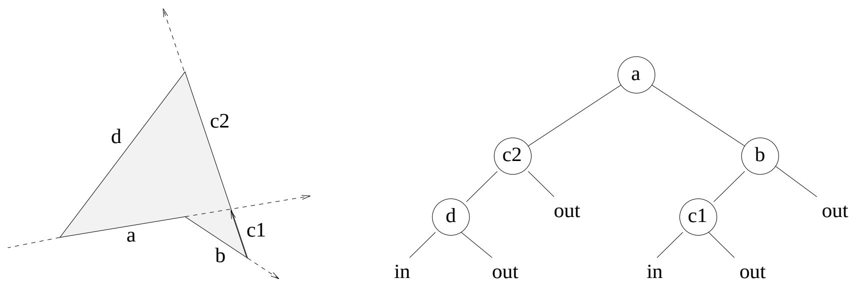
Octree: Recursive selective subdivision of a cuboid volume into eight subcuboids.

Binary space partition (BSP) tree: Recursive irregular subdivision of space, traditionally by halfplanes. See also Sections 33.8.2 and 42.5.

Spatial subdivision decomposes a solid into cells, each with a simple topological structure and often also with a simple geometric structure; [TN87, Mea82]. Subdivision representations are divided into *boundary conforming* and *boundary approximating*.

Important boundary conforming subdivision schemata are finite-element meshes and the *BSP tree*. Mesh representations are used in finite element analysis, a method for solving continuous physical problems. The mesh elements can be geometric tetrahedra, hexahedra, or other simple polyhedra, or they can be deformations of topological polyhedra so that curved boundaries can be approximated exactly. See Sections 29.4–5.

FIGURE 57.1.4
A polygon and a representing BSP tree.



Binary space partition trees are recursive subdivisions of 3-space. Each interior node of the tree separates space into two disjoint point sets. In the simplest case, the root denotes a separator plane. All points of $\mathbb{R}^3$ below or on the plane are represented by one subtree, all points above the plane are represented by the other subtree. The two point sets are recursively subdivided by halfplanes at the subtree nodes. The leaves of the tree represent cells that are labeled IN or OUT. The (half) planes are usually face planes of a polyhedron, and the union of all cells labeled IN is the polyhedron. For an example in $\mathbb{R}^2$ see Figure 57.1.4. Note that algebraic halfspaces can be used as separators, so that curved solids can be represented exactly.

Boundary approximating representations are *grids*, *voxels*, and *octrees*. In grids, space is subdivided in conformity with a coordinate system. For Cartesian coordinates, the division is into hexahedra whose sides are parallel to the coordinate planes. In cylindrical coordinate systems, the division is into concentric sectors, and so on. The grids may be regular or adaptive, and may be used to solve continuous physical problems by differencing schemes. Rectilinear grids that are geometrically deformed can be boundary-conforming. Otherwise, they approximate curved boundaries. Voxels are rectilinear grids, each hexahedron/voxel of equal size.

An octree divides a cube into eight subcubes. Each subcube may be further subdivided recursively. Cubes and their subdivision cubes are labeled white, black, or grey. A grey cube is one that has been subdivided and contains both white and black subcubes. A subcube is black if it is inside the solid to be represented, white if it is outside. In some variants, grey cells describe the contained boundary surface, leading to a boundary-conforming representation [BN90]. Quadtrees, the two-dimensional analogue of octrees, are used in many geographical information systems. See Figure 42.5.1. Some 3D printers rasterize the slices to be printed, using nonconforming spatial subdivisions.

The conversion between boundary representation and CSG can be considered a generalization of the binary space partition tree and is explored in [Hof93b, Nay90, NR95, Sha91b, SV93].

57.1.5 MEDIAL SURFACE REPRESENTATIONS

GLOSSARY

Maximal inscribed ball: Ball inscribed in a domain and not properly contained in another inscribed ball.

Medial surface transformation: Closure of the locus of centers of maximal inscribed spheres, and a function giving the minimum distance to the solid boundary. Usually called the ***MAT*** for "medial axis transformation."

Procedural representation: The solid is described by a scripting language or a notational schema that is declarative and must be evaluated deductively. Examples include: level sets of a scalar field, R-functions, subdivision surfaces.

The medial axis and medial surface can unambiguously represent two-dimensional domains and 3D solids, respectively [Blu73]. The representations are not widely used for this purpose at this time; more frequently they are used for shape recognition (see Section 54.4). However, as explained below, some meshing algorithms are based on the medial axis and the medial surface [FAR16].

The medial axis of a two-dimensional domain is defined as the closure of the locus of centers of maximal disks inscribed within the domain. A disk is maximal if no other disk properly contains it. An example is shown in Figure 57.1.5 along with some maximal disks.

The medial surface of a solid is the closure of the locus of centers of maximal inscribed spheres. When we know the radius (the limit radius in case of closure points) of the corresponding sphere for each point on the medial surface, then an

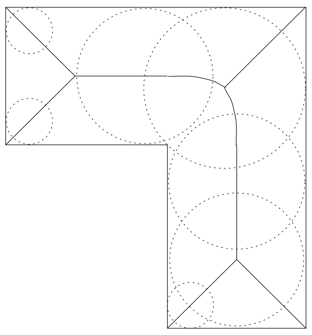


FIGURE 57.1.5
L-shaped domain and associated medial axis. Some maximal inscribed circles contributing to the medial axis are shown.

unambiguous solid representation is obtained that is sometimes called the medial axis transform (MAT). The MAT is the deformation retract of a solid and has a number of intriguing mathematical properties. For example, by enlarging the radius values by a constant, the MAT of a dilatation of the solid is obtained.

Inverting, we can construct tubular surfaces as envelope of a family of spheres centered on (a segment of) a space curve. The torus is a simple example, the Dupin cyclides a more complex one. Generalized cylinders have been used in so-called skeleton models. They also play a role in geometric dimensioning and tolerancing (GD&T).

Early on, solid modeling has investigated the MAT for the purpose of constructing shell solids (obtained by subtracting a small inset), for organizing finite element meshing algorithms, and for recognizing form features; e.g., [Hof92, SAR95, Ver94]. More recently, the role of the MAT in surface reconstruction has begun to impact solid modeling; see Chapter 35. Surface reconstruction arises in solid modeling for its application in reverse engineering where a model is to be constructed from a physical object by an automated measuring strategy.

57.1.6 CONSTRUCTIVE REPRESENTATIONS

Early on, solid models were constructed by extending a programming language with special operations for creating primitive solids, arranging them with respect to a coordinate system, and combining them. The latter used, for the most part, regularized Boolean operations. This approach created a scripting language that is effectively a constructive representation. For example, the PADL system and its variants used Fortran as script language to specify solids. CSG expressions and directives were embedded into the Fortran program. The solid was then evaluated into an internal format, usually CSG expressions plus a Brep evaluation for efficient rendering. Other modeling systems at the time used other programming languages. Since such scripting languages were based on general, procedural programming languages, the solid evaluation can be highly complex and may include any deterministic, step-by-step computation. Procedural script languages include Fortran for PADL [Bro82], Lisp for Alpha_1 [GDC94], and Scheme for SGDL [Sys01].

But the same constructive representation can also be declarative, describing requirements and properties of the complex solid model, without committing to a specific evaluation sequence. For example, instead of procedurally executing CSG operations, CSG expressions may be used as implicit representations [BB97] specifying which subsets of space (points, line segments, faces, voxels, etc.) are contained within the represented set. Specific algorithms for evaluating such subsets vary widely with applications and include polygonization, rendering, and mass property computations [Rot82, Til80]. Another example of declarative modeling is *variational geometric constraint solving* discussed in Section 57.2.5.

Constructive representations (both procedural and declarative) have experienced a resurgence recently, fueled by the need and desire to create complex solid models in architecture and in additive manufacturing. This resurgence was made possible by a rich repertoire of advanced constructions, and their compositions, including sweeps, offsets, projections, Minkowski operations, blends, free-form deformations, and interpolations [PASS95, WGG99, PPV$^+$03, MUSA15]. The semantics of these constructions is usually hidden from the user (see Section 57.3.1) and may be specified in terms of set-theoretic operations, affine and smooth transformations, polynomial constraints, R-functions [Sha07], and B-splines to name a few. As described in Section 57.3.4, these constructions are usually evaluated into Breps, mesh representations, or spatial discretizations for further processing and downstream applications.

57.1.7 MODELS OF INTERIOR

With few exceptions, solid models historically described a division of space into three point sets: the interior of the solid, its surface, and the exterior of the solid. Neither exterior nor interior were considered to have further structure. This conceptualization reflects the early focus of solid modeling on engineering applications for rigid objects with homogeneous interior such as those that were manufactured by extant manufacturing techniques of that time, including CNC machining and other unit manufacturing processes [UMP95]. The interior's homogeneous material properties were abstracted by appropriate label and material constants describing how the material behaves under various physical conditions (structural, thermal, electrical, etc.)

Industry's adaptation of solid modeling progressed quickly to more complex applications requiring models and representations of solids with *discrete,* piecewise homogeneous interiors partitioned by "interior boundaries," for example in modeling VLSI layouts and Micro-Electro-Mechanical Systems (MEMS). Rapid generalization of traditional solid modeling representations ensued, as surveyed, for example in [Tak92]. Notably, Brisson [Bri93] observed that most of such representations are essentially spatial subdivision representations based on cellular stratifications of d-dimensional manifolds. Generalizations of both boundary representations [RO89] and CSG representations [RR91] have been proposed for modeling of such cellular structures.

However, many natural and engineered objects exhibit *continuous* spatial variability of material properties. Such properties include density of bone and other natural tissues, porosity of mineral materials, volume fraction and anisotropy in functionally graded materials (FGM), and many other properties of natural and engineered materials. The continuous properties are naturally abstracted by fields: functions of spatial coordinates defined over a spatial decomposition of a solid model. Such fields may be represented in a piecewise continuous fashion over decompositions of solids [KD97] or as discrete distributions (algebraic topological chains defined over cell complexes) [PS93]. The need to model complex spatially varying fields over decomposed solids gave prominence to a subfield of *heterogeneous* *object modeling,* which conceptually is formalized in terms of fiber bundles where the spatial decomposition of the solid serves as atlas and collections of material attributes are represented by sections of the bundles [Zag97, KBDH99].

A key technical issue that dominates heterogeneous object modeling is conceptualization, representation, and control of continuous property fields, as surveyed in [KT07]. In a typical modeling scenario, values of material property of interest are specified by a user or application at certain locations in the solid considered *material features*. From these features, the field values at any location within the solid must be automatically determined, subject to gradient constraints, experimental data, as well as empirical and physical laws. As discussed in [BST04], solving such problems requires combining interpolation methods (most commonly weighted by powers of distances to material features) and numerical solvers of boundary value problems. Hence heterogeneous modeling methods may be categorized based on the methods used for interpolating property values and gradients, and based on the spatial decomposition used to evaluate the solution. Popular choices for the latter include cell complexes [KD97, AKK$^+$02], conforming and nonconforming finite element meshes [JLP$^+$99, BST04] and voxelizations of solid [DKP$^+$92, CT00].

With the advent of *additive manufacturing*, also referred to as *3D printing*, the need for modeling solids with heterogeneous interior has become obvious and growing, because heterogeneity is capable of describing composites, active structures, engineered materials, and so on. Proposed representations vary widely and are, in many cases, closely focused on specific applications, with voxel representations gaining in popularity as modern 3D printers are capable of depositing unique material at spatial resolutions of a few microns. Project Maxwell, e.g., [DKP$^+$92], included efforts to develop heterogeneous manufacturing paradigms. This, and more recent projects, proposed to employ shape and topology optimization techniques on the voxel-represented objects to be printed in order to realize functional properties via geometrical representations.

However mechanical and physical functions of solids are usually designed and controlled at a much coarser scale, suggesting that it may be beneficial to aggregate the individual voxels into local *unit cells* serving particular purpose. Such unit cells may be selected from a great variety of periodic and stochastic tessellations, or produced automatically by shape and topology optimization algorithms [Ros07]. Other benefits of using unit cells in design of materials and tissue engineering [Hol05, SSND05] include increased material and manufacturing efficiency, the ability to mimic natural and synthetic materials (for example, based on Voronoi diagrams), as well as creating and fine tuning new custom meta-materials with unusual properties, such as auxetic materials, medical scaffolds, and lightweight composite structures.

The great variety of unit cells and their combinations lead to the need for modeling and representing graded, anisotropic, periodic and stochastic lattice structures that can fill the interior of a solid. This is an active area of research with competing proposals to extend Voronoi and mesh representations [YBSH16], boundary representations [WCR05], trivariate parameterizations [Elb15], implicit periodic representations [FVP13], and stochastically generated structures [LS15], as well as their combinations [KT10, LS16]. Such lattices are not, however, boundary-conforming and additional techniques are needed to blend them with classical representations of the boundary. In medical imaging applications, organ boundaries may be approximated, resulting in a nonmanifold boundary representation whose cells represent specific organs or parts thereof. Finally, we note that cells in a lattice themselves could be further structured, leading to the notion of multi-scale solid modeling.

The extraordinary diversity of research into the technologies and applications of 3D printing presages that there will be a great variety of representations of the interior, and that calls for universal standards [KT07] may not come to pass. Nevertheless, a cell-structured interior, based on Voronoi tessellations, is a popular choice for material performance models, since it can generate realistic homogeneous and heterogeneous structures in both 2D and 3D; e.g., material performance sound absorption [JDKK07], polycrystalline materials [GLM96], thermal insulation [MY14], and thermo-elastic behavior of functionally graded materials [Bin01]. Other representations of cell structures employ medial surfaces to model organ boundaries; e.g., [NSK$^+$97], or derive smooth boundaries by approximating (polyhedral) boundaries of cells or aggregations of cells with B-splines, [KT07]. The representation of fruits in [MVH$^+$08, AVH$^+$13] uses Voronoi cells whose vertices have been snubbed to model internal passages for respiration.

Voxel-based representations are basically a sampling of the internal structure. Uniform sampling has to contend with unfavorable scaling behavior [CMP95]. This applies as well to procedural voxel representations where the voxels are constructed on the fly; e.g., [GQD13, Oxm11, DTD$^+$15]. Nonuniform samplings, on the other hand, argue for an explicit cell representation where different cells would be sampled at different densities. Khoda and Koc [KK13] propose a layered tissue representation where each slice to be printed has a preferred grain direction. Slices vary angle and density of the extruded filament.

57.2 LEVELS OF ABSTRACTION

GLOSSARY

Substratum: Basic computational primitives of a solid modeler, such as incidence tests, matrix algorithms, etc.

Algorithmic infrastructure: Major algorithms implementing conceptual operations, such as surface intersection, edge blending, etc.

Graphical user interface (GUI): Visual presentation of the functionality of the system.

Application programming interface (API): Presentation of system functionality in terms of methods and routines that can be included in user programs.

Substratum problem: Unreliability of logical decisions based on floating-point computations.

Large software systems should be structured into layers of abstraction. Doing so simplifies the implementation effort because the higher levels of abstraction can be compactly programmed in terms of the functionality of the lower levels. Thereby, the complexity of the system is reduced. A solid modeling system spans several levels of abstraction:

1. On the lowest level, there is the substratum of arithmetic and symbolic computations that are used as primitives by the algorithmic infrastructure. This level contains point and vector manipulation routines, incidence tests, and so on.

2. Next, there is an intermediate level comprising the algorithmic infrastructure. This level implements the conceptual operations available in the user interface, as well as a wide range of auxiliary tools needed by these operations. There is often an application programming interface (API) available with which programs can be written that use the algorithmic infrastructure of the modeling system.

3. Designing families of shapes is at a higher level yet, through the use of features and constraints. Features and constraints are defined using feature parameters and geometric and dimensional constraints. An instance design so prepared can be changed to another instance design by changing parameters and/or constraints — which triggers a re-evaluation of the design by the system in accordance with these changes. This style of shape design can be conceptualized as a generic design of a family of specific designs or instance designs.

Ideally, the levels of abstraction should be kept separate, with the higher levels leveraging the functionality of the lower levels. However, this separation is fundamentally limited by the interaction of numeric and symbolic computation. As it were, the substratum is not fully dependable.

More than that, there is no complete semantic characterization of the generic design process at the third level, since the re-evaluation of the changed parameters and constraints requires an understanding of how specific shape elements (vertices, edges, faces) correspond, between different family members, and what it means when such a correspondence does not exist or has ambiguities.

The hierarchy of tools and concepts can be used in a variety of ways, creating different views of the capabilities:

1. Traditionally, a graphical user interface (GUI) presents to the user a view of the functional capabilities of the system. Interaction with the GUI exercises these functions, for instance, for solid design. Tools for editing and archiving solids are included.

2. Another way to access is by an application programming interface (API). An API exposes functionality of the system much as a software library would. Most common is access to algorithmic infrastructure, but APIs can also expose GUI functionality, giving users the tools to customize the GUI for their purposes.

3. A third style of accessing system capabilities is through a query interface. Here, a client–server architecture can be constructed that allows various systems to collaborate using a message-passing paradigm.

57.2.1 THE SUBSTRATUM

The substratum consists of many low-level computations and tests; for example, matrix computations, simple incidence tests, and computations for ordering points along a simple curve in space. Ideally, these operations create an abstract machine whose functionality simplifies the algorithms at the intermediate level of abstraction. But it turns out that this abstract machine is unreliable in a subtle way when implemented using floating-point arithmetic. Exact arithmetic would remedy this unreliability, but is held by many to be unacceptably inefficient when dealing with solids that have curved boundaries. See Section 45.4. Problems include input accuracy as well as tolerances used in deciding incidence and other numerical predicates.

A root problem is the dichotomy between approximate numerical floating-point computation and exact topological incidence decisions based on it; [Hof89][Chap. 4]. It is therefore no surprise that systems built on a substratum that only does exact computation avoid this unreliability and do not experience consequent failures. However, exact arithmetic is not efficient unless geometric coverage is very limited. An interesting variant was proposed in [For97], where for polyhedral solids the needed predicates are evaluated exactly only when the floating-point computation cannot be guaranteed to be correct. Although the problem was observed early-on, its analysis appeared only in the late 1980s; see [Hof01a]. The discovery motivated work on the use of exact arithmetic in polyhedral modeling [SI89, For97], as well as for curved surfaces [KKM99a, KKM99b].

Another unavoidable problem arises in systems that use parametric curves and surfaces as shape elements. Here, the representation of two parametric surfaces is usually recorded as a parametric curve. Unfortunately, in general the intersection of two parametric surfaces is not a parametric curve and must be approximated; e.g., [Hos92, PM02]. It is then difficult to interpret such shape constructs correctly.

Binary arithmetic of fixed precision cannot always encode decimal input of fixed precision. An example is 1/10 which requires one decimal but is a periodic binary fraction. For more detail on this subject see Chapter 45.

Work in symbolic algebraic computation (Chapter 37) has foundational importance, for instance in regard to converting between surface representations. Some of the applications of symbolic computation are explored in [BCK88, Cho88, Hof90].

57.2.2 ALGORITHMIC INFRASTRUCTURE

Algorithmic infrastructure is a prominent research subject in solid modeling. Among the many questions addressed is the development of efficient and robust algorithms for carrying out the geometric computations that arise in solid modeling. The problems include point/curve/surface-solid classification [Til80], computing the intersection of two solids, determining the intersection of two surfaces [PM02], interpolating smooth surfaces to eliminate sharp edges on solids, and many more.

Specific algorithms often require specific data structures and representations. Together, algorithms and representations develop by co-evolution that occasionally is disrupted. Past disruptions include the design style of Pro/Engineer that compelled other systems to adopt a similar design vocabulary. It remains to be seen how technologies such as 3D printing and concepts such as query-based system architectures impact the development of the algorithmic infrastructure and its representations. The examples that are described primarily rely on Brep as system representation.

Academic work considers structuring *application programming interfaces* (APIs) that encapsulate the functional capabilities of solid modelers so they can be used in other programs; [ABC$^+$00]. Such APIs play a prominent role in applications because they allow building on existing software functionality and constructing different abstraction hierarchies than the one implemented by a full-service solid modeling system. The work attempts to give a system-independent specification of basic API functionality for solid modeling.

An important consideration when devising infrastructure is that the algorithms are often used by other programs, whether or not there is an API. Therefore, they must be ultra-reliable and in most cases must not require user intervention for exceptional situations. This requirement becomes crucial for more recent work seeking to integrate multiple systems, each addressing different aspects of an overarching application. For instance, one subsystem would be used to design an artifact X, a second analyzes thermal properties of X, a third subsystem analyzes tolerance requirement for given manufacturing processes, and so on. Here, the communication between subsystems raises critical issues, see Section 57.4. Restricting to shape design and manipulations, some of the researched design operations, their implementation, and their presentation in the GUI include the following.

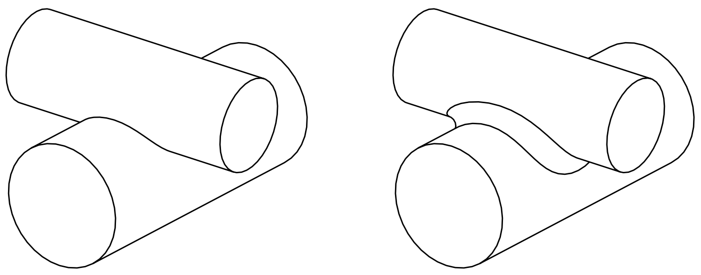
Surface intersection. Given two bounded areas of two surfaces, determine all intersection curve components. All components of the intersection of the two patches must be correctly identified, including isolated points and singularities. Since this computation is done in $\mathbb{R}^3$, classical algebraic geometry is of limited help. A solution must negotiate well the unreliability of the substratum; see also [PM02].

Offsetting. Given a surface, its ***offset*** is the set of all points that have fixed minimum distance from the surface. Offsets can have self-intersections that must be culled. There is a technical relationship between offsetting and forming the MAT. Namely, when offsetting a curve or surface by a fixed distance, the self-intersections must lie on the medial axis. Offsetting can be used to determine certain blending surfaces, and is also useful for *shelling* that creates thin-walled solids. Offsetting and shelling are global solid operations, considered in [BW89, For95, PS95, RSB96].

Blending. Given two intersecting surfaces, a third surface is interpolated between them to smooth the intersection edge. A simple example is shown in Figure 57.2.1. A locally convex blend surface is often called a *round*, and a locally concave one a *fillet*. The blend surface in Figure 57.2.1 is a fillet.

Blending has been considered almost since the beginning of solid modeling, and some intuitive and interesting techniques have been developed over the years. For example, consider blending two primary surfaces f and g. Roll a ball of fixed radius r along the intersection such that it maintains contact with both f and g. Then the surface of the volume swept by the ball can be used as a blending surface, suitably trimmed. Note that the center of the ball lies on the intersection of the offsets, by

FIGURE 57.2.1
Left: two cylinders intersecting in a closed edge. Right: edge blended with a constant-radius, rolling-ball blend; the bounding curves of the blend are shown.



r, of both f and g. In more complicated schemes the radius of the ball is varied along the intersection.

Blends can interfere: Figure 57.2.2 shows the problem of overlapping blends. The fillet and round constructed separately do not meet in the region of overlap. Possible resolutions can be proposed but are difficult to systematize. Multiple surfaces have to be composed and shown to have appropriate global characteristics, primary surfaces have to be cut back appropriately.

When the primary surfaces meet at a vertex tangentially, blending surfaces must "dissipate." Figure 57.2.3 shows several methods to dissipate round and fillet at the end vertices. The examples are from [Bra97] and illustrate the dimensions of the global problem.

Deformations. Given a solid body, deform it locally or globally. The deformation could be required to obey constraints such as preserving volume or optimizing physical constraints. For example, we could deform the basic shape of a ship hull to minimize drag in fluids of various viscosities.

Shelling. Given a solid, hollow out the volume so that a thin-wall solid shape remains whose outer surface is part of the boundary of the input solid. The wall thickness is a parameter of the operation. Variations include designating parts of the solid surface as "open." For instance, taking a solid cylinder and designating

FIGURE 57.2.2

Global blend interference [Bra97]: The round of the front edge overlaps with the fillet of the cylinder edge on top (left). Without further action, the two blends do not connect, leaving a gap in the surface. The solution shown in the middle modifies the front round. Other possibilities include modifying the fillet or inserting a separate blend in the overlap region (right).

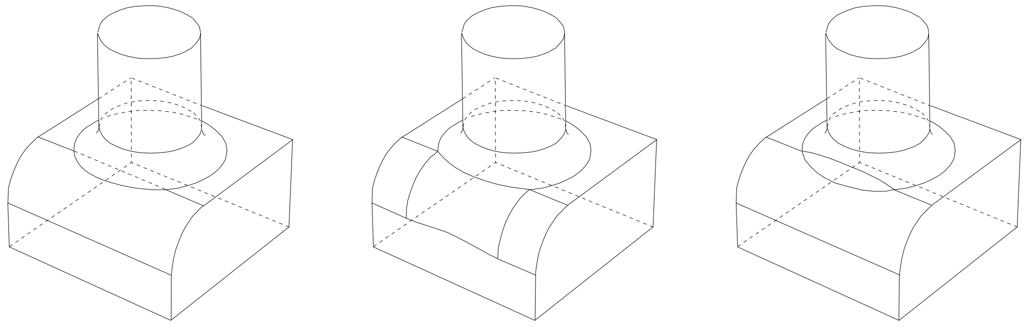
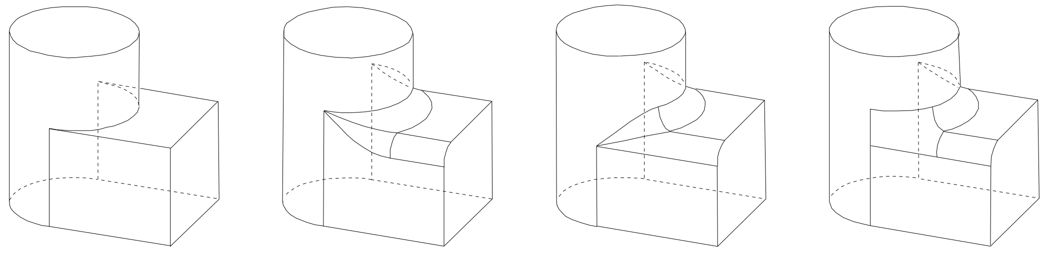


FIGURE 57.2.3

Global blend interference [Bra97]: At ending vertex, the round and the fillet must be merged into a compatible structure. Several solutions are illustrated.



both flat end faces as open the operation creates a hollow tube of the same outside diameter. Conceptually, the operation subtracts an inset of the solid, obtained by shrinking the original solid, an offset operation.

57.2.3 FEATURES AND CONSTRAINTS

GLOSSARY

Form feature: Any stereotypical shape detail that has application significance.

Geometric constraint: Prescribed distance, angle, collinearity, concentricity, etc.

Generic design: Solid design with constraints and parameters without regard to specific values.

Design instance: Resulting solid after substituting specific values for parameters and constraints.

Parametric constraint solving: Solving a system of nonlinear equations, arising in geometric constraint solving, that has a fixed triangular structure.

Variational constraint solving: Solving a system of nonlinear simultaneous equations arising in geometric constraint solving.

In solid modeling, two design paradigms have become standard for manufacturing applications, *feature-based design* and *constraint-based design.* The paradigms expose a need to reconsider solid representations at a different level of abstraction.

The representations reviewed before are for individual, specific solids. However, we need to represent entire *classes* of solids, comprising a generic design. Roughly speaking, solids in a class are built structurally in the same way, from complex shape primitives, and are instantiated subject to constraints that interrelate specific shape elements and parameters. How these families should be defined precisely, how each generic design should be represented, and how designs should be edited are all important research issues of considerable depth.

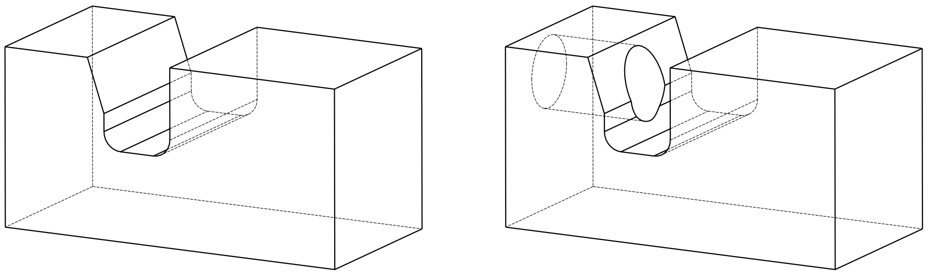
Neither features nor constraints are new concepts, so there is a sizable literature on both. The confluence of the two issues in solid modeling systems, however, is new and raises a number of questions that have only more recently been articulated and addressed. [SHL92, KRU94] discuss feature work. Constraints are the subject of [BFH$^+$95, HV94, Kra92]. The confluence of the two strands and some of the implications are discussed in [HJ92]. Some of the technical issues that must be addressed are explained in [Hof93a, CH95a], and there is more work emerging on this subject. In particular, Shapiro and Raghothama propose several criteria for defining a family of solids; [RS02a, RS98].

57.2.4 FEATURE-BASED DESIGN

Feature-based design is usually understood to mean designing with shape elements such as slots, holes, pockets, etc., that have significance to manufacturing applications relating to function, manufacturing process, performance, cost, and so on. Focusing on shape primarily, we can conceptualize solid design in terms of three classes of features: generative, modifying, and referencing features; [CCH94, CH95a]. A feature is added to an existing design using attachment attributes and placement conditions. Subsequent editing may change both types of attachment information.

As an example, consider the solid shown to the right in Figure 57.2.4. A hole was added to the design on the left, and this could be specified by giving the diameter of the hole, placing its cross section, a circle, on the side face, and requiring that the hole extend to the next face. Should the slot at which the hole ends be moved or altered by subsequent editing, then the hole would automatically be adjusted to the required extent.

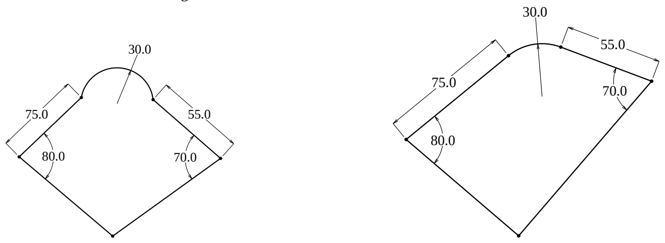
FIGURE 57.2.4
Left: Solid block with a profiled slot. Right: After adding a hole with the attribute "through next face," an edited solid is obtained. If the slot is moved later, the hole will adjust automatically.



57.2.5 CONSTRAINT-BASED DESIGN

Constraint-based design refers to specifying shape with the help of constraints, when placing features or when defining shape parameters [Owe91, BFH$^+$95]. For instance, assume that we are to design a cross section for use in specifying a solid of revolution. A rough topological sketch is prepared (Figure 57.2.5, left), annotated with constraints, and instantiated to a sketch that satisfies the constraints exactly (Figure 57.2.5, right). Auxiliary geometric structures can be added, such as an axis of rotation. There is an extensive literature on constraint solving, from a variety of perspectives. Surveys with a solid modeling perspective include, e.g., [HJA05, FHJA16].

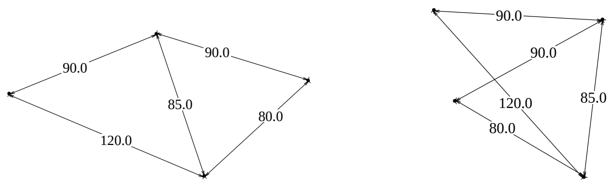
FIGURE 57.2.5
Geometric constraint solving. Input to the constraint solver shown on the left. Here, the arc should be tangent to the adjacent segments, and the two other segments should be perpendicular. Output of the constraint solver shown on the right.



Most solid modeling systems use both features and constraints in the design interface. The constraints on cross sections and other two-dimensional structures are usually unordered, but the constraints on 3D geometry may restrict to a fixed sequence. Solving systems of unordered constraints is also referred to as *variational constraint solving*. Mathematically, it is equivalent to solving a system of nonlinear simultaneous equations. Solving constraints in a fixed sequence is also known as *parametric constraint solving*. The latter is equivalent to solving a system of nonlinear equations that has a triangular structure where each equation introduces a new variable.

A *well-constrained* geometric constraint problem corresponds naturally to a system of nonlinear algebraic equations with a finite set of solutions. In general, there will be several solutions of a single, well-constrained geometric problem. An example is shown in Figure 57.2.6. This raises the question of exactly how a constraint solver should select one of those solutions efficiently, and why.

FIGURE 57.2.6
The well-constrained geometric problem of placing 4 *points by* 5 *distances has two distinct solutions.*



From symbolic computation we know that there are algorithms to convert a system of nonlinear equations that is not triangular, into an equivalent, triangular system. The distinction between parametric and variational constraint solving is therefore artificial in theory. However, full-scale triangularization of systems of nonlinear equations is not tractable in many cases, so the distinction is relevant in practice.

Many constraint solvers proceed in two major phases; see [Owe91, BFH$^+$95, HLS01b, HLS01a].

In the first phase, the constraint problem is abstracted into a graph whose vertices represent the geometric primitives to be placed, and whose edges represent the given constraints. Vertices are labeled by the number of independent coordinates needed to position the represented primitive in a coordinate system. Edges are labeled by the number of independent equations needed to express the represented constraint. The constraint solver analyzes this constraint graph seeking to determine a set of small subproblems that can be solved separately.

In the second phase, the solver processes the subproblems, found by the first phase, solving the associated equations and combining their solutions, thereby determining the solution of the original constraint problem.

In principle, the graph patterns for constraint problems in 2D can be of arbitrary complexity. However, when restricting to problems involving only points, lines and circles, a small set of patterns suffices for the first phase. They can be found with straightforward graph algorithms. Moreover, a small set of simple equations arises in the second phase: when the radius of the circles is given, only univariate quadratic equations must be solved in the second phase. [Owe91].

Spatial constraint solving is very much more demanding than planar constraint solving. In particular, the subsystems of equations that must be solved in the second solver phase ramp up steeply. Spatial constraint problems appear in two different species in the literature.

1. The geometric elements connected by constraints are points, lines and planes in 3-space. The constraints are distance, angle, perpendicularity, parallel, etc. See, e.g., [HV94, GHY04].

2. The geometric elements connected by constraints are as before but they are elements of a solid. The constraints are as before. See, e.g., [JS13]

From an application perspective the second variety appears to be of greater significance. See also Section 57.4.

57.3 ARCHITECTURES AND SYSTEMS

Today, solid modeling technology is at the core of most scientific, engineering, and consumer applications that require unambiguous representation of shape information. The methods for utilizing the rich infrastructure of solid modeling may be divided into three broad categories:

1. application-specific user interfaces that allow maximum leveraging of the constraint-based and feature-based layers;

2. application programming interfaces (API) that support packaging of selective solid modeling capabilities and components, much like any other software library; and

3. plug-compatible functional components to support the current trend towards modularized and distributed system architectures.

Both efficacy and limitations of these methods are often determined by their ability to use more than one representation at the same time and convert between them if necessary.

57.3.1 USER INTERFACES

Ultimately, the functional capabilities of a solid modeling system have to be presented to a user, typically through a *graphical user interface* (GUI). It would be a mistake to dismiss GUI design as a simple exercise. If the GUI merely presents the functionality of the infrastructure literally, an opportunity for operational leveraging has been lost. Instead, the GUI should conceptualize the functionalities an application needs. As in programming language design, this conceptual view can be convenient or inconvenient for a particular application. Research on GUIs therefore is largely done with a particular application area in mind.

User interfaces are intimately tied to specific solid modeling representations supported at an appropriate level of abstraction. For example, in mechanical engineering product design, an important aspect of the GUI might be to allow the user to specify the shape conveniently and precisely. Early solid modeling systems relied heavily on parametric constructive and procedural representations to express the intended shape which were usually converted to other representations for visualization and downstream applications such finite element meshing and manufacturing process planning [RV83]. Common target representations included boundary representations and various spatial discretizations.

As solid modeling systems gained acceptance in industry, user interfaces began to require direct references to boundary representations to support sketching, reference datums, and direct surface manipulations needed to support advanced automotive and aerospace applications. With time, advanced interfaces emerged to support mechanical design in terms of constraints (on distances, radii, angles, and other dimensions and parameters) and application-specific features (sheet metal, NC machining, structural, molding, etc.) that would combine elements of constructive and boundary representations [CH95b]. To support such user interfaces, the algorithmic infrastructure must be capable of supporting and maintaining consistency of both types of representations within a single system [RS00].

In interfaces for virtual environment definition and navigation, on the other hand, approximate constraints and direct manipulation interfaces would be better. More recently, widespread use of solid modeling in architecture, industrial design, 3D printing, and many consumer applications witnessed renewed popularity of procedural and declarative specifications that are presented to users as functional programming compilers [PPV$^+$03], visual programming languages [M$^+$10], cloud-hosted web browser interfaces, and open source systems [KW14]. Increased popularity of meshes and voxels led to interfaces designed specifically for the creation and editing of such representations and algorithms for many emerging applications, including scanning and shape reconstruction, geometric signal processing, and 3D printing [CCC$^+$08, SS10, DTD$^+$15].

57.3.2 APPLICATION PROGRAMMING INTERFACES (API)

The widespread demand for solid modeling technology in many different applications and industries led to componentization and packaging of different layers into standalone libraries that are accessible via Application Programming Interfaces (APIs). The most popular packages are solid modeling kernels that comprise the substratum and algorithmic layers for boundary representations supporting low-degree polynomial and spline surfaces, notably Parasolid, ACIS, openNURBS, Open Cascade and others. The constraint management layer is usually available as a separate component, for example from D-Cubed [Hof01b]. Modeling kernels based on other representation schemes are also available, but are less popular, e.g., PADL-2 (based on CSG and quadric surface Breps) and Hyperfun (based on declarative implicit functional representations). Solid modeling capabilities are also being exposed in APIs of more general purpose systems that aim at broader computational arenas, notably in computational geometry systems such as CGAL [FP09], visualization toolkits (VTK), and mesh processing systems such as Meshlab, to name a few.

While APIs have been the primary means for integrating solid modeling technology in thousands of applications, it is important to note that APIs are usually fine tuned to the representation and algorithmic infrastructure they encapsulate. As such, they do not cleanly encapsulate the internal implementation details: These APIs and are not interchangeable in that replacing one solid modeling component with another is usually difficult or impossible). Moreover, APIs are not necessarily interoperable when they are based on incompatible mathematical models and representations. [ABC$^+$00] describes a significant effort to design a representation-independent API for solid modeling. The approach is based on the idea that most representations, including CSG, boundary, and spatial discretizations, may be put into a canonical stratified form similar to a cell complex where individual strata are (primitive) sign-invariant k-manifolds in the decomposition of shape [Sha91b]. The results of the effort remained largely academic because such stratifications are nontrivial to compute, lead to excessive fragmentation of geometric information, and are rarely used in practice.

57.3.3 MODEL EXCHANGE AND PLUG-COMPATIBILITY

Given the great diversity of modeling representation schemes, standalone systems, and APIs, the need for interchanging, sharing, and combining solid models from different sources has become greater than ever. To meet this need, there have been, and continue to be, many efforts to find standards for exchanging shape models. These efforts began already in the early days of solid modeling research; [RV82, Fre96]. Abstractly, the problem may be stated as follows:

> Let M be a solid model authored in system C. In order to use M in another (modeling) system C', M has to be converted into an "equivalent" model M' in the native representation of system C'. The model M' would then be used as if it had been authored by C'.

This informal formulation masks the fundamental challenge underlying the task: the notion of equivalence assumes the existence of a rigorous model semantics that is common to both systems and is adhered to when authoring and/or converting the two models. Unfortunately, this is often not the case. Conceptually, there are three plausible approaches to achieve flawless model interoperability and exchange as discussed below: (1) exchange of generic models, (2) exchange of representations of model instances, and (3) query-supported interchangeability of models.

The simplest and the most elegant method to solve the above problem is to treat M and M' as instances of a common generic model corresponding to some specific valuation of parameters and dimensional constraints. The generic model may then be construed as a procedure for generating model M in any system C which is capable of evaluating this procedure. This approach has been advocated for generic Erep models [HJ92], and as a method for interoperability between major commercial CAD systems [SR04]. Because generic models are not fully instantiated, they tend to be compact and contain mostly symbolic information, thereby supporting efficient and robust solutions to the interchange problem. The problem with this approach is the lack of standard semantics and lack of consistent support for generic models in different systems [Sha91a, KPIS08]. Thus, a "slot" feature may well be instantiated differently in systems C and C'. Initial efforts towards standardization of two-dimensional procedural definitions is reported in [KMHP11]. Formal semantics is known for many constructive representations, such as CSG, but they can express only a small subset of widely used generic models.

The second approach attempts to solve the problem directly by exchanging instantiated and fully evaluated representations of M and M'. The approach assumes that the two representations can in fact be converted into each other, either exactly or approximately, but well enough to be considered equivalent in some sense. The representations are imported (exported) in representation-specific file format. One way to avoid a quadratic number of such convertors is to convert M into a neutral format as model M_N that can also be read by C' and that C' can convert to M'. By the far the most popular neutral exchange format is STEP [Kem99] which covers many flavors of boundary representations and some basic constructive representations. Unfortunately, these standardization and data translation efforts continue to have limited success. A percentage of exchanges result in models M' that must be manually repaired subsequently.

The difficulty realizing the first two scenarios is due in part to mathematical facts. It is also in part due to the fact that the systems C and C' may interpret the same data items differently. Consider a system C that models curved surfaces as trimmed nonuniform, rational B-splines (NURBs) and can evaluate them to an accuracy of 10^{-9}, and a system C' that models faceted surfaces only and can evaluate them to an accuracy of 10^{-6}. The evaluation accuracy could be absolute, in system C, but relative in system C'. Moreover, the trim curves of NURB faces in system C can only be approximated in the NURB framework, necessitating algorithms that make sophisticated interpretations when evaluating the model M, based on assumptions which may not be represented in M_N. It is natural then to look for a notion by which a model M, authored in C, is *almost equal* to a model M' authored in C', based on some specific metric. But, again, the metrics concepts fail usually for lack of transitivity.

The third approach to the exchange problem is inspired by the notion of interchangeable parts in mechanical assembly [HSS14] and works as follows. Instead of ascertaining that two part models are *almost the same*, a relation that is not transitive, one ascertains that two part models M and M' are each within allowable tolerance of the reference model M_0. This induces a relation between M and M' that is transitive. Note that the models are never compared to each other, and the common formal semantics is embodied into the measurement procedure that is performed according to the *Geometric Dimensioning and Tolerancing* (GD&T) standard [ASME09]. An analogous approach for solid modeling suggests that different systems C and C' should exchange not models but formally defined and standardized *queries*. The closeness of a given model M to a reference model M_0 is established by queries of M and M_0. Queries are simple geometric predicates. They request the authoring system C to query a model M, thereby avoiding the translation problems characterized above. Examples of standard queries include point-membership classification (PMC) where a query point p is classified as being *in* the interior of the modeled solid, *on* the surface, or *out*side of the solid; distance from a point to the boundary of a solid. A line-solid classification intersects a query line with a solid, and segments the line accordingly. Such classification queries are analogous to testing a physical part with a coordinate measuring machine. In computer science terms, a solid model M is encapsulated as an object model with a query-based interface that, in contrast to APIs, does not expose the internal details of representation or implementations in the authoring system C. The authoring system interprets the solid model, M, thereby actualizing all inferences and particular interpretations of M, whether explicit or implicit in the model.

Queries can be used to make CAD systems interoperate, as well as exchange partial or complete model information [HSS14]. They have also been used to integrate solid modeling systems with engineering analysis systems [FST06, FST11].

57.3.4 REPRESENTATION CONVERSIONS

Representation conversions have influenced the architecture and evolution of solid modeling systems from early days in several important ways [RV83]. They are needed when interchanging solid data between systems that use different native representations. They are also used when certain operations require them, either within the same system or for interoperability between different systems. For instance, the conversion from CSG to Brep supports rendering the solid shapes, using algorithms that are optimized for rendering triangles. Rendering CSG-represented solids directly, by ray casting for example, would be much less efficient. Meshing is another common example where Brep is converted into a finite element mesh used for engineering analysis. The problem may be formulated as follows:

> A solid model M is given in a representation X. We seek to represent M in another representation Y, either exactly or approximately.

The statement assumes that M is a well-defined mathematical model, or at least that it is possible to check if two representations X and Y represent the same solid.

Conversions may be broadly divided into two categories: deterministic *evaluations* when the source representation X contains more information than the target information Y, and nondeterministic *comprehensions* when additional information in Y has to be synthesized based on additional assumptions or goals [HSS11].

Computational properties of evaluation procedures are widely researched and are mostly well understood. In abstract terms, suppose that the target representation Y can be described as a finite collection of primitive geometric elements $\{y\}$. Then any evaluation procedure implements a *generate and test* paradigm [Sha97]:

1. Generate a sufficient set of candidates y;
2. test each y against the source representation X;
3. assemble the target representation Y from those elements y that passed the test.

Details vary depending on the mathematical properties of the two representations and the efficiency of the first two steps. We discuss evaluation conversions first.

Boundary evaluation is an evaluation conversion that generates the Brep Y from a constructive representation X. It is used in most commercial systems to evaluate generic and feature-based representations. The procedure is a generalization of the well-known CSG to Brep conversion [RV85], and amounts to generating candidate faces and edges from the primitives in CSG, classifying them with respect to the constructive definition, and merging those passing the test into an optimized boundary data structure. Brep solids are usually converted into polygonal (most commonly, triangle) mesh representations that are required in many applications, including rendering and 3D printing. These polygonizations can also be produced directly from constructive representations, a common approach when dealing with implicit representations [ALJ$^+$15]. Spatial discretizations are commonly computed from constructive, boundary, and mesh representations; generation of a nonconforming regular discretization is straightforward with voxelization or Delaunay tetrahedralization [Si15].

Constructing conforming discretizations is more challenging, particularly when conversion is done for finite element analysis or other numerical treatments of continuum problems. In that case, the problem is not a geometric problem alone: the quality of the discretization must also be judged by nongeometric criteria that derive from the nature of the physical problem and the numerical algorithms used to solve it. This is an active area of research with many approaches based on octree subdivision, on Delaunay triangulation, and on MAT computations.

Algorithms for constructing the MAT from constructive and boundary representations involve generating a sufficient set of bisectors (curves or surfaces) for primitives in the source representation X, trimming them against each other, and assembling the pieces into a complete medial axis [CKM04, MCD11]. Because simple boundary geometry elements can produce very complicated curve segments and surface patches in the MAT, approximation approaches are favored in practice. Some are based on geometric principles, some on a Delaunay triangulation of an approximated boundary, and some on a grid subdivision of ambient space. The conversion from MAT to Brep has been addressed by Vermeer [Ver94] and later by Amenta [ACK01]. Note that a polyhedral MAT produces a solid boundary that can contain spherical, conical, and cylindrical elements.

The *comprehension* conversion procedures are similar to the *evaluation* conversions, except that the step of generating a sufficient set of primitives $\{y\}$, needed to construct the target representation Y, usually involves application-specific heuristics. For example, Breps may be constructed from either spatial discretizations or from meshes by fitting surfaces, but the type of the surfaces to be fitted (splines, algebraic, quadratic, etc.) must be assumed. Constructing generic feature-based representations from Breps is a further generalization of this class of conversions that requires heuristically matching procedural and parametric primitives to sets of boundary surfaces [HPR00, NMLS15].

Constructing a CSG representation from a Brep is a classical problem. For Breps with polygonal faces the problem amounts to constructing a BSP tree. The solution in [SV93] for Brep faces that include natural quadrics (sphere, cone and cylinder) reveals that the core problem for curved boundary faces concerns the first step of generating sufficient candidates $\{y\}$. The set of candidates $\{y\}$, here cells, requires finding additional (nonunique) separating primitives that do not contribute to the boundary and hence are not explicitly present in the boundary representation. This problem is mostly solved for second-degree surfaces, but for higher degree surfaces it is open.

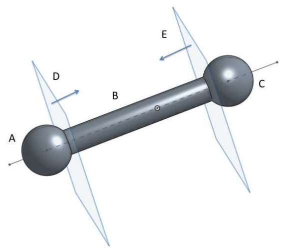


FIGURE 57.3.1
A Brep solid that cannot be converted to a CSG expression when restricting to the three half spaces A, B, and C, and their complements, that contribute nonempty boundary face areas.

Figure 57.3.1 shows an example of conversion for a simple solid: The solid's boundary is a union of three faces: two spherical and one cylindrical. However the solid cannot be represented as a Boolean set combination of the three corresponding halfspaces A, B, and C, and their complements A', B' and C', as shown in the figure. The intersection term $A'BC'$ contains both points inside and points outside the solid. A CSG representation becomes possible when using two additional (nonunique) separating halfspaces D and E. A canonical CSG representation of the solid, as a union of intersection terms, is

$$\begin{aligned}&ABC'D'E + AB'C'D'E + A'BCDE' + A'B'CDE'+\\&ABC'DE + A'BC'DE + A'BCDE + A'BC'DE,\end{aligned}$$

where regularized $\cap$ and $\cup$ operations are represented by multiplication and addition respectively, and X' stands for regularized complement of X. An equivalent optimized CSG expression for the same solid is

$$A + CDE + B.$$

57.4 OPEN PROBLEMS

Most major problems in solid modeling contain a distorting conceptualization aspect. That is, a precise, technical formulation of the problem commits to a specific conceptualization of the larger context that may be contentious. For example, consider the following technical problem:

Given an implicit algebraic surface S and a distance d, find the "offset" of S by d.

Assuming a precise definition of offset, and a restriction to irreducible algebraic surfaces S, the problem statement ignores the fact that a solid model is not bounded by a single, implicit surface, and that implicit surfaces of high algebraic degree may cause severe computational problems when used in a solid modeler.

CONSTRAINT SOLVING

Geometric constraint solvers trade efficiency for generality. Some very interesting techniques have been developed for planar problems that are fast but not very general. Nevertheless, they are useful in solid modeling applications. They could be extended in various ways without substantially impacting on efficiency. Such extensions, for constraint solving in the plane, include the incorporation of parametric curve segments as geometric elements, more general constraint configurations, as well as relations among distances and angles. The bulk of the work needed is a robust equation solver for the second phase.

Formulating explicit constraints on more general representations of polynomial and rational curves and surfaces is a challenging problem. Recent proposals call to replace explicit constraints with "black box" constraints that are defined implicitly by queries they support [GFM$^+$16].

Recall from Section 57.2.5 the definition of the constraint graph. In a sense, the graph captures structure: by varying the valuation of the dimensional constraints, angle and distance, we obtain a family of constraint problems of a particular structure. Several questions come to mind:

1. Is the family of problems generically well-constrained? That is, does there exist a valuation of the dimensional constraints for which the resulting constraint problem is well-defined. The same question for over- and under-constrained families; see Chapter 61.

2. Constraint valuations for which a problem is well-constrained represent a manifold in a space of high dimension. What is the geometry of that manifold? The same question is of interest for under-constrained families since they include linkages; see Chapter 9.

3. Restricting to points and Euclidean distance, we obtain linkages whose properties have been studied in mathematics; e.g., [Max64, Hen08]. Characterize graphs that are rigid; see Chapter 63.

For points and lines in the plane there is a characterization of generically well-defined graphs [Lam70]. When circles are allowed as well, the result of [Lam70] no longer holds. For spatial constraint problems, with points, lines and planes as primitives, the problem is open. However, when the geometric primitives are drawn from the Brep of a set of solids, then at least a partial characterization is known [JS13].

Spatial geometric constraint solving in particular poses many other open problems, both variational and parametric. The set of parametric/sequential problems for points and planes is straightforward. Minimal variational problems for points and planes are understood; e.g., [Ver94]. One of the sequential construction problems for lines is to find all common tangents of four spheres in space. Equivalently, construct a line at prescribed distances from four fixed points. There are up to 12 common tangents determined by an equation system of degree 24 [HY00].

Constraint problems involving points, planes and lines have been investigated by Gao. Here the number of minimal configurations involving only a few primitives is large: [GHY04] shows that the minimal configurations with four, five and six lines number, respectively, 1, 12, and 494.

FEATURES

Manufacturing applications need cogent definitions of features to accelerate the design process. Such definitions ought to be in terms of generic mechanisms of form and of function, which are largely missing. Most feature definitions rely on specific representations and are not interchangeable. Also needed are mapping algorithms interrelating different feature schemata.

A set of features, say those conceptualizing machining a shape from stock, represents a particular *view* of the shape. In manufacturing applications there are many views, including machining, tolerancing, design view, etc. The problem of altering a design in one view with an automatic update of the other views is challenging. Some approaches have been based on subdividing the shape by superimposing all feature boundaries, and then tracking how the subdivision is affected by changes to one of the features. Abstractly, this problem is a generalization of a representation conversion problem where multiple representations of the same shape must be maintained simultaneously [RS00]. Maintaining multiple views for additively manufactured material structures is much more difficult because each view may not only require a different representation but may also be based on a distinct mathematical model and geometric shape [RRSS16].

SEMANTICS OF CONSTRAINT-BASED DESIGN

A solid shape design in terms of constraints can be changed simply by changing constraint values. To date, all such changes have been specified in terms of the procedures and algorithms that effect the change. What is needed is an abstract definition of shape change under such constraint changes to obtain a semantic definition of generic design and constraint-based editing. Such a definition must be visually intuitive.

Cellular homotopy has been proposed as a basic principle that formalizes the notion of intuitive update for both constructive and boundary representations [RS02b, RS98], but practical applications of this principle require a solution to the persistent naming problem [MP02].

MODEL TOLERANCES AND RECTIFICATION

Because of the substratum problem, Brep data structures can be invalid in the sense that the geometric description does not agree fully with the topological description. For instance, there may be small cracks between adjacent faces, the edge between two adjacent faces may not be where the curve description would place it, and so on. This has motivated work to “heal” the defective surface by closing cracks, eliminating overlaps, and so on. Some approaches sew up cracks with smaller faces, and in the case of polyhedra with triangles. Optimal healing is known to be NP-hard.

An intuitive idea is to assign tolerances to faces, edges and vertices, effectively thickening them so that the surface closes up [Jac95]. The difficulty is to work out what happens when nonadjacent faces merge into adjacent ones. The natural geometric enlargement, moreover, creates mathematically difficult surfaces; for instance, the offset surface of an ellipsoid increases the algebraic degree by a factor of 4. So, an interval-based approach has been proposed in which there is no closed-form description of the enlarged geometric elements [SSP01]. More formally, model rectification requires establishing validity of the enlarged sets[QS06] and its consistency with the intended exact representation [Sha08, SPB04].

INTEROPERABILITY

The broad term of interoperability subsumes a number of problems that have not been fully resolved, including model interchangeability and plug compatibility, system interoperation, and system integration. While progress has been made in all of these areas, interoperability challenges represent a major technological and economic barrier to wider adaption of solid modeling [HSS11]. A key open issue is lack of common formal semantics model that spans the broad spectrum of solid modeling models, representations, and systems.

The departure from a data-centric to a query-based approach as basis of system integration and interoperation is recent and appear promising [HSS14]. More work is needed to better understand the pros and cons. Technical issues that are clear include completeness and soundness of various query systems, how to structure queries appropriately for applications, granularity of the communication substrate, latency, and efficiency, to name a few. There are also nontechnical issues: the total number of queries needed to integrate foreign models may reveal more about the model creation and use than either system would be willing to share. These considerations should become clearer as the approach is tested over time and investigated scenarios are found to be compelling or otherwise.

MODELS OF INTERNAL STRUCTURE

Representation problems for the interior of solid models are closely linked to material modeling applications [PKM13]. Given the diversity of application needs, the absence of unified representations of the solid interior stands out as a key issue that ought to be addressed. To get a feel for the diversity, consider the following.

Laminated, composite objects, such as airplane wings and ship hulls, need to be represented. Embedded sensors, even computers and actuators may have to be added to create active structures. Polycrystalline materials may require different representational constructs, as do porous materials, insulating materials, etc. Functionally graded materials must be represented and analyzed. In medical applications, organ shape and tissue structure must be represented and understood.

These, and other areas of applications all require new ideas on the representation side so that they can fully explore and analyze their research questions computationally [RRSS16]. A unified representation would therefore have significant scientific value.

57.5 SOURCES AND RELATED MATERIAL

RELATED CHAPTERS

Chapter 29: Triangulations and mesh generation
Chapter 35: Curve and surface reconstruction
Chapter 42: Geometric intersection
Chapter 45: Robust geometric computation
Chapter 52: Computer graphics
Chapter 56: Splines and geometric modeling
Chapter 61: Rigidity and scene analysis
Chapter 68: Two computational geometry libraries: LEDA and CGAL

REFERENCES

[ABC$^+$00] C. Armstrong, A. Bowyer, S. Cameron, J. Corney, G. Jared, R. Martin, A. Middleditch, M. Sabin, and J. Salmon. *Djinn: A Geometric Interface for Solid Modelling.* Information Geometers, Winchester, 2000.

[ACK01] N. Amenta, S. Choi, and R.K. Kolluri. The power crust. In *Proc. 6th ACM Sympos. Solid Modeling Appl.*, pages 249–260, 2001.

[AKK$^+$02] V. Adzhiev, E. Kartasheva, T. Kunii, A. Pasko, and B. Schmitt. Cellular-functional modeling of heterogeneous objects. In *Proc. 7th ACM Sympos. Solid Modeling Appl.*, pages 192–203, 2002.

[ALJ$^+$15] B.R. de Araújo, D.S. Lopes, P. Jepp, J.A. Jorge, and B. Wyvill. A survey on implicit surface polygonization. *ACM Comp. Surv.*, 47:60, 2015.

[ASME09] ASME. *Dimensioning and Tolerancing: Y.14.5-2009.* American Society of Mechanical Engineers, New York, 2009.

[AVH$^+$13] M.K. Abera, P. Verboven, E. Herremans, T. Defraeye, S.W. Fanta, Q.T. Ho, J. Carmeliet, and B.M. Nicolaï. 3D virtual pome fruit tissue generation based on cell growth modeling. *Food Bioprocess Tech.*, 7:542–555, 2013.

[BB97] J. Bloomenthal and C.L. Bajaj, editors. *Introduction to Implicit Surfaces.* Morgan Kaufmann, San Francisco, 1997.

[BCK88] B. Buchberger, G.E. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Annual Reviews Comp. Sci.*, 3:85–120, 1988.

[BFH$^+$95] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Comput.-Aided Des.*, 27:487–501, 1995.

[Bin01] S. Biner. Thermo-elastic analysis of functionally graded materials using Voronoi elements. *Materials Sci. Engrg: A*, 31:136–146, 2001.

[Blu73] H. Blum. Biological shape and visual science (part I). *J. Theoret. Biol.*, 38:205–287, 1973.

[BN90] P. Brunet and I. Navazo. Solid representation and operation using extended octrees. *ACM Trans. Graphics*, 9:170–197, 1990.

[Bra75] I. Braid. The synthesis of solids bounded by many faces. *Comm. ACM*, 18:209–216, 1975.

[Bra97] I. Braid. Non-local blending of boundary models. *Comput.-Aided Des.*, 29:89–100, 1997.

[Bri93] E. Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete Comput. Geom.*, 9:387–426, 1993.

[Bro82] C.M. Brown. PADL-2: A technical summary. *IEEE Comput. Graph. Appl.*, 2:69–84, 1982.

[BST04] A. Biswas, V. Shapiro, and I. Tsukanov. Heterogeneous material modeling with distance fields. *Comput. Aided Geom. Design*, 21:215–242, 2004.

[BW89] M.I.G. Bloor and M.J. Wilson. Generating blending surfaces with partial differential equations. *Comput.-Aided Des.*, 21:165–171, 1989.

[CCC$^+$08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: An open-source mesh processing tool. In *Proc. Eurographics Italian Chapter Conf.*, pages 129–136, 2008.

[CCH94] V. Capoyleas, X. Chen, and C.M. Hoffmann. Generic naming in generative, constraint-based design. *Comput.-Aided Des.*, 28:17–26, 1994.

[CH95a] X. Chen and C.M. Hoffmann. On editability of feature based design. *Comput.-Aided Des.*, 27:905–914, 1995.

[CH95b] X. Chen and C.M. Hoffmann. Towards feature attachment. *Comput.-Aided Des.*, 27:695–702, 1995.

[Chi88] H. Chiyokura. *Solid Modeling with Designbase*. Addison-Wesley, Boston, 1988.

[Cho88] S.-C. Chou. *Mechanical Geometry Theorem Proving*. Vol. 41 of *Mathematics and Its Applications*, Reidel Publishing Co., Dordrecht, 1988.

[CKM04] T. Culver, J. Keyser, and D. Manocha. Exact computation of the medial axis of a polyhedron. *Computer Aided Geom. Design*, 21:65–98, 2004.

[CMP95] V. Chandru, S. Manohar, and C.E. Prakash. Voxel-based modeling for layered manufacturing. *IEEE Comput. Graphics Appl.*, 15:42–47, 1995.

[CMS98] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22:37–54, 1998.

[CT00] M. Chen and J.V. Tucker. Constructive volume geometry. In *Computer Graphics Forum*, 19:281–293, 2000.

[DKP+92] D. Dutta, N. Kikuchi, P. Papalmbros, F. Prinz, and L. Weiss. Project Maxwell: Towards rapid realization of superior products. In *Proc. Solid Freeform Fabrication Sympos.*, pages 54–62, University of Texas at Austin, 1992.

[DPS14] A. DiCarlo, A. Paoluzzi, and V. Shapiro. Linear algebraic representation for topological structures. *Comput.-Aided Des.*, 46:269–274, 2014.

[DTD+15] E.L. Doubrovski, E.Y. Tsai, D. Dikovsky, J.M.P. Geraedts, H. Herr, and N. Oxman. Voxel-based fabrication through material property mapping: A design method for bitmap printing. *Comput.-Aided Des.*, 60:3–13, 2015.

[Elb15] G. Elber. Precise construction of micro-structures and porous geometry via functional composition. *Comput.-Aided Des.*, 2015. Manuscript submitted for publication.

[Far88] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design*. Academic Press, San Diego, 1988.

[FAR16] H.J. Fogg, C.G. Armstrong, and T.T. Robinson. Enhanced medial-axis-based block-structured meshing in 2-d. *Comput.-Aided Des.*, 72:87–101, 2016.

[FHJA16] I. Fudos, C.M. Hoffmann, and R. Joan-Arinyo. Tree-decomposable and underconstrained geometric constraint problems. Preprint, `arXiv:1608.05205`, 2016.

[For95] M. Forsyth. Shelling and offsetting bodies. In *Proc. 3rd ACM Sympos. Solid Modeling Appl.*, pages 373–381, 1995.

[For97] S. Fortune. Polyhedral modeling with multi-precision integer arithmetic. *Comput.-Aided Des.*, 29:123–133, 1997.

[FP09] A. Fabri and S. Pion. CGAL: The computational geometry algorithms library. In *Proc. 17th ACM SIGSPATIAL GIS*, pages 538–539, 2009.

[Fre96] S. Frechette. Interoperability requirements for CAD data transfer in the AutoSTEP project. Technical Report NISTIR 5844, National Institute of Standards and Technology, Gaithersburg, 1996.

[FST06] M. Freytag, V. Shapiro, and I. Tsukanov. Field modeling with sampled distances. *Comput.-Aided Des.*, 38:87–100, 2006.

[FST11] M. Freytag, V. Shapiro, and I. Tsukanov. Finite element analysis in situ. *Finite Elements in Analysis and Design*, 47:957–972, 2011.

[FVP13] O. Fryazinov, T. Vilbrandt, and A. Pasko. Multi-scale space-variant frep cellular structures. *Comput.-Aided Des.*, 45:26–34, 2013.

[GDC94] The Geometric Design and Computation Group, University of Utah. *Alpha_1 advanced experimental CAD modeling system*, `www.cs.utah.edu/gdc/projects/alpha1/`, 1994.

[GFM$^+$16] G. Gouaty, L. Fang, D. Michelucci, M. Daniel, J.-P. Pernot, R. Raffin, S. Lanquetin, and M. Neveu. Variational geometric modeling with black box constraints and dags. *Comput.-Aided Des.*, 75:1–12, 2016.

[GHY04] X.-S. Gao, C.M. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. *Comput.-Aided Des.*, 36:111–122, 2004.

[GLM96] S. Ghosh, K. Lee, and S. Moorthy. Two scale analysis of heterogeneous elastic-plastic materials with asymptotic homogenization and Voronoi cell finite element model. *Computer Methods Appl. Mechanics Engrg.*, 132:63–116, 1996.

[GQD13] Q. Ge, H.J. Qi, and M.L. Dunn. Active materials by four-dimension printing. *Applied Physics Letters*, 103:131901, 2013.

[Hen08] L. Henneberg. *Die graphische Statik der starren Körper*. In F. Klein and C. Müller, editors, *Encyklopädie der Mathematischen Wissenschaften mit Einschluss ihrer Anwendungen*, pages 345–434, Springer, Wiesbaden, 1908.

[HJ92] C.M. Hoffmann and R. Juan. Erep: An editable, high-level representation for geometric design and analysis. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric Modeling for Product Realization*, pages 129–164, North-Holland, Amsterdam, 1992.

[HJA05] C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Comput.-Aided Des.*, 2:655–663, 2005.

[HL93] J. Hoschek and D. Lasser. *Computer Aided Geometric Design*. A.K. Peters, Natick, 1993.

[HLS01a] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint problems, part II: New algorithms. *J. Symbolic Comput.*, 31:409–427, 2001.

[HLS01b] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems, part I: Performance measures for CAD. *J. Symbolic Comput.*, 31:367–408, 2001.

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

[Hof90] C.M. Hoffmann. Algebraic and numerical techniques for offsets and blends. In W. Dahmen, M. Gasca, and C.A. Micchelli, editor, *Computations of Curves and Surfaces*, pages 499–528, Kluwer Academic, Dordrecht, 1990.

[Hof92] C.M. Hoffmann. Computer vision, descriptive geometry, and classical mechanics. In B. Falcidieno and I. Herman, editors, *Computer Graphics and Mathematics*, pages 229–244, Springer, Berlin, 1992.

[Hof93a] C.M. Hoffmann. On the semantics of generative geometry representations. In *Proc. 19th ASME Design Automation Conf.*, vol. 2, pages 411–420, 1993.

[Hof93b] C.M. Hoffmann. On the separability problem of real functions and its significance in solid modeling. In *Computational Algebra*, vol. 151 of *Lecture Notes Pure Appl. Math.*, pages 191–204, Marcel Dekker, New York, 1993.

[Hof95] C. Hoffmann. Geometric approaches to mesh generation. In I. Babuska, J. Flaherty, W. Henshaw, J. Hopcroft, J. Oliger, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, pages 31–51, Springer, Berlin, 1995.

[Hof01a] C.M. Hoffmann. Robustness in geometric computations. *J. Comput. Inf. Sci. Eng.*, 1:143–155, 2001.

[Hof01b] C.M. Hoffmann. D-cubed's dimensional constraint manager. *J. Comput. Inf. Sci. Eng.*, 1:100–101, 2001.

[Hol05] S.J. Hollister. Porous scaffold design for tissue engineering. *Nature Materials*, 4:518–524, 2005.

[Hos92] M. Hosaka. *Modeling of Curves and Surfaces in CAD/CAM.* Springer, New York, 1992.

[HPR00] J. Han, M. Pratt, and W.C. Regli. Manufacturing feature recognition from solid models: A status report. *IEEE Trans. Robot. Autom.*, 16:782–796, 2000.

[HSS11] C.M. Hoffmann, V. Shapiro, and V. Srinivasan. Geometric interoperability for resilient manufacturing. Technical report CSD 11-015, Purdue University, 2011.

[HSS14] C. Hoffmann, V. Shapiro, and V. Srinivasan. Geometric interoperability via queries. *Comput.-Aided Des.*, 46:148–159, January 2014.

[HV94] C.M. Hoffmann and P. Vermeer. Geometric constraint solving in $\mathbb{R}^2$ and $\mathbb{R}^3$. In D.Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, 2nd edition, World Scientific, Singapore, 1994.

[HY00] C.M. Hoffmann and B. Yuan. On spatial constraint solving approaches. In *Proc. 3rd Workshop on Automated Deduction in Geometry*, vol. 2061 of *LNCS*, pages 1–15, Springer, Berlin, 2000.

[Jac95] D.J. Jackson. Boundary representation modelling with local tolerancing. In *Proc. 3rd ACM Sympos. Solid Modeling Appl.*, pages 247–253, 1995.

[JDKK07] L.J.M. Jacobs, K.C.H. Danen, M.F. Kemmere, and J.T.F. Keurentjes. Quantitative morphology analysis of polymers foamed with supercritical carbon dioxide using Voronoi diagrams. *Comput. Materials Sci.*, 38:751–758, 2007.

[JLP$^+$99] T.R. Jackson, H. Liu, N.M. Patrikalakis, E.M. Sachs, and M.J. Cima. Modeling and designing functionally graded material components for fabrication with local composition control. *Materials & Design*, 20:63–75, 1999.

[JS13] A.L.S. John and J. Sidman. Combinatorics and the rigidity of CAD systems. *Comput.-Aided Des.*, 45:473–482, 2013.

[KBDH99] V. Kumar, D. Burns, D. Dutta, and C. Hoffmann. A framework for object modeling. *Comput.-Aided Des.*, 31:541–556, 1999.

[KD97] V. Kumar and D. Dutta. An approach to modeling multi-material objects. In *Proc. 4th ACM Sympos. Solid Modeling Appl.*, pages 336–345, 1997.

[Kem99] S.J. Kemmerer. *STEP: the grand experience.* US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1999.

[KK13] A.K.M.B. Khoda and B. Koc. Functionally heterogeneous porous scaffold design for tissue engineering. *Comput.-Aided Des.*, 45:1276–1293, 2013.

[KKM99a] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: I—representations. *Comput. Aided Geom. Design*, 16:841–859, 1999.

[KKM99b] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: II—computation. *Comput. Aided Geom. Design*, 16:861–882, 1999.

[KMHP11] B.C. Kim, D. Mun, S. Han, and M.J. Pratt. A method to exchange procedurally represented 2D CAD model data using ISO 10303 STEP. *Comput.-Aided Des.*, 43:1717–1728, 2011.

[KPIS08] J. Kim, M.J. Pratt, R.G. Iyer, and R.D. Sriram. Standardized data exchange of CAD models with design intent. *Comput.-Aided Des.*, 40:760–777, 2008.

[Kra92] G. Kramer. *Solving Geometric Constraint Systems.* MIT Press, Cambridge, 1992.

[KRU94] F.-L. Krause, E. Rieger, and A. Ulbrich. Feature processing as kernel for integrated CAE systems. In *Proc. IFIP Conf. Feature Modeling and Recognition in Advanced CAD/CAM Systems*, vol. II, pages 693–716, 1994.

[KT07] X.Y. Kou and S.T. Tan. Heterogeneous object modeling: A review. *Comput.-Aided Des.*, 39:284–301, 2007.

[KT10] X.Y. Kou and S.T. Tan. A simple and effective geometric representation for irregular porous structure modeling. *Comput.-Aided Des.*, 42:930–941, 2010.

[KW14] M. Kintel and C. Wolf. Openscad. *GNU General Public License, p GNU General Public License*, 2014.

[Lam70] G. Laman. On graphs and the rigidity of plane skeletal structures. *J. Engrg. Math.*, 4:331–340, 1970.

[LS15] X. Liu and V. Shapiro. Random heterogeneous materials via texture synthesis. *Comput. Materials Sci.*, 99:177–189, 2015.

[LS16] X. Liu and V. Shapiro. Sample-based design of functionally graded material structures. In *ASME Design Engineering Technical Conf. and Computers and Inf. in Engng. Conf.*, pages IDETC2016–60431, 2016.

[M$^+$10] R. McNeel et al. Grasshopper-generative modeling with rhino. `http://www.grasshopper3d.com`, 2010.

[Män88] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, New York, 1988.

[Max64] J.C. Maxwell. On reciprocal figures and diagrams of forces. *Philos. Mag.*, 27:250–261, 1864.

[MCD11] S. Musuvathy, E. Cohen, and J. Damon. Computing medial axes of generic 3D regions bounded by B-spline surfaces. *Comput.-Aided Des.*, 43:1485–1495, 2011.

[Mea82] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics Image Processing*, 19:129–147, 1982.

[MP02] D. Marcheix and G. Pierra. A survey of the persistent naming problem. In *Proc. 7th ACM Sympos. Solid Modeling Appl.*, pages 13–22, 2002.

[MUSA15] O. Morgan, K. Upreti, G. Subbarayan, and D.C. Anderson. Higeom: A symbolic framework for a unified function space representation of trivariate solids for isogeometric analysis. *Comput.-Aided Des.*, 65:34–50, 2015.

[MVH$^+$08] H.K. Mebatsion, P. Verboven, Q.T. Ho, B.E. Verlinden, and B.M. Nicolaï. Modelling fruit (micro)structures, why and how? *Trends Food Sci. Tech.*, 19:59–66, 2008.

[MY14] M.Y. Ma and H. Ye. An image analysis method to obtain the effective thermal conductivity of metallic foams via a redefined concept of shape factor. *Appl. Thermal Engrg.*, 73:1277–1282, 2014.

[Nay90] B. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Comput.-Aided Des.*, 22:250-252, 1990.

[NMLS15] Z. Niu, R.R. Martin, F.C. Langbein, and M.A. Sabin. Rapidly finding CAD features using database optimization. *Comput.-Aided Des.*, 69:35–50, 2015.

[NR95] B. Naylor and L. Rogers. Constructing binary space partitioning trees from piecewise Bézier curves. In *Proc. Graphics Interface*, pages 181–191, 1995.

[NSK$^+$97] M. Näf, G. Székely, R. Kikinis, M.E. Shenton, and O. Kübler. 3D Voronoi skeletons and their usage for the characterization and recognition of 3D organ shape. *Computer Vision and Image Understanding*, 66:147–161, 1997.

[Owe91] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In *Proc. 1st ACM Sympos. Solid Modeling Found. and CAD/CAM Appl.*, pages 397–407, 1991.

[Oxm11] N. Oxman. Variable property rapid prototyping. *Virtual Physical Prototyping*, 6:3–31, 2011.

[PASS95] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: Concepts, implementation and applications. *The Visual Computer*, 11:429–446, 1995.

[PKM13] J.H. Panchal, S.R. Kalidindi, and D.L. McDowell. Key computational modeling issues in integrated computational materials engineering. *Comput.-Aided Des.*, 45:4–25, 2013.

[PM02] N.M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer-Aided Design and Manufacture*. Springer, Berlin, 2002.

[PPV$^+$03] A. Paoluzzi, V. Pascucci, M. Vicentino, C. Baldazzi, and S. Portuesi. *Geometric Programming for Computer Aided Design*. John Wiley & Sons, New York, 2003.

[PS93] R.S. Palmer and V. Shapiro. Chain models of physical behavior for engineering analysis and design. *Research Engrg. Design*, 5:161–184, 1993.

[PS95] A. Pasko and V. Savchenko. Algebraic sums for deformation of constructive solids. In *Proc. 3rd ACM Sympos. Solid Modeling Appl.*, pages 403–408, 1995.

[QS06] J. Qi and V. Shapiro. ε-topological formulation of tolerant solid modeling. *Comput.-Aided Des.*, 38:367–377, 2006.

[Req77] A.A.G. Requicha. Mathematical models of rigid solids. Technical Report PAP 28, University of Rochester, 1977.

[RO89] J.R. Rossignac and M.A. O'Connor. *SGC: A Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries*. IBM TJ Watson Research Center, Yorktown Heights, 1989.

[Ros07] D.W. Rosen. Computer-aided design for additive manufacturing of cellular structures. *Computer-Aided Design Appl.*, 4:585–594, 2007.

[Rot82] S.D. Roth. Ray casting for modeling solids. *Computer Graphics Image Processing*, 18:109–144, 1982.

[RR91] J.R. Rossignac and A.A.G. Requicha. Constructive non-regularized geometry. *Comput.-Aided Des.*, 23:21–32, 1991.

[RRSS16] W. Regli, J. Rossignac, V. Shapiro, and V. Srinivasan. The new frontiers in computational modeling of material structures. *Comput.-Aided Des.*, 77:73–85, 2016.

[RS98] S. Raghothama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Trans. Graphics*, 17:259–286, 1998.

[RS00] S. Raghotama and V. Shapiro. Consistent updates in dual representation systems. *Comput.-Aided Des.*, 32:463–477, 2000.

[RS02a] S. Raghotama and V. Shapiro. Topological framework for part families. In *Proc. 7th ACM Sympos. Solid Modeling Appl.*, pages 1–12, 2002.

[RS02b] S. Raghothama and V. Shapiro. Topological framework for part families. *J. Comput. Inf. Sci. Eng.*, 2:246–255, 2002.

[RSB96] A. Rappoport, A. Sheffer, and M. Bercovier. Volume-preserving free-form solids. *IEEE Trans. Visualization Computer Graphics*, 2:19–27, 1996.

[RV77] A.A.G. Requicha and H.B. Voelcker. Constructive solid geometry. Tech. Report 25, University of Rochester, 1977.

[RV82] A.A.G. Requicha and H.B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Comput. Graphics Appl.*, 2:9–24, 1982.

[RV83] A.A.G. Requicha and H.B. Voelcker. Solid modeling: Current status and research directions. *IEEE Comput. Graphics Appl.*, 3:25–37, 1983.

[RV85] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proc. IEEE*, 73:30–44, 1985.

[RW91] S.J. Rock and M.J. Wozny. A flexible file format for solid freeform fabrication. In *Proc. Solid Freeform Fabrication Sympos.*, pages 1–12, University of Texas at Austin, 1991.

[Sam89a] H.J. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison–Wesley, Boston, 1989.

[Sam89b] H.J. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods*. Addison–Wesley, Boston, 1989.

[SAR95] D.J. Sheehy, C.G. Armstrong, and D.J. Robinson. Computing the medial surface of a solid from a domain Delaunay triangulation. In *Proc. 3rd ACM Sympos. Solid Modeling Appl.*, pages 201–212, 1995.

[Sha91a] J.J. Shah. Assessment of features technology. *Comput.-Aided Des.*, 23:331–343, 1991.

[Sha91b] V. Shapiro. *Representations of Semialgebraic Sets in Finite Algebras Generated by Space Decompositions*. PhD thesis, Sibley School of Mechanical Engineering, Cornell University, 1991.

[Sha97] V. Shapiro. Errata: Maintenance of geometric representations through space decompositions. *Internat. J. Comput. Geom. Appl.*, 7:383–418, 1997.

[Sha07] V. Shapiro. Semi-analytic geometry with r-functions. *Acta Numer.*, 16:239, 2007.

[Sha08] V. Shapiro. Homotopy conditions for tolerant geometric queries. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, pages 162–180, Springer, Berlin, 2008.

[SHL92] J. Shah, D. Hsiao, and J. Leonard. A systematic approach for design-manufacturing feature mapping. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric Modeling for Product Realization*, pages 205–222, North Holland, Amsterdam, 1992.

[SI89] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. *J. Inform. Process.*, 12:380–393, 1989.

[Si15] H. Si. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Software*, 41:11, 2015.

[SPB04] T. Sakkalis, T.J. Peters, and J. Bisceglio. Isotopic approximations and interval solids. *Comput.-Aided Des.*, 36:1089–1100, 2004.

[SR04] S. Spitz and A. Rappoport. Integrated feature-based and geometric CAD data exchange. In *Proc. 9th ACM Sympos. Solid Modeling Appl.*, pages 183–190, 2004.

[Sri03] V. Srinivasan. *Theory of Dimensioning: An Introduction to Parameterizing Geometric Models*. CRC Press, Boca Raton, 2003.

[SS10] R. Schmidt and K. Singh. Meshmixer: An interface for rapid mesh composition. In *Proc. ACM SIGGRAPH*, article 6, 2010.

[SSND05] W. Sun, B. Starly, J. Nam, and A. Darling. Bio-CAD modeling and its applications in computer-aided tissue engineering. *Comput.-Aided Des.*, 37:1097–1114, 2005.

[SSP01] G. Shen, T. Sakkalis, and N. Patrikalakis. Analysis of boundary representation model rectification. In *Proc. 6th ACM Sympos. Solid Modeling Appl.*, pages 149–158, 2001.

[SV93] V. Shapiro and D. Vossler. Separation for boundary to CSG conversion. *ACM Trans. Graphics*, 12:35–55, 1993.

[Sys01] S. Systems. *The SGDL Language*. `www.sgdl-sys.com`, 2001.

[Tak92] T. Takala. A taxonomy on geometric and topological models. In B. Falcidieno, I. Herman, and C. Pienovi, editors, *Computer Graphics Math.*, pages 147–171, Springer, Berlin, 1992.

[Til80] R.B. Tilove. Set membership classification: A unified approach to geometric intersection problems. *IEEE Trans. Computers*, 100:874–883, 1980.

[TN87] W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proc. 14th ACM Conf. Comp. Graphics*, pages 153–162, 1987.

[TWM85] J.E. Thompson, Z.U.A. Warsi, and C.W. Mastin. *Numerical Grid Generation*. North Holland, Amsterdam, 1985.

[UMP95] Unit Manufacturing Process Research Committee. *Unit Manufacturing Processes: Issues and Opportunities in Research*. National Academies Press, Washington, 1995.

[Ver94] P. Vermeer. *Medial Axis Transform to Boundary Representation Conversion*. PhD thesis, Purdue University, 1994.

[WCR05] H. Wang, Y. Chen, and D.W. Rosen. A hybrid geometric modeling method for large scale conformal cellular structures. In *Proc. ASME Internat. Design Engineering Technical Conf. and Computers and Inf. in Engng. Conf.*, pages 421–427, 2005.

[Wei86] K. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Inst., 1986.

[WGG99] B. Wyvill, A. Guy, and E. Galin. Extending the CSG tree. Warping, blending and Boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18:149–158, 1999.

[YBSH16] U. Yaman, N. Butt, E. Sacks, and C. Hoffmann. Slice coherence in a query-based architecture for 3D heterogeneous printing. *Comput.-Aided Des.*, 75:27–38, 2016.

[Zag97] J. Zagajac. *Engineering Analysis over Subsets*. PhD thesis, The Sibley School of Mechanical and Aerospace Engineering, Cornell University, 1997.