1991

Fundamental Techniques for Geometric and Solid Modeling

Christoph M. Hoffmann
Purdue University, cmh@cs.purdue.edu

George Vaněček

Report Number:
91-044

FUNDAMENTAL TECHNIQUES FOR GEOMETRIC AND SOLID MODELING

Christoph M. Hoffman
George Vanecek, Jr.

CSD-TR-91-044
June 1991

FUNDAMENTAL TECHNIQUES FOR GEOMETRIC AND SOLID MODELING[1]

Abstract

We review traditional and novel paradigms for representing solids and interrogating them. The traditional paradigms reviewed are the boundary, constructive, and spatial subdivision representations. The novel representation paradigms are the B-rep index, the dimensionality paradigm, and the skeleton (medial-axis transform).

The *B-rep index* is a polyhedral representation that integrates boundary and subdivision representation. We show how to construct it and explain some of the advantages this representation offers for operations such as point/solid and line/solid classification. We also discuss how it can account for the robustness problem.

The *dimensionality paradigm* is a technique for representing exactly complex surfaces that satisfy prescribed constraints, and we discuss some of the algorithmic infrastructure available to manipulate and interrogate this surface representation. This paradigm generalizes both implicit and parametric representations, and can deal with surfaces that otherwise could be obtained exactly only through elimination computations of forbidding complexity.

The *skeleton* is a solid representation originally proposed in computer vision. It is an informationally-complete solid representation, and seems to facilitate operations such as automatic mesh generation for finite-element computations, and geometric tolerancing. We show how the skeleton relates to the cyclographic map, a concept from classical descriptive geometry, and also explain the relationship between the skeleton and the Hamilton-Jacobi equation. Finally, we review some algorithms for two-dimensional mesh generation based on the skeleton.

[1]To appear in *Advances in Control and Dynamics*, C. T. Leondes, ed., Academic Press.

Contents

FUNDAMENTAL TECHNIQUES FOR GEOMETRIC AND SOLID MODELING

CHRISTOPH M. HOFFMANN[1]
GEORGE VANĚČEK, JR.[2]

Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

I. INTRODUCTION

There are three well-established paradigms for representing solids that are based on the boundary, on spatial subdivision, and on construction from primitives using regularized set operations. Around these paradigms, a substantial literature has grown and many efficient and ingenious algorithms have been devised for working with solids so represented. Yet despite this extensive work, many tasks of interest remain that appear to be cumbersome to implement based on these traditional solid and surface representations. For instance, given a solid, how can we derive a new solid that is the *offset* of the old one, having

[1]Supported in part by ONR Contract N00014-90-J-1599, NSF Grant CCR 86-19817, and NSF Grant ECD 88-03017.
[2]Supported by NSF Grant CCR 86-19817.

a surface that is at constant distance from the old one? To devise a complete algorithm for this task is not simple. In fact, the mathematical difficulty of offsetting a general curved surface is in marked contrast to the simplicity with which this task can be defined and communicated between people. Difficulties of this kind suggest that we remain on the lookout for new solid and surface representations that might facilitate such operations.

In this chapter, we present a number of new paradigms for representing solids and surfaces. These new approaches show potential for pressing practical problems, and, in some cases, have already delivered. Yet one should not conclude that they will therefore displace traditional representation paradigms. Indeed, a representation is intimately linked with algorithmic efficiency and convenience, and so one should expect that there will always be a need to switch to a different representation in response to the algorithmic problem at hand. The new representations we discuss here are intended to supplement the repertoire of geometric and solid modeling, not to supplant it.

In accordance with our outlook that many representations will continue to coexist, we begin by reviewing the three classical representation paradigms and the algorithmic ideas underlying their interrogation. So, we review *spatial subdivision*, *boundary representation*, and *constructive solid geometry*. We then discuss in some detail the *B-rep index*, a spatial subdivision structure that integrates boundary-based and spatial-subdivision representations. The B-rep index has proved to be extremely valuable in applications in which a large number of moving objects are queried for collision, and where, upon collision, a geometric analysis must be made of the locale at which a contact has been determined. The favorable performance of the B-rep index in this situation is based on the ease with which lines and points can be classified, with respect to a solid. But the B-rep index also does well in generating rectangular meshes of solid domains, and is also amenable to algorithmic enhancements that significantly increase the robustness of geometric computations.

Technically, constructing the B-rep index from a boundary-based representation (B-rep) is akin to the problem of converting a B-rep to constructive solid geometry. This problem is fully solved in the polyhedral case, but only partially solved in the curved-surface case.

Just as the performance of a solid representation depends on the operation one has in mind, surface representations exhibit a similar relativity. The two major paradigms are the parametric and the implicit surface representations. Again, each has specific strengths and weaknesses, and certain rather intuitive and desirable operations, including offsetting, are difficult to carry out exactly. Here, the *dimensionality paradigm* offers some attractive alternatives. We review the dimensionality paradigm in the section on constrained-surface representations, giving a detailed example illustrating the method, and discussing how some of the standard interrogation algorithms on surfaces so represented can be implemented.

A major strong point of the dimensionality paradigm is its ability to represent exactly the *bisectors* of curved surfaces. That is, given two surfaces, we can represent *exactly* those points in three space that have equal distance from the given surfaces. This allows us to represent the *skeleton* precisely, another alternate solid representation scheme that recently has generated much interest.

Originally proposed as a shape representation in computer vision, the *skeleton* has been used successfully in a number of algorithms for generating finite-element meshes completely automatically. Furthermore, the skeleton is intimately related to a classical geometric concept, the *cyclographic map*, which can be thought of as an explicit map of the Euclidean distance of the points in space from a given geometric shape. More than that, the skeleton is the locus of the shocks in the Hamilton-Jacobi equation, and thus can be computed in principle with standard PDE solvers.

We discuss the skeleton in the final section of this chapter, explaining how it relates to cyclographic maps and to the Hamilton-Jacobi equation, and discuss several algorithms for computing the skeleton. We then review two algorithms for finite-element mesh generation, and remark on potential applications in geometric tolerancing.

II. TRADITIONAL PARADIGMS FOR REPRESENTING SOLIDS

Point set topology provides a precise language for describing the basic properties of solids, [1]. Let $\mathbf{E}^3$ denote three-dimensional Euclidean space with the usual topology, and let A be a subset of $\mathbf{E}^3$. The *interior of* A is denoted by iA, the *boundary of* A by bA, and the *complement of* A by cA. Note that the interior, the boundary and the complement of A partition $\mathbf{E}^3$. The *closure of* A, kA, is obtained by adding to A the limits of convergent point sequences in iA. The point set A is *regular* if $A = kiA$. If A is not regular, then we *regularize* it by forming $rA = kiA$. A set is *bounded* if it is contained in an open ball. An *r-set* is a regular and bounded set.

Geometrically, the boundaries of r-sets may be extremely complex [2]. In solid modeling, only surfaces that are simple in a technical sense are of interest [3]. Roughly speaking, the boundary must be finitely describable, and must be of finite variation; i.e., any line segment is contained or intersects the boundary in finitely many points. Then *solids* are r-sets that are simple in this sense.

A solid can be represented explicitly by its boundary, or by its volume, or implicitly by specifying operations on volumetric primitives that construct it. Accordingly, there are three dominant schemas: boundary representations, cell-decomposition representations, and constructive solid geometry; e.g., [2,4].

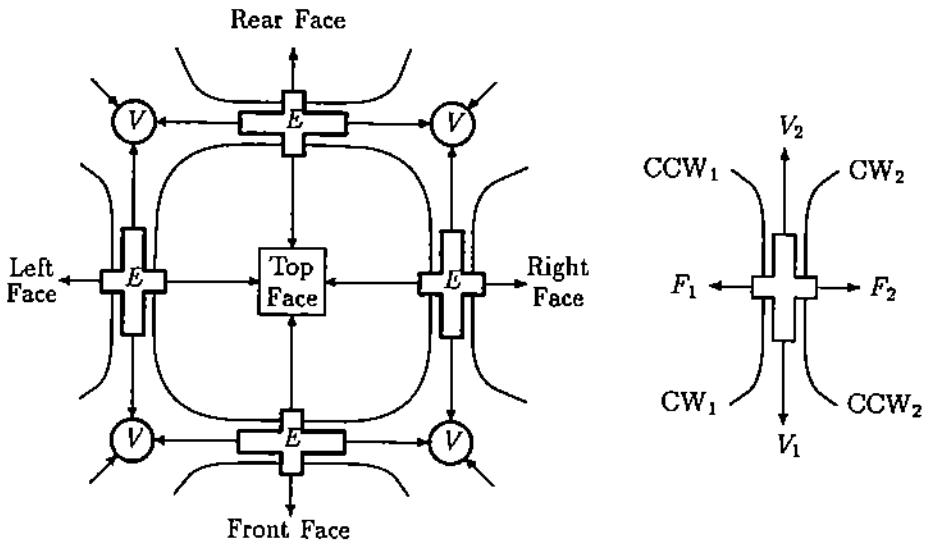


Figure 1: Top Face of Box Represented by the Winged-Edge Data Structure; Edge Node Format.

A. BOUNDARY REPRESENTATIONS

The boundary of a solid consists of vertices, edges and faces. For each of these entities, the geometric part of the representation fixes the shape and/or location in space, and the topological part records the adjacencies. The combination of the topological and geometrical information is a boundary representation (B-rep) of a solid.

In all, there are nine adjacency relationships between the three types of topological entities. For example, the face-edge adjacency relationship specifies for each face the adjacent edges. A complete solid modeling representation must allow the retrieval of any topological entity and any of the nine adjacency relationships. However, explicitly maintaining all nine adjacency relationships is redundant. Weiler has shown that three of the ordered adjacency relationships are sufficient to obtain all others [5]. There is a space/time tradeoff: explicitly maintaining all adjacency relationships requires more space but little time retrieving them. On the other hand, maintaining only sufficient adjacency relationship requires little space but more time to derive the others. A comparison of the space/time tradeoffs of different representation schemas, each consisting of a different subset of the topological adjacency relationships, can be found in [6].

Many different kinds of boundary representations are used for representing solids with a manifold surface. An early representation schema is Baumgart's winged-edge data structure for manifold solids [7]. In the winged-edge data

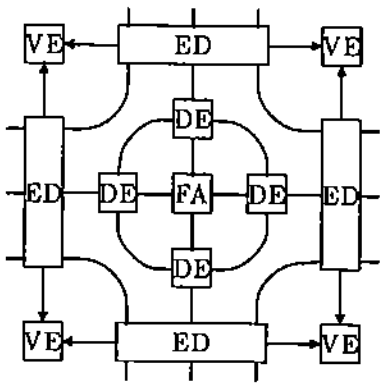


Figure 2: Diagrams of Directed-Edge Based Data Structure Where VE, ED, DE, and FA Mark Vertices, Edges, Directed Edges and Faces.

structure, an edge node records the information about the edge orientation, the face adjacencies, and the clockwise and counterclockwise successor and predecessor edges about the adjacent faces. An example representing the top face of a box is shown in Figure 1. The representation assumes that the faces are simply connected. Multiply connected faces need to be partitioned in this representation.

Braid modified the winged-edge data structure to include multiply connected faces by introducing a fourth topological entity called a *loop*. In Braid's data structure, each face consists of one or more edge loops, each bounding a hole in the face [8].

Yamaguchi and Tokieda modified the winged-edge data structure differently by introducing bridge edges to allow multiply connected faces [9]. A *bridge edge* is a double edge that connects two edge cycles of a given face. In their representation, all faces are triangulated using bridge edges. Other manifold representations are Mäntylä's half-edge data structure [4], Guibas and Stolfi's quad-edge representation [10], Hanrahan's face-edge representation [11], and Ansaldi, De Floriani and Falcidieno's hierarchical face adjacency hypergraph [12]. There are others.

The result of set operations can be a solid with a nonmanifold boundary. Manifold representations such as the winged-edge data structure [7,13] can handle nonmanifold solids only in special cases, and this complicates the algorithms for set operations unnecessarily [14]. In response, nonmanifold representations have been investigated; e.g., by Kevin Weiler who introduced the radial-edge data structure [15]. The radial-edge data structure can accommodate general nonmanifold models. More generally, the data structure can model arbitrary solids and structures that are not regularized. For example, a single object

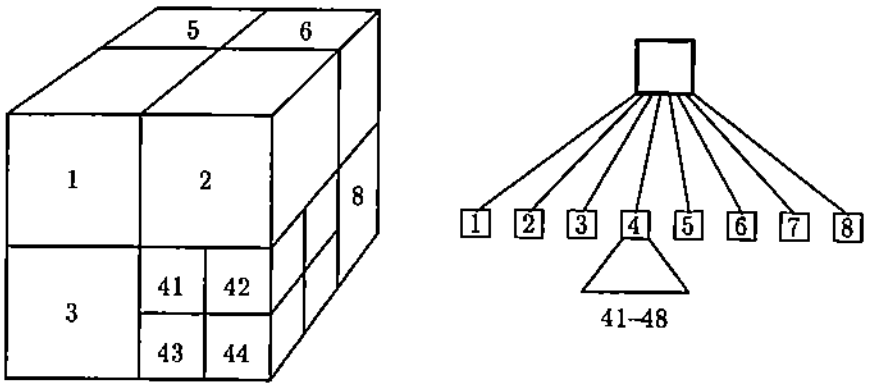


Figure 3: The Octree Data Structure.

may contain wire-frames, isolated vertices and edges, and arbitrary regions that need not be bounded. Focusing on regularized set operations, others have developed variations of the half-edge data structure to allow nonmanifolds. Two similar such data structures are Karasick's star-edge data structure [16], and Vaněček's fedge-based data structure [17,18]. Karasick's representation contains face loop and shell information explicitly. Vaněček's representation uses bridge edges instead, but without triangulating faces. An example of Vaněček's representation is shown in Figure 2.

B. CELL-DECOMPOSITION REPRESENTATIONS

Instead of representing the boundary of an object, the object can be represented explicitly by its volume. The volume is represented as a collection of cells of a partition of space. The various data structures differ in how they organize the cells and what information about the object each cell contains. Some widely used data structures are hierarchical and subdivide space recursively.

The simplest hierarchical data structure is the *region octree* based on regular decomposition. The method partitions a cuboidal region into eight equal sized octants. The region is represented by a node in the region octree, and the eight octants are its eight children. Each node in the tree is labeled either as a *gray* node, if it is further decomposed, a *white* node if it is completely outside the object, or a *black* node if it is completely inside the object. A simple example is shown in Figure 3.

Region octrees are suitable only for solids with faces that are parallel to the principal axes. Solids with inclined faces are approximated by region octrees. Thus they give only a rough description of the boundary of the object. The accuracy of the approximation depends on the subdivision level. For a thorough

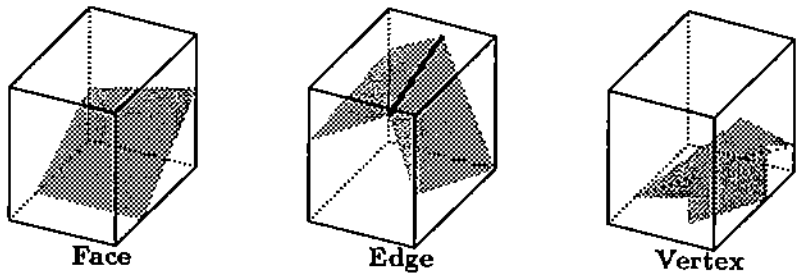


Figure 4: Examples of Face, Edge, and Vertex Nodes of Extended Octree.

discussion of octrees, hierarchical data structures, and related algorithms, see Samet [19].

Region octrees have been generalized to take the object boundary into account, economizing the space requirements of the tree. Samet and Webber developed the PM-octree which added nodes of type vertex, edge and face to the already present black, white and gray nodes. For regions that contain portions of the boundary, the recursive decomposition terminates when the region contains exactly one vertex, one edge or one face. For example, a region with a vertex and its adjacent edges and faces is represented directly and is not further decomposed. Variations of PM-octrees are the *polytree* by Carlbom, Chakravarty and Vanderschel [20], and the *extended octree* of Navazo, Ayala and Brunet [21].

The two tree structures differ in minor ways. Extended octrees assume only manifolds, while polytrees allow nonmanifolds. Extended octrees maintain for each leaf node a list of the oriented support planes of the faces intersecting the region, along with information on how to construct the faces. In contrast, in polytrees each leaf node contains a list of polygons representing the parts of the faces that intersect that region. No adjacency information is kept between the edges of two adjacent polygons in one region or between adjacent regions. Thus all regions have independent information.

Both the extended octree and the polytree are allowed to grow only to a prespecified depth. For certain solids, larger trees would be required, for instance, for a solid two vertices that are very close together. A region might then require further decomposition, but is not decomposed. Instead, it is represented by a *nasty* node that stores a list of pointers to all the planes (or polygons) it contains. The size of region octrees, extended octrees and polytrees is proportional to the solid's surface area and resolution [22,21]. Furthermore, the tree structure is sensitive to rotations other than by 90°.

A generic drawback of the octree and its variants is that the topological structure of the boundary is not readily available. A single face of a solid might

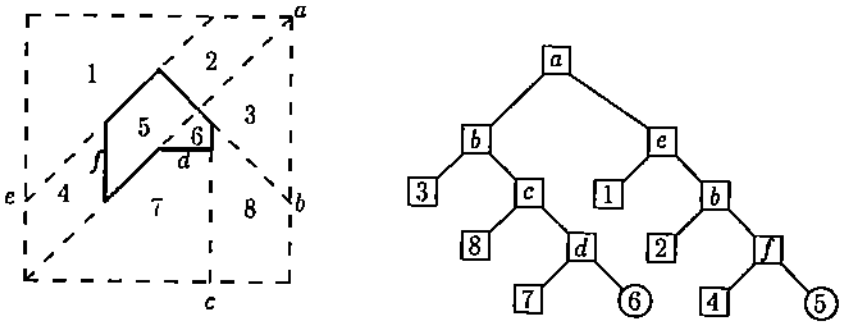


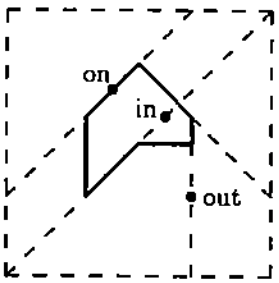
Figure 5: 2D Example of Solid and Its BSP Tree. Regions 5 and 6 are Inside.

extend across many nodes. If a boundary representation is needed, it must be derived. In the case of region octrees this computation is intricate. On the other hand, the octree inherently localizes space so that most problems can be solved recursively in a small volume of space rather that globally. Extended octrees and polytrees are more compact than the region octrees, and provide implicitly an exact representation of the boundary.

The octree is constructed by recursively dividing a cube into eight octants. If the cube is cut into two equal boxes instead, the structure is called a *bintree*; e.g., [19]. In a bintree, space is divided into two equal halves by a single plane that is orthogonal to one of the coordinate axes. When the cut plane can be sloped, we obtain the *binary space partition tree* (BSP). A BSP tree uses cut planes that are the support planes of the faces of the polyhedron that the BSP tree represents. The structure of the tree is not unique, and its size may vary. Both depend on the order in which the cut planes have been selected in the conversion of the B-rep to BSP, because it influences the amount of edge and face fragmentation. Thus carefully choosing the cut planes can reduce the size of the tree significantly. Naylor has suggested a variety of heuristics for choosing splitting planes [23].

Fuchs, Kedem and Naylor first used the BSP tree for determining visible surfaces in computer graphics [24]. Thibault and Naylor showed later how the BSP trees can be used to model arbitrary polyhedra [23]. They also showed how to perform Boolean set operations on BSP trees, by merging a BSP tree with a B-rep, and how to convert B-reps to BSP trees and BSP trees to B-reps. As an example, Figure 5 shows a polygon and its BSP tree. Recently, [25] proposed a merging algorithm that can be used to perform Booleans on two BSP trees directly.

With the BSP tree representation it is easy to classify points, that is, to determine whether the point is inside, outside or on the boundary of the solid



Above	Below	Class
in	in	in
in	out	on
out	in	on
out	out	out
in/out	on	on
on	in/out	on

Table 1: Classifying Points on Cut Planes of BSP Tree.

represented by the BSP tree. The point is classified by starting at the root of the BSP tree and at each internal node moving down into the subtree corresponding to the half space containing the point. When the point reaches a leaf, the classification of INSIDE or OUTSIDE is returned. When the point lies on a cut plane, both subtrees are visited, and the classifications from both are combined to determine the final classification. An example of each is shown in Table 1.

Line classification is a direct extension of the point classification: A line is inserted at the top of the BSP tree, cut into segments by the cut planes, and the segments are passed to the respective subtrees. If the segment is in a cut plane, it is passed to both subtrees, and subdivided and reclassified in a second pass from the leaves to the root.

BSP trees have been applied in computer graphics to modeling three-dimensional scenes. They are also well-suited to problems that require an efficient point/solid and line/solid classification, such as grid generation [26] and collision detection and analysis [27].

C. CONSTRUCTIVE SOLID GEOMETRY REPRESENTATIONS

Both the boundary-based and the volume-based representations we have discussed are explicit representations. They provide information on the shape or the volume of a solid, but not on its possible construction. If instead a solid is described in terms of operations on simple volumetric primitives, we obtain an implicit constructive representation. Such a representation is *constructive solid geometry* (CSG) [3].

A CSG representation is a tree structure in which the internal nodes represent operations and transformations, and the leaves represent primitives. The operators are regularized Boolean set operations, and transformations that position and orient the solid represented by the subtree. See Figure 6. The primitives can be the set of all closed linear half spaces. They are easy to evaluate and can be converted readily to equivalent B-reps. CSG representa-

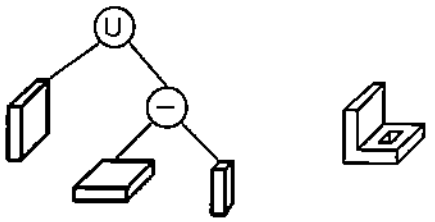


Figure 6: CSG Representation and Corresponding Polyhedron

tions are not restricted to linear half spaces. The standard primitives include the natural quadrics, sphere, cone and cylinder, as well as the torus. More generally, low-degree algebraic half spaces have been used in the Bath solids modeler [28]. Other possible primitives include swept volumes, extrusions of planar contours, or solids of revolution.

Requicha, Volcker and Tilove, and Voelcker and Requicha presented algorithms for converting from CSG to B-rep, based on a generate and test paradigm. An equivalent B-rep is obtained by traversing the CSG tree intersecting the surfaces bounding the half spaces or primitives. The resulting curves are clipped with a curve/solid classification. The edges so obtained are connected to form a wire frame. By computing the edge neighborhoods, the faces are then determined and added, thus completing the B-rep. Converting a B-rep to a CSG representation is more difficult. Recently, Shapiro and Vossler have developed such conversion algorithms, in 2D, and have discussed methods for minimizing the resulting CSG trees [29,30].

D. BOOLEAN SET OPERATIONS ON SOLIDS

Consider two solids A and B. Their set-theoretic union, difference, and intersection need not produce an r-set. However, the result of the set-theoretic operation can be "trimmed" so that we obtain an r-set. The trimming operation is called *regularization* and is defined as $rC = kiC$. In Figure 7 we see that regularization eliminates the dangling edges and faces left by the set-theoretic operation.

Algorithms for, say, intersecting two solids could first compute the set-theoretic intersection and then regularize it. However, it is more convenient to incorporate regularization directly into the intersection computation. The algorithms vary with the details of the representation, but all are conceptually based on boundary classification. We will discuss them for CSG, for boundary representations, and for octree representations.

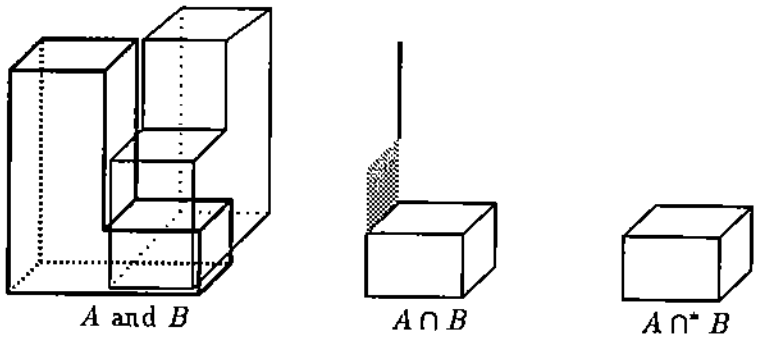


Figure 7: Set-Theoretic and Regularized Intersection of Solids A and B

1. SET OPERATIONS IN CSG AND CONVERSION TO B-REP

The constructive representation of a solid is a tree structure in which the internal nodes represent either the regularized set operators or rigid transformations, and the leaf nodes represent primitive solids. Since this is an implicit representation, performing a boolean operation on two solids given by CSG trees is trivial: A new tree is formed by creating a new root labeled with the set operation and with the CSG trees of the two solids as children. However, boundary evaluation which generates the B-rep described by the given CSG, is not trivial.

In their work on CSG, Voelcker and Requicha showed that the regularized operations can be defined as the selective combination of certain sets, called the *classification sets*. The classification sets are computed by a membership classification function $M[X, S]$ which partitions a candidate set X with respect to a reference set S. The reference set S is regular in a topological space $W \subset \mathbf{E}^3$, while the candidate set X is regular in a subspace $W' \subset W$, in the relative topology.

Definition 1 If X is regular in W' and S is regular in W, then the membership classification function, $M[X, S]$, is defined as follows:

$$\begin{aligned} M[X,S] &= (X\mathrm{in}S, X\mathrm{on}S, X\mathrm{out}S), \text{ where} \\ X\mathrm{in}S &= X \cap^{*'} iS = k'i'(X \cap iS), \\ X\mathrm{on}S &= X \cap^{*'} bS, \text{ and} \\ X\mathrm{out}S &= X \cap^{*'} cS. \end{aligned}$$

The classification sets $X\mathrm{in}S$, $X\mathrm{on}S$, and $X\mathrm{out}S$ correspond to regular subsets of X that are respectively inside, on the boundary, or outside S. Together they form the regular partition of X:

Theorem 1[Tilove] If $M[X,S] \simeq (X\text{in}S, X\text{on}S, X\text{out}S)$ then

$$X = X\text{in}S \cup^{*'} X\text{on}S \cup^{*'} X\text{out}S,$$

and

$$X\text{in}S \cap^{*'} X\text{on}S = X\text{on}S \cap^{*'} X\text{out}S = X\text{in}S \cap^{*'} X\text{out}S = \emptyset.$$

Through the use of the membership classification function, a classification result of a regularized composition of set operations can be expressed recursively in terms of the classification results. The following theorem states this result for the intersection operation. See Tilove's Master's thesis [31] for similar theorems for the operations of regularized union and set difference.

Theorem 2[Tilove] Let

$$\begin{aligned} M[X,A] &\simeq (X\text{in}A, X\text{on}A, X\text{out}A) \\ M[X,B] &= (X\text{in}B, X\text{on}B, X\text{out}B) \end{aligned}$$

and let $S = A \cap^{*} B$. Then $M[X,S] \simeq (X\text{in}S, X\text{on}S, X\text{out}S)$, where:

$$\begin{aligned} X\text{in}S &= X\text{in}A \cap^{*'} X\text{in}B, \\ X\text{on}S &= (X\text{in}A \cap^{*'} X\text{on}B) \cap^{*'} (X\text{on}A \cap^{*'} X\text{in}A) \\ &\quad \cap^{*'} r' \left(\{p \in X\text{on}A \cap^{*'} X\text{on}B \mid N(p;S) = \emptyset\}\right), \text{ and} \\ X\text{out}S &= X\text{out}A \cup^{*'} X\text{out}B \cup^{*'} \\ &\quad r' \left(\{p \in X\text{on}A \cap^{*'} X\text{on}B \mid N(p;S) \neq \emptyset\}\right). \end{aligned}$$

In the theorem, $N(p;S)$ is the *neighborhood* of the point $p \in S$. The neighborhood is defined as the intersection of S and an open ball, $B(p,r)$, with radius r and center p, namely

$$N(p;S) = S \cap B(p,r), \text{ for small } r > 0.$$

Theorem 2 forms the basis of the membership classification used in regularized intersection. For the regularized union and difference, analogous theorems can be formulated. Given a CSG tree S, the following algorithm evaluates the membership classification function for a candidate set X:

```
M[Candidate set X, CSG tree S]
    if S is a primitive then
        return(prim-M(X, S))
    else
        return(combine(M[X, left(S)], M[X, right(S)], operation(S)))
```

Prim-M is a classification function based on the known set of primitive objects, and classifies X with respect to the primitive S. *Combine* merges the classification results of the two subtrees according to the set operation *operation*(S), as prescribed by Theorem 2 and its variations for the other boolean operations.

The membership classification algorithm is used to construct the equivalent B-rep of a CSG tree, by a *generate-and-test* paradigm [32]. In the incremental version of the algorithm, the B-reps of the primitives are constructed and are passed up the tree to the root. At each internal node, the B-reps are classified and combined. In a single-phase algorithm, the faces, or the support surfaces of all the primitives are classified against the entire CSG tree to obtain the faces of the final B-rep and are then assembled. Intermediate B-reps are not constructed in this case [2].

2. SET OPERATIONS ON BOUNDARY REPRESENTATIONS

For computing the union, intersection or difference of two solids in B-rep, we need to find the intersection curves of their boundaries on both solids and must classify the faces of both B-reps to determine which ones are inside, outside or on the boundary of the other solid. With the classification, the resulting B-rep can be constructed. We can perform the boolean operations by the following four step algorithm, where A and B are the input B-reps:

Boundary Merging: Fragment B-rep A so that no face or edge penetrates or grazes B, and conversely fragment B-rep B so that no face or edge penetrates or grazes A.

Boundary Classification: Classify the faces of both A and B to obtain an eight-way classification. The set of faces of each B-rep is partitioned into four classification sets

$$\begin{aligned} F(A) &= A\text{out}B \cup A\text{in}B \cup A\text{with}B \cup A\text{anti}B, \\ F(B) &= B\text{out}A \cup B\text{in}A \cup B\text{with}A \cup B\text{anti}A. \end{aligned}$$

corresponding to the faces that are outside, inside, on the boundary with same orientation and on the boundary with an orientation opposite to the other solid.

Construction: Construct the resulting B-rep according to the operation by either merging the two B-reps and removing unused faces, or by copying the needed faces from both B-reps to form a new B-rep. The needed faces are the faces of exactly three of the eight classification set as shown in Table 2.

	Aout$B$	Ain$B$	$A$with$B$	Aanti$B$	Bout$A$	Bin$A$	$B$with$A$	Banti$A$
$A \cup^* B$	$\oplus$		$\oplus$		$\oplus$			
$A \cap^* B$		$\oplus$				$\oplus$	$\oplus$	
$A -^* B$	$\oplus$			$\oplus$		$\ominus$		
$B -^* A$		$\ominus$			$\oplus$			$\oplus$

Table 2: Faces Needed In Set Operation Indicated by $\oplus$ and $\ominus$. Face Orientation Must Be Reversed for $\ominus$

Topological Reduction: Apply topological reduction on the resulting B-rep to form maximally connected faces. Adjacent coplanar faces and adjacent collinear edges are merged.

Exactly how these four steps are implemented varies in the proposed algorithms. Two algorithms follow the four steps exactly, the one by Laidlaw, Trumbore and Hughes [33] and the one by Mäntylä [4]. In Laidlaw's algorithm, the faces of the two objects that penetrate the other object are subdivided. Classification of the faces is performed by casting a ray from a face of one object through the other object. Classifying each face separately is avoided by first grouping together all adjacent faces that do not penetrate the boundary. After classification, faces that do not contribute to the resulting object are removed, and the remaining faces of the two objects are glued to form the new object. The representation assumes convex faces which somewhat simplifies the face/face intersection algorithm. Mäntylä's set operation algorithm allows arbitrary polygonal faces and uses a vertex-neighborhood computation to classify the faces [14].

Robust implementations of the boolean set operators on B-reps are difficult to achieve with finite-precision arithmetic [34]. Typically, the source of the problem is that different numerical computations may imply related geometric facts, such as incidence or nonincidence. In this situation, two separate computations may lead to contradictory conclusions, and this entails failure of the algorithm. For a survey of this problem and approaches to solving it see [35], or [2, Chapter 4].

3. SET OPERATIONS ON OCTREES

The union, difference and intersection operations are easy to implement for region octrees, by a coordinated traversal of both trees. Figure 8 shows a two-dimensional example.

Boolean operations on extended octrees and polytrees are straightforward extensions of the region octree algorithms. Suppose we want to compute the

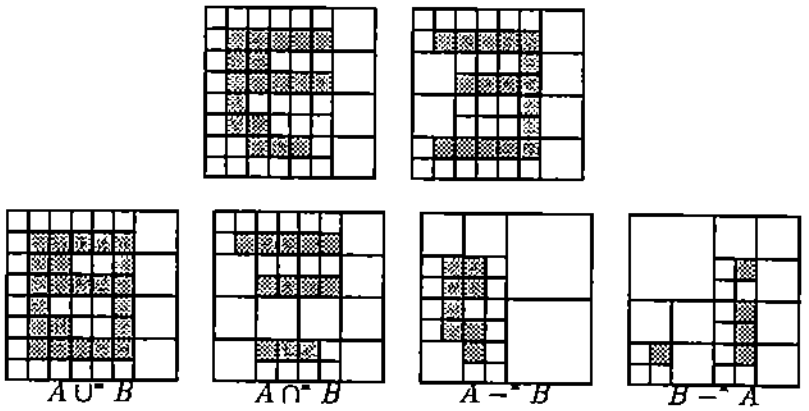


Figure 8: Two Quadtrees and Their Union, Intersection, and Differences

intersection of two region octrees. The two input trees are simultaneously traversed, and a new tree is constructed bottom up, combining corresponding nodes of the input trees. Note that it may be necessary to split nodes in order to obtain compatible levels of detail. For the extended octrees and polytrees, the algorithms must take into account the three additional node types, namely, the vertex, the edge, and the face nodes, as shown in Table 3. The intersection of two face regions can result in a black node, a white node, a face node, an edge node, or a gray node. For example, suppose one of the nodes is a gray node and the other is a vertex node. Since at this level, it is not known how the vertex region interacts with the descendents of the gray node, the vertex node region must be subdivided into eight octants. This changes the problem from a gray/vertex pair to a gray/gray pair, at the cost of processing the eight child pairings recursively. Eventually, nodes representing regions of the same level of detail are reached, and are intersected. The intersection may require additional subdivision in case the geometric structure obtained is too complicated.

It is customary to implement only one binary operation, say regularized intersection, and to obtain the remaining operations by one intersection and several complement operations. The complement of an extended octree is computed by changing all the black nodes into white nodes, changing all the white nodes into black nodes, and changing the orientation of all the plane equations in the other nodes.

If the recursion reaches the limit depth, further recursion is abandoned and a *nasty* node is created. Nasty nodes are either left empty, or they are associated with a list of all the faces that intersect in it. The rationale of this step is that the regions represented are so small that referencing them is unlikely. In the event they are referenced, they can be evaluated further on demand.

$\cap^*$	White	Black	Face	Edge	Vertex	Gray
White	W	W	W	W	W	W
Black	W	B	F	E	V	G
Face	W	F	WFEG	WFEVG	WFEVG	G
Edge	W	E	WFEVG	WFVG	WEVG	G
Vertex	W	V	WFEVG	WEVG	WVG	G
Gray	W	G	G	G	G	G

Table 3: Possible Node Types For Intersection Using Octree Representation

III. MULTIDIMENSIONAL SPACE PARTITIONING

We have reviewed three schemas for representing solids: boundary representations, cell decomposition representations, and constructive solid geometry. Each has certain advantages over the others, and many geometric modeling systems improve efficiency of their operations by maintaining solids in several different representations, possibly converting from one representation to another when appropriate. Systems that use explicit representations frequently use a B-rep as the primary representation and add some volume-based information to speed up access to parts of the B-rep. In this section, we present as an alternative a unified representation that combines a B-rep with a volume-based representation, called the *B-rep index*, and show how to implement some common geometric operations using this unified representation.

The *B-rep index* is an extension of the BSP tree [23]. A d-dimensional BSP tree hierarchically decomposes space into d-dimensional convex regions. Given a solid, the root node of a BSP tree represents the entire space while the leaves represent regions that are either completely inside or completely outside the solid. Each internal node contains an oriented cut plane that divides the region represented by the node into two open subregions, one above and one below the cut plane. This recursive decomposition of d-dimensional space is sufficient to uniquely represent polyhedral solids. One of the many operations that the BSP tree can support is the classification of a point in relation to a solid, that is, whether the point lies inside, on the boundary or outside the solid. However, since the BSP tree decomposes only the d-dimensional space and does not partition the boundary of the solid which is of lower dimension, the BSP tree cannot provide detailed information about the boundary. Yet many geometric modeling problems including Boolean set operations, orthogonal grid generation, and collision detection and analysis need to retrieve this boundary information.

The B-rep index overcomes the deficiency of the missing boundary information by extending the BSP tree recursively. Briefly, the hyperplanes separating the d-dimensional open regions are decomposed recursively. In the case of solids in 3-space, therefore, space is decomposed into three, two, one and zero-dimensional open regions. The partition is represented by a ternary tree, called the *multidimensional space partitioning* (MSP) tree, in which the middle subtrees represent the lower-dimensional regions contained in the dividing hyperplanes, whereas the left and the right subtrees represent the open regions above and below them.

In Section III.A we give an algorithm for constructing the B-rep index. The constructed MSP tree can be compressed, as described in Section III.B. The point/solid and the line/solid classification problems are solved both robustly and efficiently in Section III.C. In the remaining sections we give solutions to three common problems, based on the MSP tree. The first is orthogonal grid generation used in PDE solving; the second is static collision detection between solids; and the third problem is how to perform Boolean set operations.

A. CONSTRUCTING THE B-REP INDEX

We divide d-dimensional space by oriented $(d-1)$-dimensional hyperplanes defined by equations of the form

$$a_0 + a_1 x_1 + \ldots + a_d x_d = 0$$

with real coefficients a_i. Unlike the BSP tree, however, each hyperplane is recursively decomposed by $(d-2)$-dimensional hyperplanes

$$b_0 + b_1 y_1 + \ldots + b_{d-1} y_{d-1} = 0$$

which in turn are decomposed using $(d-3)$-dimensional hyperplanes and so on. Note that this formulation entails a coordinate change. In the B-rep index, the coordinate change is avoided and all lower dimensional hyperplanes are represented implicitly as the intersection of certain $(d-1)$-dimensional hyperplanes, that are on the path from the root to the current node in the tree.

In the 3-dimensional case, we orient the plane

$$ax + by + cz + d = 0$$

by the convention that the half space in the direction (a, b, c) is considered *above* the plane. We denote an oriented plane, the half space above it, and the half space below it by P, P^+, and P^-, respectively. Each internal node n of the tree represents a region $R(n)$, and specifies a plane, $P(n)$, that intersects $R(n)$. The three children of the node represent the subregions of $R(n)$ that

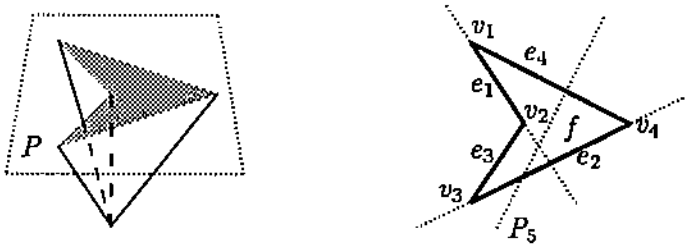


Figure 9: Top Face of Left Solid Partitioned by Cut Planes on Right. Plane P_i Contains Edge e_i.

lie above, on, and below $P(n)$, and are referred to as ABOVE(n), ON(n), and BELOW(n), respectively. Thus,

$$\begin{aligned} R(\text{ABOVE}(n)) &= R(n) \cap P(n)^+ \\ R(\text{ON}(n)) &= R(n) \cap P(n) \\ R(\text{BELOW}(n)) &= R(n) \cap P(n)^- \end{aligned}$$

If the dimension, $\dim(n)$, of $R(n)$ is d, then the regions $R(\text{ABOVE}(n))$ and $R(\text{BELOW}(n))$ are also d-dimensional, but the region $R(\text{ON}(n))$ is $(d-1)$-dimensional.

Consider the support plane, P, of the top face of the solid shown in Figure 9. The partitioned support plane and the corresponding subtree of the B-rep index are shown in Figure 10.

The B-rep index is constructed in two steps. The first step creates the B-rep index by constructing the MSP tree and attaching it to the B-rep. The second step optionally compresses the MSP tree by removing internal nodes with redundant cuts. This is useful when the B-rep index is used for large classification problems such as grid generation or collision detection.

The B-rep index partitions space such that each region contains at most one vertex, one edge or one face. Regions not containing boundary elements must be entirely inside or outside the solid. A recursive function with three arguments constructs the partition. The first argument is an entity set. The second argument is the dimension of the region, and the third is a set of support planes of the region, initially empty. Intuitively, the entity set contains boundary elements that will be indexed by the subtree to be constructed at the current node, whereas the set $\mathcal{P}$ records the cut planes on the path from the root to the current node whose intersection contains the set $\mathcal{X}$.

The entity set contains triples

$$[x, (p_0, \ldots, p_{n-1}), (\ell_0, \ldots, \ell_{n-1})],$$

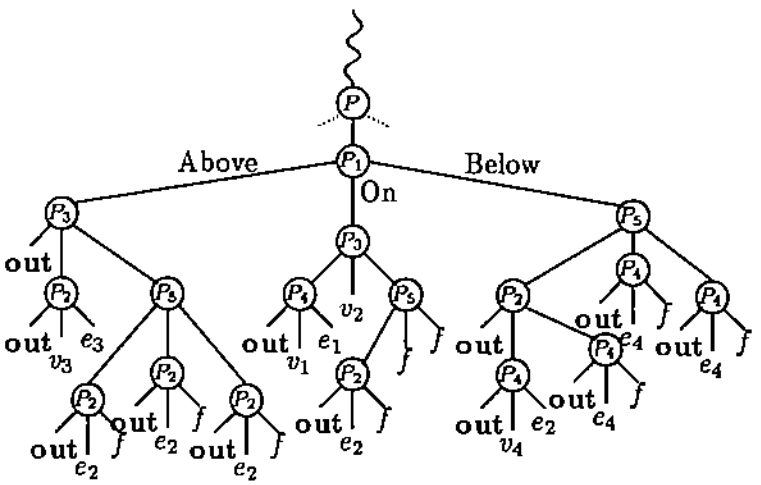


Figure 10: Subtree For Top Face of Solid in Figure 9, With 17 Nodes

where x is a face, edge, or vertex. If x is a face, the second component is a cyclic list of n vertices bounding the face, and the third component is a list of lines containing consecutive pairs of points in the cyclic list. Note that x can be a subpolygon of a face of the B-rep. If x is an edge e, the triple is $[e,(p_0,p_1),(\ell)]$ where p_0 and p_1 bound the edge and ℓ is the line containing p_0 and p_1. Again, e could be a segment of an edge of the B-rep. For a vertex v, the triple is $[v,(p),()]$, where p gives the coordinates of the vertex. Note that (p_0,p_1) could be two closely-spaced points on a longer edge. Thus explicitly representing the support line increases accuracy when constructing support planes through the two points.

CreateBrepIndex(**Set of** triples $\mathcal{X}$, Dimension d, **Set of** planes $\mathcal{P}$)
  if $|\mathcal{X}| > 1$ **then** {
    let $P \leftarrow$ ChooseCutPlane$(\mathcal{X}, d, \mathcal{P})$
    $(\mathcal{X}_a, \mathcal{X}_o, \mathcal{X}_b) \leftarrow$ Cut$(\mathcal{X}, P)$
    $i_a \leftarrow$ **if** $\mathcal{X}_a = \emptyset$ **then OUTSIDE else** CreateBrepIndex$(\mathcal{X}_a, d, \mathcal{P})$
    $i_o \leftarrow$ CreateBrepIndex$(\mathcal{X}_o, d-1, \mathcal{P} \cup \{P\})$
    $i_b \leftarrow$ **if** $\mathcal{X}_b = \emptyset$ **then INSIDE else** CreateBrepIndex$(\mathcal{X}_b, d, \mathcal{P})$
    return Node(P, i_a, i_o, i_b)
  } **else return** x **given that** $\mathcal{X} = \{[x,(p),()]\}$

The function finds an oriented cut plane, P, that contains entity x that occurs in a triple in $\mathcal{X}$ and is such that P does not already appear in $\mathcal{P}$. If x

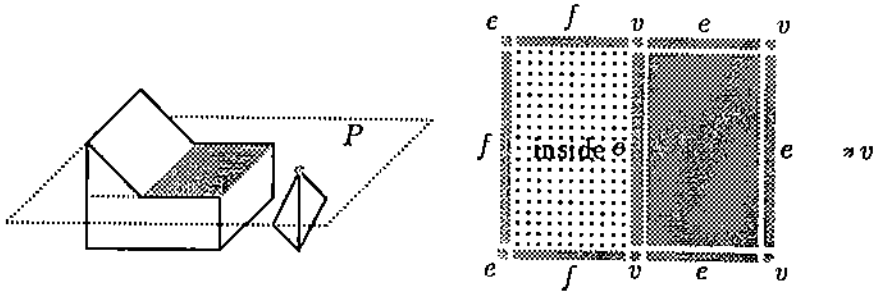


Figure 11: Set $\mathcal{X}$ With 15 Entities, Resulting From Cut With Plane P Through Solids On Left.

is a face, then the cut plane is the support plane of the face. If x is an edge or a vertex, the cut plane is the support plane of one of the adjacent faces of x. If there is more than one adjacent face, and this may be the case for a vertex or a nonmanifold edge, then a plane is chosen that maximizes transversality with the planes in $\mathcal{P}$. If the support plane of every adjacent face is already in $\mathcal{P}$, then P is chosen to be perpendicular to the planes in $\mathcal{P}$.

In the left part of Figure 11, for example, the two solids shown are cut by a plane P that contains the shaded face of the left solid. The cut generates 15 entities contained in the plane. Seven of them are open line segments and these will be further isolated by other cut planes.

All entities of $\mathcal{X}$ that cross the cut plane P are split by the intersection with P. The resulting subentities are then put into the sets $\mathcal{X}_a$, $\mathcal{X}_o$, and $\mathcal{X}_b$ according to whether they lie above, on or below P. A face crossing P is split into faces lying above and below, and edges lying on P. An edge crossing P is split into the edges above and below and the vertex on P. Figure 12 illustrates the fragmentation with plane P of a face, its 16 edges and its 16 vertices. After the cut, the region above P consists of two faces, six edges and four vertices. The region on P consists of seven vertices and six edges. The region below P consists of three faces, nine edges, and six vertices.

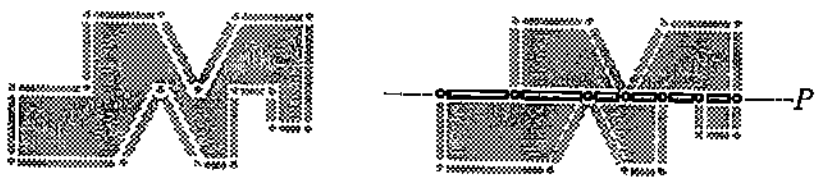


Figure 12: Fragmentation of a Face, Bordering Edges and Vertices by Cut Plane P.

B. COMPRESSING THE B-REP INDEX

After the B-rep index has been so created, it can be used directly. However, the B-rep index can be compressed considerably, and doing so speeds up accessing it. Compression proceeds in three steps:

1. Locate and eliminate all redundant cut planes.

2. Reorder some cuts to uncover and eliminate additional redundant cuts. By restructuring the tree so that the cut planes are queried in a different order, other cut planes may become redundant and the corresponding nodes eliminated.

3. If the tree is badly unbalanced (for instance in the case of convex objects) rebalance the tree by introducing a few well-chosen redundant cuts.

We give tree rewrite rules that compress the tree and illustrate the rules by compressing the tree of Figure 10. In prefix notation, the subtree rooted in n is written as $P(a,b,c)$ where P is the cut plane at n, and a, b, and c denote the above, on, and below subtrees of n, respectively.

The first rule states that any extended region can be cut redundantly by a plane that is transversal with that region. For $P \cap R(a) \subset R(a)$ and $\dim(a) > 0$,

$$a \Rightarrow P(a,a,a). \qquad (R1)$$

The converse rule states that this redundant cut can be removed

$$P(a,a,a) \Rightarrow a. \qquad (R2)$$

If the cut plane orientation is reversed, then the order of the children must be reversed also:

$$P(a,b,c) \Leftrightarrow -P(c,b,a). \qquad (R3)$$

Two transversal cuts of a given region commute as follows:

$$P_1(P_2(a,b,c),P_2(d,e,f),P_2(g,h,i)) \Leftrightarrow P_2(P_1(a,d,g),P_1(b,e,h),P_1(c,f,i)). \qquad (R4)$$

The above rules do not need any geometric information. In contrast, the next rule requires knowing the orientation of the cut planes and generalizes AVL tree rotation [36]. Given that the cut plane of a child node does not intersect the cut plane of the parent node within the parent region, the order of the cuts can be interchanged in a way that depends on the relative orientation of the cut planes.

Given that $P_1 \cap P_2 \cap R(n) = \emptyset$,

$$P_1(P_2(a,b,c),d,e) \Rightarrow \begin{cases} P_2(a,b,P_1(c,d,e)) & \text{if } R(e) \subset P_2^- \\ P_2(P_1(a,d,e),b,c) & \text{if } R(e) \subset P_2^+, \end{cases} \qquad (R5a)$$

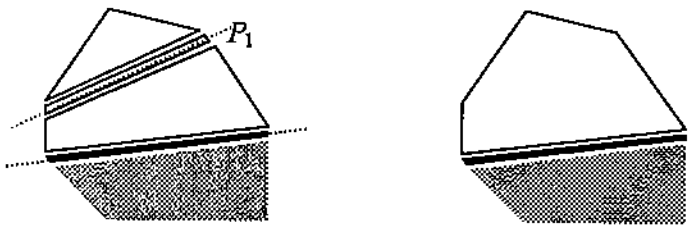


Figure 13: Elimination of Redundant Cut P_1 by Rule 7.

and

$$P_1(a,b,P_2(c,d,e)) \Rightarrow \begin{cases} P_2(P_1(a,b,c),d,e) & \text{if } R(a) \subset P_2^+ \\ P_2(c,d,P_1(a,b,e)) & \text{if } R(a) \subset P_2^-. \end{cases} \qquad (R5b)$$

From the given rules additional rules can be derived. For instance,

$$\begin{aligned} P_1(P_2(a,b,c),c,P_3(c,d,e)) &\Rightarrow P_2(a,b,P_3(c,d,e)) \\ P_1(P_2(a,b,c),c,P_3(c,d,e)) &\Rightarrow P_3(P_2(a,b,c),d,e), \end{aligned} \qquad (R6)$$

and

$$P_1(P_2(a,b,c),c,c) \Rightarrow P_2(a,b,c). \qquad (R7)$$

Both rules have additional variations that are derived using Rule (R3).

To illustrate the use of the rewrite rules, we compress the B-rep index of Figure 10. When a subtree is changed, the rule that applies is indicated in the superscript. We begin the compression by noting that P_5 is redundant, and remove it by propagating it to the bottom of the tree where it is removed by Rule (R2).

$$\begin{aligned} & P_1(P_3(\mathbf{out},P_2(\mathbf{out},v_3,e_3), \\ & \qquad P_5(P_2(\mathbf{out},e_2,f),P_2(\mathbf{out},e_2,f),P_2(\mathbf{out},e_2,f))^{\mathrm{R4}}), \\ & \quad P_3(P_4(\mathbf{out},v_1,e_1),v_2,P_5(P_2(\mathbf{out},e_2,f),f,f)^{\mathrm{R7}}), \\ & \quad P_5(P_2(\mathbf{out}^{\mathrm{R1}},P_4(\mathbf{out},v_4,e_2),P_4(\mathbf{out},e_4,f))^{\mathrm{R4}}, \\ & \qquad P_4(\mathbf{out},e_4,f),P_4(\mathbf{out},e_4,f))) \\ = \; & P_1(P_3(\mathbf{out},P_2(\mathbf{out},v_3,e_3), \\ & \qquad P_2(P_5(\mathbf{out},\mathbf{out},\mathbf{out})^{\mathrm{R2}},P_5(e_2,e_2,e_2)^{\mathrm{R2}},P_5(f,f,f)^{\mathrm{R2}})), \\ & \quad P_3(P_4(\mathbf{out},v_1,e_1),v_2,P_2(\mathbf{out},e_2,f)), \\ & \quad \ldots) \\ = \; & P_1(P_3(\mathbf{out},P_2(\mathbf{out},v_3,e_3),P_2(\mathbf{out},e_2,f)), \\ & \quad P_3(P_4(\mathbf{out},v_1,e_1),v_2,P_2(\mathbf{out},e_2,f)), \\ & \quad P_4(\mathbf{out},P_2(\mathbf{out},v_4,e_4),P_2(\mathbf{out},e_2,f))) \end{aligned}$$

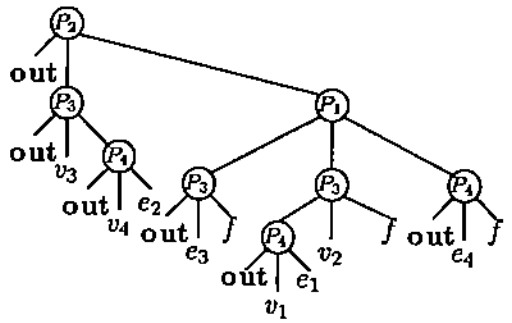


Figure 14: The Compressed B-rep Index of B-rep Index of Figure 10.

Next we observe that the subtree $P_2(\mathbf{out}, e_2, f)$ appears multiply as a cut of P_1, and so we exchange the order of the cuts by moving P_2 up and exchanging with Rule R4.

$$
\begin{aligned}
& P_1(P_3(\mathbf{out}^{\mathrm{R1}}, P_2(\mathbf{out}, v_3, e_3), P_2(\mathbf{out}, e_2, f))^{\mathrm{R4}}, \\
& \quad P_3(P_4(\mathbf{out}, v_1, e_1), v_2, P_2(\mathbf{out}, e_2, f))^{\mathrm{R5}}, \\
& \quad P_4(\mathbf{out}^{\mathrm{R1}}, P_2(\mathbf{out}, v_4, e_4), P_2(\mathbf{out}, e_2, f))^{\mathrm{R4}}) \\
= \; & P_1(P_2(\mathbf{out}, P_3(\mathbf{out}, v_3, e_2), P_3(\mathbf{out}, e_3, f)), \\
& \quad P_2(\mathbf{out}, e_2, P_3(P_4(\mathbf{out}, v_1, e_1), v_2, f)), \\
& \quad P_2(\mathbf{out}, P_4(\mathbf{out}, v_4, e_2), P_4(\mathbf{out}, e_4, f)))^{\mathrm{R4}} \\
= \; & P_2(P_1(\mathbf{out}, \mathbf{out}, \mathbf{out})^{\mathrm{R2}}, \\
& \quad P_1(P_3(\mathbf{out}, v_3, e_2), e_2, P_4(\mathbf{out}, v_4, e_2))^{\mathrm{R6}}, \\
& \quad P_1(P_3(\mathbf{out}, e_3, f), P_3(P_4(\mathbf{out}, v_1, e_1), v_2, f), \\
& \quad\quad P_4(\mathbf{out}, e_4, f))) \\
= \; & P_2(\mathbf{out}, P_3(\mathbf{out}, v_3, P_4(\mathbf{out}, v_4, e_2)), \\
& \quad P_1(P_3(\mathbf{out}, e_3, f), P_3(P_4(\mathbf{out}, v_1, e_1), v_2, f), \\
& \quad\quad P_4(\mathbf{out}, e_4, f))).
\end{aligned}
$$

The original B-rep index tree consisted of 17 internal nodes; the final tree has only eight nodes, as shown in Figure 14. In this simple example, we achieved a reduction in the size of the tree by more than fifty percent. Vaněček has shown that for a B-rep with v vertices, e edges and f faces its B-rep index has at least $v + e + f$ internal nodes, and $1 + 2(v + e + f)$ leaf regions [37]. In 2D, the number of internal nodes is at least $v + e$, and the number of leaf regions is at least $1 + 2(v + e)$. Our 2D example has four vertices and four edges, so the compressed tree with eight nodes is minimal.

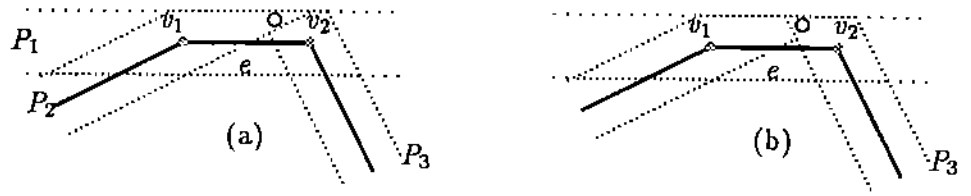


Figure 15: Anomaly Caused by a Tolerance That is Too Large.

C. POINT AND LINE CLASSIFICATION

Most geometric computations reduce to point/solid and line/solid classification. This includes boundary merging, collision detection, grid generation and ray casting. The point/solid classification determines whether a point is inside, outside or on the boundary of a solid. With the B-rep index, if the point lies on the boundary of the solid, the boundary entity on which it lies is also determined. The line/solid classification partitions the line where it penetrates or touches the boundary.

Classifying a point with the B-rep index is simple. Starting from the root, check the position of the point relative to the cut planes of the internal nodes, and proceed through the corresponding subtrees until a leaf is reached. At the leaf the point classification is known.

---

classify(Point p, Node n)
  while n is an internal node **do** {
    Let d be the signed distance of p to $P(n)$

$$n \leftarrow \begin{cases} \text{ABOVE}(n) & \text{if } d > \epsilon \\ \text{BELOW}(n) & \text{if } d < -\epsilon \\ \text{ON}(n) & \text{otherwise} \end{cases}$$

  }
  return n

Since an exact answer to "a lies on b" cannot be given in floating point arithmetic, we compute an approximate answer that depends on a tolerance $\epsilon > 0$.

The tolerance can be viewed as a thickness of the boundary. Since using exact arithmetic is not practical, the tolerance will affect the classification. Recall that a given B-rep does not have a unique MSP tree. The tree structure depends on the order and choice of the cutting planes. The example of Figure 15

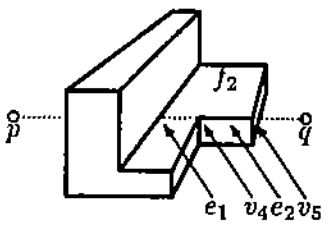


Figure 16: Line-Segment/Solid Classification.

shows a boundary fragment with two different MSP subtrees. Figure 15(a) is

$$P_1(\ldots, P_2(\mathbf{out}, v_1, P_3(\mathbf{out}, v_2, e)), \ldots),$$

and Figure 15(b) is

$$P_1(\ldots, P_3(\mathbf{out}, v_2, P_2(\mathbf{out}, v_1, e)), \ldots),$$

and these are equivalent by Rule R5b. Now consider the point indicated by the circle in the figures. Using tree (a), the point is found to be coincident with v_1, but using tree (b), the point is found to be coincident with v_2. This shows that equivalent MSP trees for a given solid do not classify points the same way. This discrepancy is caused by a tolerance that is too large relative to the separation between two entities. In particular, as the tolerance changes, the classification result may change. It is possible that a point classified as coincident with a vertex, at one tolerance, is found coincident with another vertex at a different tolerance.

Consider now the line/solid classification problem and refer to Figure 16 for an example. The classification of the line segment $\overline{pq}$ should have the result

$$[\mathbf{out}, (f_1, p), \mathbf{in}, (e_1, q), f_2, (v_4, p_4), e_2, (v_5, p_5), \mathbf{out}], \tag{1}$$

where p, q, p_4, p_5 are the points at which the line penetrates the corresponding entities, and the v's, e's and f's are the vertices, edges and faces of the B-rep.

An intuitive way to classify the line is to pass the line segment down the tree, split the line segment whenever it crosses $P(n)$ at a node n, and classify the portions above, on, and below recursively. This approach, however, is problematic when the line segment forms a very small angle with a cutting plane. Figure 17 shows a line segment, ℓ, close to vertex v. Although ℓ can be arbitrarily close to v, the line segment crosses the plane P_1 far away from v, because of the small angle between P_1 and the line. So the point of intersection on P_1 and the two subsegments fall into the regions outside the solid. This is not intuitive, partly because the thickness of the $\pm\epsilon$ region around the planes

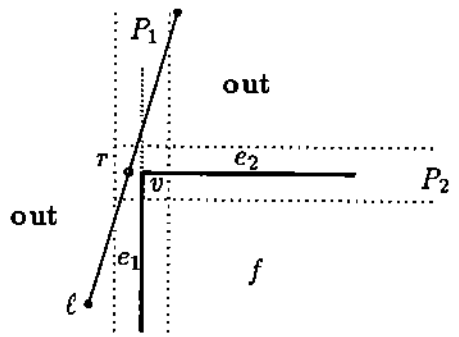


Figure 17: Part of Face With Line Segment ℓ Crossing at v Within Tolerance.

has not been well accounted for in the above approach. A better classification is obtained when the interval of ℓ passing through the $\pm\epsilon$ regions of P_1 and P_2 is computed. The closest point r on ℓ to v is then computed by projecting v onto ℓ.

The line classification thus proceeds by passing the original interval $[t_1, t_2]$ from the top to the bottom of the B-rep index in phase one, followed by a second phase in which the final line partition is determined. In phase one, the interval $[t_1, t_2]$ reaches the node n and is partitioned as either $[t_1, t_-, t_+, t_2]$ or $[t_1, t_+, t_-, t_2]$ depending on the orientation of the plane $P(n)$. The three subintervals are processed in the corresponding subtrees. An interval is given a preliminary classification at the leaf node it reaches.

Next, the sequence of intervals is reduced by merging adjacent intervals with common classifications and by isolating the boundary penetrations. In the example of Figure 17, after step 1, we have the classification

$$[\mathbf{out}, e_1, v, \mathbf{out}, \mathbf{out}]$$

After merging adjacent equal classifications, this sequence is reduced to

$$[\mathbf{out}, e_1, v, \mathbf{out}]$$

The classification corresponds to the sequence of dimensions d_i, $[3, 1, 0, 3]$. We replace each maximal bitonic sequence $d_i > \cdots > d_j < \cdots < d_k$ with the triple d_i, d_j, d_k. That is, all intervening intervals and their classification are dropped. Then the sequence is reduced to $[3, 0, 3]$ with the associated classification $[\mathbf{out}, v, \mathbf{out}]$. The reduced classification is used to compute the final interval partition, by computing the intersection of the line segment with the remaining faces, edges, and vertices. Note that this may require projecting say a vertex onto the line segment.

The reduction above does not address how to reduce classification sequences that contain adjacent, different entities of the same dimension. Consider line

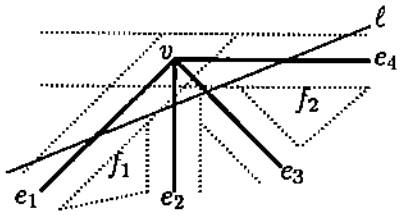


Figure 18: Ambiguous Line Classification With Oversized Tolerance.

ℓ of Figure 18. Its classification after step one is

$$[\mathbf{out}, e_1, e_2, e_3, e_4, \mathbf{out}]. \tag{2}$$

The correct classification is either

$$[\mathbf{out}, v, \mathbf{out}],$$

or

$$[\mathbf{out}, e_1, f_1, e_2, \mathbf{out}, e_3, f_2, e_4, \mathbf{out}].$$

From Sequence (2) we cannot derive the two classifications nor can we determine which one is correct. In this situation, the line has to be reclassified with a different tolerance. In our example, a larger tolerance yields the first classification, whereas a smaller tolerance yields the second.

In many applications, both point/solid and line/solid classification are done. Consider the following example, and refer to Figure 17. The segment ℓ enters the tolerance region of edge e_1 from the left, at a point q. There is a part of ℓ adjacent to q that is in the tolerance region of e_1, but not in the tolerance region of v. Let p be a point on that part of ℓ. Classified as a point, p is found to be on the edge e_1. When ℓ is classified, however, the final segment classification will be $[\mathbf{out}, v, \mathbf{out}]$, thus p will be determined to be **out**, a contradictory classification, even though both classifications used the same tolerance. In problems such as grid generation, we account for this possibility.

D. ORTHOGONAL GRID GENERATION

We generate orthogonal grids using line/solid classification. Orthogonal grids are used by finite-differencing solvers for partial differential equations (PDE), for example in ELLPACK [38], a system for solving elliptic PDEs. The grids can be regular, with uniform distance between grid points, or adaptive, with grid points that are denser in some parts of the domain.

A regular three-dimensional grid can be specified by two extreme points of the grid's bounding cuboid, and by the number of grid planes n_x, n_y, and

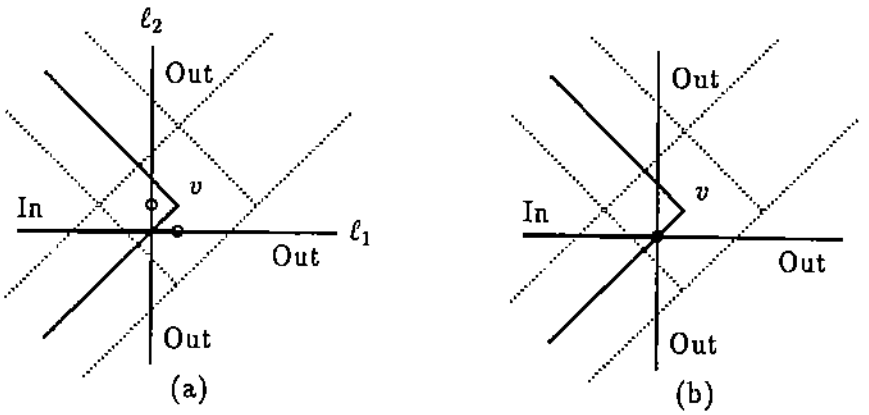


Figure 19: Grid Line Classification, (a) Before and (b) After, Consistent Classification At Grid Points.

n_z along each major axis. The intersection of the grid planes defines $n_x n_y + n_y n_z + n_z n_x$ grid lines and $n_x n_y n_z$ grid points. Our task is to classify all grid points and determine where the grid lines cross the boundary between grid points. It is clear from the previous section that grid points and grid lines should not be classified independently. Instead, we classify only the grid lines and process local inconsistencies, thus obtaining a consistent grid point and grid line classification.

Consider the two-dimensional example of Figure 19(a) in which the two grid lines ℓ_1 and ℓ_2 were classified independently. The intersections of ℓ_1 and ℓ_2 are the grid point p that is near to the vertex v. On ℓ_1, p is classified as in; on ℓ_2, it is classified as **out**, because of the projection computation when segmenting the lines. Classified as a point, p would be v. By shifting them to the grid point, the two vertex projections are made coincident, giving p a consistent classification; Figure 19(b). This approach requires that the tolerance regions are substantially smaller than the minimum grid point separation. Otherwise, there could be several grid points to which to shift the two projections of v. The classification of grid points that lie within tolerance of an edge or a face is corrected in the same way.

The cost of classifying the grid lines is $O((n_x n_y + n_y n_z + n_z n_x) L(m))$, where $L(m)$ is the cost of the line/solid classification in a MSP tree with m nodes. The cost of correcting the grid point classifications by shifting the vertex projections is $O(n_x n_y n_z)$. The method has been implemented; [26].

E. COLLISION DETECTION

Our approach to orthogonal grid generation is also used for collision detection and analysis of moving solid objects, [27]. The collision problem arises in

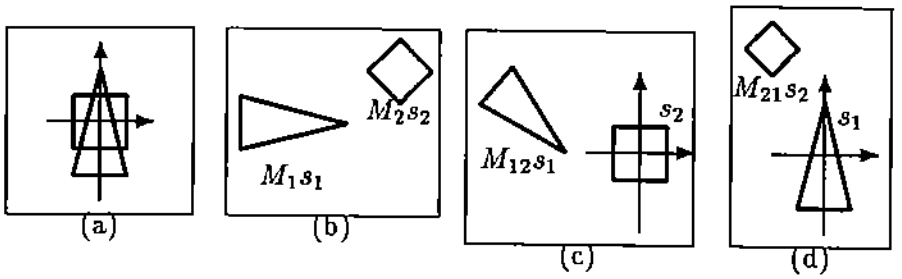


Figure 20: (a) Solids s_1 and s_2 With Respect to Local Coordinates. b) s_1 and s_2 Placed in Global Frame of Reference. c) s_1 Mapped To Local s_2-Frame of Reference. d) s_2 Mapped To Local s_1-Frame of Reference.

computer simulations of physical systems, based on rigid-body dynamics; for example, in the Newton system [39,40]. Here, objects of arbitrary shape are permitted, and their motion cannot be predicted in advance, so that collisions can happen anywhere and at any time. Moreover, objects may be in contact with each other for some time, and such contacts begin and end in ways that depend entirely on the dynamics of the system, and again cannot be predicted beforehand. So, the positional relationships of all objects must be analyzed.

In the Newton system, the dynamics evaluation inquires at certain time instances whether two bodies are in contact, and if so, a geometric analysis of contact locality is required. Collision detection and analysis can be reduced to a static problem in Newton as follows: All solids have a local frame of reference centered at the mass center. Objects are placed in relation to each other by a 4×4 transformation matrix M that positions and orients the local frame with respect to a global frame of reference. As objects move, the matrix M is changed accordingly.

Consider two solids s_1 and s_2 and their associated transforms M_1 and M_2. Then solid s_1 can be related to the local frame of reference of s_2 by the transformation

$$M_{12} = M_2^{-1} \cdot M_1$$

See also Figure 20. Similarly, $M_1^{-1} \cdot M_2$ relates s_2 to the local frame of s_1. We classify each edge of s_1 against s_2 after transforming the edges with M_{12}, and classify each edge of s_2, after transformation by M_{21}, against s_1. Note that this strategy does not require changing the B-rep index of either solid, irrespective of their motion over time. The method has been implemented in the Newton system

As in the case of orthogonal grid generation, floating-point arithmetic causes robustness problems. Consider an edge of s_1 touching an edge of s_2 at an angle. Projecting one edge onto the other determines a pair of contact

points, one point on each edge. When classifying the edges of s_1 we obtain one pair, and when classifying the edges of s_2 we obtain another pair. The points in the two pairs should coincide, but often they do not. We resolve this discrepancy by averaging corresponding points. It is also possible that the first classification reports two touching edges, whereas the second classification does not find this crossing. In this case, we give priority to the contact detection. These strategies have been very successful in the Newton system. Clearly, such heuristics ameliorate robustness problems but do not completely eliminate them. Indeed, solving the robustness problem completely is extremely costly, [2, Chapter 4]. Thus, in practice it is often acceptable to use incomplete solutions as long as they eliminate most robustness problems that arise.

F. BOOLEAN SET OPERATIONS WITH THE B-REP INDEX

We implement the boolean set operations using the B-rep index with the four-step algorithm of Section II.D.1. That is, we merge the boundaries, classify the faces, construct the result solid, and then compress its representation. Let A and B be the input solids. We could use the algorithm of Section III.A to construct a B-rep index of both B-reps together, obtaining an MSP tree in which the leaves have two classification fields, one for the region with respect to solid A, the other with respect to solid B. Since the B-reps of A and B are not explicitly subdivided, it is difficult to construct the B-rep index of the result. In particular, extracting the curves of intersection from the joint index would have to be done without explicit adjacency information.

A better approach is to construct (implicitly) a joint B-rep index of both input solids that explicitly subdivides the boundaries of A and B, so that the intersection curves on the surfaces can be extracted easier. Algorithm MergeBreps does this by maintaining separate entity sets, with $\mathcal{X}$ initially all boundary entities of A and $\mathcal{Y}$ all boundary entities of B. We intend to cross-classify entities, so that, ultimately, the boundary entities of A are classified as **in** or **out** with respect to B, or are classified by the coincident entity of B. Likewise, the entities of B should be cross-classified with respect to A. We associate with each entity x a classification field, $C(x)$, initially set to **unknown**. At the end, the classification will be known for coincident and intersection entities, but not for entities that are outside or inside the other solid.

Figure 21 shows two objects before and after boundary merging, with the dotted lines indicating coincident entities. Note that the merging step, constructing the joint MSP tree, subdivides both boundaries, but only in the vicinity of the intersection curves. In particular, the right edge of the box in the figure is not separated from its bounding vertices.

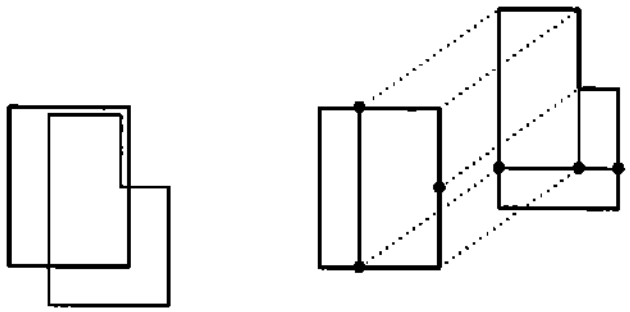


Figure 21: Objects Before and After Boundary Merging.

```
MergeBreps(Set of entities 𝒳, 𝒴; Dimension d; Set of planes 𝒫)
  if |𝒳| = 1 and |𝒴| = 1 then
    C(x) ← y, and C(y) ← x, where 𝒳 = {x}, and 𝒴 = {y}
  else {
    let P ← ChooseCutPlane(𝒳, 𝒴, d, 𝒫)
    (𝒳_a, 𝒳_o, 𝒳_b) ← Cut(𝒳, P)
    (𝒴_a, 𝒴_o, 𝒴_b) ← Cut(𝒴, P)
    if 𝒳_a ≠ ∅ and 𝒴_a ≠ ∅ then
      MergeBreps(𝒳_a, 𝒴_a, d, 𝒫)
    if 𝒳_b ≠ ∅ and 𝒴_b ≠ ∅ then
      MergeBreps(𝒳_b, 𝒴_b, d, 𝒫)
    if 𝒳_o ≠ ∅ and 𝒴_o ≠ ∅ then
      MergeBreps(𝒳_o, 𝒴_o, d − 1, 𝒫 ∪ {P})
  }
```

After the joint MSP tree T has been constructed, we must cross-classify the remaining entities. We do this by casting a ray as follows. Let x be an entity, say of A, whose classification is unknown, and which therefore is either **in** or **out** B. Note that x is attached to a leaf of T that represents a convex region, and that the parent's region is also convex. The parent's region must contain part of the boundary of B, say some entities including y. We cast a ray from an interior point of x to an interior point of y. Then only ray intersections with boundary elements in the parent's region are considered, because the region is convex, and these entities are indexed by the subtree rooted in the parent. Thus, classifying by ray casting is very efficient, since only a small part of the boundary must be considered.

After this second step, every entity x of either B-rep has the correct classification $C(x)$, which is either the labels **in**, or **out**, or a pointer to a vertex, edge,

or face of the other B-rep. Note that this step can be incorporated directly into the construction of the joint MSP tree. Doing so has the advantage that the joint MSP tree need not be explicitly represented. This is appropriate, because in many cases the joint MSP tree would be much larger than we would need for the result of the operation.

Having classified all boundary entities, we can now assemble the result B-rep destructively, by merging and pruning the two B-reps, or nondestructively, by copying the appropriate faces indicated in Table 2.

IV. CONSTRAINED CURVED SURFACE REPRESENTATIONS

A. BACKGROUND

The majority of systems computing with curved surfaces represent and manipulate *parametric surfaces*, i.e., surfaces that are constructed from *patches* defined by coordinate functions

$$\begin{aligned} x &= h_1(u,v) \\ y &= h_2(u,v) \\ z &= h_3(u,v) \end{aligned}$$

where the parameters u and v take values over a finite domain, for example the unit square $[0,1] \times [0,1]$, and the coordinate functions $h_i(u,v)$ are polynomials or ratios of polynomials in u and v. Customarily, the coordinate functions are represented in a special basis, for example the Bernstein-Bèzier basis, that admits a correlation between a net of *control points* in 3-space and the coefficients of the coordinate functions. This has the advantage of correlating the control points with the shape of the surface patch in a geometrically intuitive manner, and it also permits formulating continuity constraints between patches in an elegant way as conditions on the control points. The literature on this subject is vast, and we refer the reader to books such as [41,42,43] or to survey articles such as [44] for an entry into this subject. Parametric surfaces are not closed under some geometric operations such as offsetting.

Implicit algebraic surfaces provide an alternative to the parametric surface representation. An implicit algebraic surface is defined by an equation

$$f(x,y,z) = 0$$

where f is a polynomial in x, y and z. Algebraic surfaces are more general than parametric surfaces and are, in contrast, closed under most geometric operations, including offsetting. However, they lack a strong correlation between coefficient choice and resulting shape, necessitating different design

paradigms. Nevertheless, implicit surfaces are useful in geometric modeling, and [2] gives many details of their properties and describes techniques for interrogating them. Implicit algebraic surfaces include in principle all parametric surfaces, but not vice versa. Current techniques for converting from parametric to implicit form, and methods for parameterizing implicit algebraic surfaces that possess a rational parametric form, can be demanding symbolic computations [45]. Simple algorithms are known for *monoids*, a special class of surfaces that includes all quadratic surfaces. See [2] for details.

In certain applications, one would like to construct curved surfaces that satisfy prescribed constraints. In many cases, these surfaces are readily explained and informally specified in intuitive geometric terms. For instance, given two surfaces f and g, consider all points in space that have equal minimum distance from the given surfaces. Such points form the *equal-distance* or *Voronoi surface* of f and g. Another example is the *rolling-ball blend*, a blending surface between given surfaces f and g obtained as follows: Roll a sphere such that it maintains contact with both f and g, at all times. Considering the volume in space traversed by the sphere, the rolling ball blend is defined by the surface of that volume.

The basic property of these and similar other surface definitions is that the new surface to be defined can be expressed in terms of one or more base surfaces and a number of intuitive geometric constraints. Despite this conceptual simplicity, it is by no means a trivial undertaking to represent such surfaces in exact mathematical terms, in a manner suitable for computation. Often, neither implicit nor parametric surface representations can be found, and so we seek an alternative to these two representation schemata. The *dimensionality paradigm* provides a straightforward and mathematically rigorous method for exactly representing surfaces defined through constraints. The dimensionality paradigm simplifies representing complicated surfaces, and it permits generic algorithms for manipulating them. The method is also of interest because it is an enabling technology for constructing promising curved-solid representations such as the skeleton discussed later in Section V.

B. THE DIMENSIONALITY PARADIGM

We observe that the definition of complex constrained surfaces often simplifies when we consider the surface as the natural projection of a manifold in higher-dimensional space. The manifold is defined by a system of nonlinear equations in n variables, where $n > 3$. The extra dimensions may be point coordinates on the base surface(s), distances, or other quantities used to express the constraints that must be obeyed. The surface we want is then the natural projection of this manifold into a three-dimensional subspace.

When the base surfaces are algebraic, and this is normally the case in geometric and solid modeling, then the resulting system of equations can be

processed by a number of symbolic computation algorithms that eliminate all additional variables and arrive at a single implicit equation for the surface. Such an approach is normally intractable, for it routinely arrives at elimination problems that are well beyond what hardware and software can deliver in the foreseeable future. Worse yet, a number of elimination algorithms require exact arithmetic, further exacerbating the magnitude of the computation. Therefore, we will work directly with the system of equations.

We will demonstrate the definitional approach with an example. We have developed a number of algorithms for surfaces defined using the dimensionality paradigm. The algorithms are general and do not require knowledge of the geometric nature of the surfaces. Therefore, they will work unchanged on offsets, on blends, on equal-distance surfaces, and so on. Some algorithms are local in nature. That is, given a point on the surface, the algorithms will explore the surface in the vicinity of the point. Such algorithms can and have been globalized, by suitably embedding the exploration into a spatial grid that coordinates the computation. We will sketch the salient features of many of these algorithms also.

C. AN EXAMPLE DEFINITION

We consider the definition of an equal-distance surface as an example of the dimensionality paradigm. Assume that we are given two implicit surfaces $f(x,y,z) = 0$ and $g(x,y,z) = 0$. Using a declarative style, we can then describe the equal-distance surface as follows:

(i) Let $p = (x,y,z)$ be a point on the equal-distance surface. Moreover, let $p_f = (u_1, v_1, w_1)$ be a point at minimum distance from p on f, and let $p_g = (u_2, v_2, w_2)$ be a point at minimum distance from p on the surface g. Then:

(ii) The point p_f satisfies the equation of f, and the point p_g satisfies the equation of g.

(iii) The distance (p, p_f) is d and is equal to the distance (p, p_g).

(iv) The line $\overline{p, p_f}$ is normal to f at p_f.

(v) The line $\overline{p, p_g}$ is normal to g at p_g.

Note that Assertion (i) declares the names of nine variables, the coordinates of three points, whereas Assertions (ii)-(v) simply state the geometric relationships that these points must satisfy.

In order to obtain an equational representation of the equal-distance surface, we translate the Assertions (ii)-(v), using the variable names of (i). We obtain in sequence:

$$f(u_1, v_1, w_1) = 0 \qquad (3)$$

$$g(u_2, v_2, w_2) = 0 \qquad (4)$$

$$
\begin{aligned}
(x-u_1)^2+(y-v_1)^2+(z-w_1)^2 &= d^2 && (5)\\
(x-u_2)^2+(y-v_2)^2+(z-w_2)^2 &= d^2 && (6)\\
[x-u_1, y-v_1, z-w_1]\cdot[-f_{v_1}, f_{u_1}, 0] &= 0 && (7)\\
[x-u_1, y-v_1, z-w_1]\cdot[f_{w_1}, 0, -f_{u_1}] &= 0 && (8)\\
[x-u_1, y-v_1, z-w_1]\cdot[0, -f_{w_1}, f_{v_1}] &= 0 && (9)\\
[x-u_2, y-v_2, z-w_2]\cdot[-g_{v_2}, g_{u_2}, 0] &= 0 && (10)\\
[x-u_2, y-v_2, z-w_2]\cdot[g_{w_2}, 0, -g_{u_2}] &= 0 && (11)\\
[x-u_2, y-v_2, z-w_2]\cdot[0, -g_{w_2}, g_{v_2}] &= 0 && (12)
\end{aligned}
$$

Subscripting, as in f_{u_1}, denotes partial differentiation.

Equations (3–6) are quite clear and express Assertions (ii) and (iii). Equations (7–9) together express Assertion (iv), since the three vectors

$$
\begin{aligned}
&[-f_{v_1}, f_{u_1}, 0]\\
&[f_{w_1}, 0, -f_{u_1}]\\
&[0, -f_{w_1}, f_{v_1}]
\end{aligned}
$$

are tangent to f and span the tangent space as long as p_f is not a singular point on the surface. Similarly, Eqs. (10–12) together express Assertion (v).

Note that if f is given in parametric form, then Eq. (3) and Eqs. (7–9) have to be adjusted. This is routine and shows that the methodology is independent of the representation of the base surfaces. In principle, an implicit equation could be derived by eliminating the variables $\{u_1, v_1, ..., w_2, d\}$ from the system, but in almost all cases this computation is not tractable.

The entire system of equations defines a manifold in 10-dimensional space.[3] The projection of that manifold into the (x, y, z)-subspace is the equal-distance surface. A number of papers discuss other examples of surface definitions using the dimensionality paradigm, including

- offset surfaces, [46,45];
- constant-radius blends, [46];
- variable-radius blends, [46,47];
- ruled surfaces in parametric blending, [48];
- trimming surfaces in skeleton computations, [49].

We do not repeat these definitions here.

[3] Note that d is a variable

D. FAITHFUL DEFINITION SYSTEMS

The equation systems formulated by the dimensionality paradigm entail certain additional point sets that are unwanted because they do not reflect the geometric intent. The origin of these additional solutions is found in possible interdependence of the individual equations at certain points in space. For example, if p_f is a singular point, then Eqs. (7–9) vanish. In consequence, the system also defines a manifold that projects to the equal-distance surface of g and the singular point.

Extraneous solutions present in the system can be excluded by introducing certain additional equations, [50]. These new equations use one or more additional variables that no longer have a direct geometric meaning, but are used to express inequalities through equations. This "trick" originates with refutational approaches in automated geometry theorem proving, [51,52]. Note that all extraneous solutions can be so eliminated. The equations added reflect a generic method, but must account for the geometry of the original system.

A technical subtlety of this work is to define precisely what is meant by "extraneous" solution. Following [50], we define the extraneous solutions in the example of the Voronoi surface, Eqs. (3–12). Let $p = (x, y, z)$ be a point of the equal-distance surface, $p_f = (u_1, v_1, w_1)$ a point on f at minimum distance from p, and $p_g = (u_2, v_2, w_2)$ a point on g also at minimum distance from p. The points p_f and p_g are *footpoints* of p on f and g, respectively. Footpoints and the associated surface point(s) are said to *correspond*. Then a solution is *extraneous* if it corresponds to a footpoint that, in turn, corresponds to infinitely many solutions. It can be shown that all real extraneous solutions to Eqs. (3–12) arise as follows, [50]:

1. Footpoints p_f or p_g are singular. In this case, we obtain as extraneous solutions points at equal distance from the singular point and the other surface. In case both footpoints are singular but not coincident, there is an additional plane. If both footpoints are singular and coincident, then every point in $\mathbf{R}^3$ is an extraneous solution.

2. The footpoints coincide, are regular, and the surfaces intersect tangentially. In this case, the common surface normal is extraneous.

Note that we must assume for the proof that f and g are algebraic surfaces, because Bezout's Theorem is used to show that all other footpoints correspond to finitely many points of the equal-distance surface.

We explain how to modify the system so as to exclude all extraneous solutions enumerated. Consider the equation

$$\alpha x - 1 = 0$$

It is solved iff both α and x are not zero. Thus, if x is a quantity occurring in some system and α is a new variable, then adjoining this equation to the

system effectively expresses the constraint

$$x \neq 0$$

Now consider the equation

$$(\alpha x - 1)(\alpha y - 1) = 0$$

It expresses effectively

$$(x \neq 0) \vee (y \neq 0)$$

We use this idea to exclude all singular footpoints from the definition of the equal-distance surface by adjoining

$$\begin{aligned}(\alpha f_{u_1} - 1)(\alpha f_{v_1} - 1)(\alpha f_{w_1} - 1) &= 0\\ (\beta g_{u_2} - 1)(\beta g_{v_2} - 1)(\beta g_{w_2} - 1) &= 0\end{aligned}$$

where α and β are new variables. The two equations express that not all partial derivatives of f and of g vanish simultaneously at footpoints. Likewise, adjoining

$$(\gamma(u_1 - u_2) - 1)(\gamma(v_1 - v_2) - 1)(\gamma(w_1 - w_2) - 1)(\gamma D - 1) = 0$$

where

$$D = \begin{vmatrix} 1 & 1 & 1 \\ f_{u_1} & f_{v_1} & f_{w_1} \\ g_{u_2} & g_{v_2} & g_{w_2} \end{vmatrix}$$

expresses that the footpoints p_f and p_g are distinct, or else that the surface normals through them do not coincide. In consequence, the system of equations

$$\begin{aligned}
f(u_1, v_1, w_1) &= 0\\
g(u_2, v_2, w_2) &= 0\\
(x - u_1)^2 + (y - v_1)^2 + (z - w_1)^2 &= d^2\\
(x - u_2)^2 + (y - v_2)^2 + (z - w_2)^2 &= d^2\\
[x - u_1, y - v_1, z - w_1] \cdot [-f_{v_1}, f_{u_1}, 0] &= 0\\
[x - u_1, y - v_1, z - w_1] \cdot [f_{w_1}, 0, -f_{u_1}] &= 0\\
[x - u_1, y - v_1, z - w_1] \cdot [0, -f_{w_1}, f_{v_1}] &= 0\\
[x - u_2, y - v_2, z - w_2] \cdot [-g_{v_2}, g_{u_2}, 0] &= 0\\
[x - u_2, y - v_2, z - w_2] \cdot [g_{w_2}, 0, -g_{u_2}] &= 0\\
[x - u_2, y - v_2, z - w_2] \cdot [0, -g_{w_2}, g_{v_2}] &= 0\\
(\alpha f_{u_1} - 1)(\alpha f_{v_1} - 1)(\alpha f_{w_1} - 1) &= 0\\
(\beta g_{u_2} - 1)(\beta g_{v_2} - 1)(\beta g_{w_2} - 1) &= 0\\
(\gamma(u_1 - u_2) - 1)(\gamma(v_1 - v_2) - 1)(\gamma(w_1 - w_2) - 1)(\gamma D - 1) &= 0
\end{aligned} \qquad (13)$$

defines the equal-distance surface of f and g without extraneous solutions. The same approach has been used in [50] to exclude extraneous solutions from offsets, constant-radius and variable-radius surface definitions.

In this approach to faithful surface definition, the exclusion of all extraneous solutions also removes finitely many solutions that, by continuity, ought to be included in the final surface. For example, associated with two coincident regular footpoints is always the solution $p = p_f = p_g$, now excluded in case of coincident normals. If the projection of the system into the (x, y, z)-subspace is done using variable elimination, then these "missing points" are reintroduced by closure; [53]. If the system is interrogated by the methods to be explained next, then the missing points can be reintroduced by compensatory steps in the algorithms. However, in almost all cases the missing points will not be noticed because the numerical processing evaluates only a discrete sample of points, and the missing points are isolated.

E. INTERROGATION OF SURFACES DEFINED WITH THE DIMENSIONALITY PARADIGM

There is a considerable body of algorithmic infrastructure to deal with surfaces defined by systems of nonlinear equations. The following algorithms are now available:

(i) Given two surfaces and an initial point on both, evaluate their intersection; see [54,2,46]. The algorithm is robust and can evaluate very high-degree surface intersections without significant precision problems.

(ii) Given a surface and an initial point, evaluate locally the curvatures, [48], and give a local parametric or local explicit surface approximant of arbitrary contact order, [55,45].

(iii) Given a surface and an initial point, globally approximate the surface; [55].

These algorithms are not confined to algebraic equation systems, and can be used as long as the nonlinear equations of the system are continuously differentiable. They are very efficient.

Less efficient are the known techniques for finding initial points. When nothing is known about the system, then generic techniques such as [56,57,58, 59,60] can be applied. Usually, however, the geometric intent of the system is known and leads to good initial estimates for starting points that can be refined using Newton iteration. Moreover, some of the search techniques from CAGD are in principle applicable. That is, by raising the dimension of the ambient space by 1, all equations can be rewritten in Bernstein-Bèzier form after which domain reduction is applied using the convex hull property. To implement this, one has to exercise care not to obtain an exponential growth in the number of control points.

1. LOCAL PARAMETRIC APPROXIMATIONS

We assume given a surface definition using the dimensionality paradigm, in the form

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0 \\ f_2(x_1, x_2, ..., x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, ..., x_n) &= 0 \end{aligned} \tag{14}$$

and an initial point $p = (u_1, u_2, ..., u_n)$ solving the system. At p, we assume that every hypersurface f_i is regular and twice continuously differentiable, and that the Jacobian matrix

$$\left(\frac{\partial f_i}{\partial x_j}\right)_{i,j}$$

has rank $n-2$. Then the manifold $\mathcal{M}$ defined by Eq. (14) is of dimension 2 in the neighborhood of p. Its natural projection into the (x_1, x_2, x_3)-subspace will be denoted by $\pi(\mathcal{M})$. We describe how to compute a local approximant to the manifold at p, using the approach of [55,45]. So, we will derive coordinate functions

$$\begin{aligned} x_1 &= h_1(s,t) \\ x_2 &= h_2(s,t) \\ &\vdots \\ x_n &= h_n(s,t) \end{aligned} \tag{15}$$

such that

$$p = (h_1(0,0), ..., h_n(0,0))$$

and

$$f_i(h_1(s,t),\, h_2(s,t),\, ...,\, h_n(s,t)) \equiv 0$$

for $i = 1, ..., m$. Note that the natural projection $\pi(\mathcal{M})$ of the manifold $\mathcal{M}$ into the (x_1, x_2, x_3)-subspace is then locally approximated by

$$\begin{aligned} x_1 &= h_1(s,t) \\ x_2 &= h_2(s,t) \\ x_3 &= h_3(s,t) \end{aligned}$$

in the vicinity of the projection $\pi(p)$ of p.

Now the Taylor expansion of the hypersurface f_i at the point $p = (u_1, ..., u_n)$ is given by

$$
\begin{aligned}
f_i(x_1,...,x_n) &= f_i(u_1+\delta_1, u_2+\delta_2, ..., u_n+\delta_n) \\
&= f_i(u_1,...,u_n) \\
&\quad +f_i^{(1)}\delta_1 + \cdots + f_i^{(n)}\delta_n \\
&\quad +[f_i^{(1,1)}\delta_1^2 + \cdots + f_i^{(n,n)}\delta_n^2]/2 \\
&\quad +f_i^{(1,2)}\delta_1\delta_2 + \cdots + f_i^{(1,n)}\delta_1\delta_n \\
&\quad +f_i^{(2,3)}\delta_2\delta_3 + \cdots + f_i^{(n-1,n)}\delta_{n-1}\delta_n \\
&\quad + \text{higher order terms}
\end{aligned} \tag{16}
$$

Here, $f_i^{(j,k)}$ denotes the partial derivative of f_i by x_j and x_k.

Next, consider the Taylor expansion of the local coordinate functions, having chosen s and t such that $p = (h_1(0,0),\ h_2(0,0),\ ...,\ h_n(0,0))$:

$$
\begin{aligned}
h_k(s,t) &= h_k(0,0) \\
&\quad +h_k^{(s)}s + h_k^{(t)}t \\
&\quad +[h_k^{(s,s)}s^2 + 2h_k^{(s,t)}st + h_k^{(t,t)}t^2]/2 \\
&\quad + \text{higher order terms}
\end{aligned}
$$

where $h_k^{(s)}$ denotes the partial derivative of h_k by s, and so on. By assumption, there is a neighborhood of p in which

$$f_i(h_1(s,t),\ h_2(s,t),\ ...,\ h_n(s,t)) \equiv 0$$

We set

$$
\begin{aligned}
\delta_1 &= h_1^{(s)}s + h_1^{(t)}t + [h_1^{(s,s)}s^2 + 2h_1^{(s,t)}st + h_1^{(t,t)}t^2]/2 + \cdots \\
\delta_2 &= h_2^{(s)}s + h_2^{(t)}t + [h_2^{(s,s)}s^2 + 2h_2^{(s,t)}st + h_2^{(t,t)}t^2]/2 + \cdots \\
&\ \ \vdots \\
\delta_n &= h_n^{(s)}s + h_n^{(t)}t + [h_n^{(s,s)}s^2 + 2h_n^{(s,t)}st + h_n^{(t,t)}t^2]/2 + \cdots
\end{aligned}
$$

and note that

$$
\begin{aligned}
\delta_k^2 &= (h_k^{(s)})^2s^2 + 2h_k^{(s)}h_k^{(t)}st + (h_k^{(t)})^2t^2 + \cdots \\
\delta_k\delta_j &= h_k^{(s)}h_j^{(s)}s^2 + (h_k^{(s)}h_j^{(t)} + h_k^{(t)}h_j^{(s)})st + h_k^{(t)}h_j^{(t)}t^2 + \cdots
\end{aligned}
$$

Now we substitute these quantities into Eq. (16), obtaining a power series in s and t. The coefficient of each term s^at^b in the power series must be zero. In consequence, the following systems of equations are implied:

$$f_i^{(1)}h_1^{(s)} + f_i^{(2)}h_2^{(s)} + \cdots + f_i^{(n)}h_n^{(s)} = 0 \tag{17}$$

$$f_i^{(1)}h_1^{(t)} + f_i^{(2)}h_2^{(t)} + \cdots + f_i^{(n)}h_n^{(t)} = 0 \tag{18}$$

$$f_i^{(1)} h_1^{(s,s)} + f_i^{(2)} h_2^{(s,s)} + \cdots + f_i^{(n)} h_n^{(s,s)} = -c_i \tag{19}$$

$$f_i^{(1)} h_1^{(s,t)} + f_i^{(2)} h_2^{(s,t)} + \cdots + f_i^{(n)} h_n^{(s,t)} = -d_i \tag{20}$$

$$f_i^{(1)} h_1^{(t,t)} + f_i^{(2)} h_2^{(t,t)} + \cdots + f_i^{(n)} h_n^{(t,t)} = -e_i \tag{21}$$

where $i = 1, ..., m$. The right hand sides are, respectively,

$$\begin{aligned} c_i = \; & f_i^{(1,1)} (h_1^{(s)})^2 + \cdots + f_i^{(n,n)} (h_n^{(s)})^2 \\ & + 2[f_i^{(1,2)} h_1^{(s)} h_2^{(s)} + \cdots + f_i^{(1,n)} h_1^{(s)} h_n^{(s)} \\ & \quad \cdots + f_i^{(n-1,n)} h_{n-1}^{(s)} h_n^{(s)}] \end{aligned}$$

$$\begin{aligned} d_i = \; & f_i^{(1,1)} h_1^{(s)} h_1^{(t)} + \cdots + f_i^{(n,n)} h_n^{(s)} h_n^{(t)} \\ & + f_i^{(1,2)} h_1^{(s)} h_2^{(t)} + \cdots + f_i^{(1,n)} h_1^{(s)} h_n^{(t)} \\ & \quad \cdots + f_i^{(n-1,n)} h_{n-1}^{(s)} h_n^{(t)} \\ & + f_i^{(1,2)} h_1^{(t)} h_2^{(s)} + \cdots + f_i^{(1,n)} h_1^{(t)} h_n^{(s)} \\ & \quad \cdots + f_i^{(n-1,n)} h_{n-1}^{(t)} h_n^{(s)} \end{aligned}$$

$$\begin{aligned} e_i = \; & f_i^{(1,1)} (h_1^{(t)})^2 + \cdots + f_i^{(n,n)} (h_n^{(t)})^2 \\ & + 2[f_i^{(1,2)} h_1^{(t)} h_2^{(t)} + \cdots + f_i^{(1,n)} h_1^{(t)} h_n^{(t)} \\ & \quad \cdots + f_i^{(n-1,n)} h_{n-1}^{(t)} h_n^{(t)}] \end{aligned}$$

The partial derivatives of the coordinate functions h_i are computed from these systems of linear equations, and define an approximate local parameterization of the manifold given by Eq. (14).

The linear systems of Eqs. (17–21) have rank deficiency 2. Their solutions, therefore, take the form

$$\alpha_1 \nabla f_1 + \cdots + \alpha_{n-2} \nabla f_{n-2} + \beta \mathbf{t}_1 + \gamma \mathbf{t}_2$$

where $\mathbf{t}_1$ and $\mathbf{t}_2$ are two linearly independent tangent directions to the surface at the point p. These tangent directions are determined by the method chosen to solve the linear systems.

To choose suitable values for the β and γ coefficients, we proceed as follows. For Eq. (17) and Eq. (18), we let

$$\alpha_1 = \alpha_2 = \cdots = \alpha_{n-2} = 0$$

For Eq. (17), we choose $(\beta, \gamma) = (1, 0)$; and for Eq. (18), we choose $(\beta, \gamma) = (0, 1)$. Then the isoparametric curves

$$(h_1(s,0), ..., h_n(s,0)) \qquad \text{and} \qquad (h_1(0,t), ..., h_n(0,t))$$

intersect transversally at p. Moreover, if $\mathbf{t}_1$ and $\mathbf{t}_2$ are orthogonal, then so are the isoparametric curves through p. For Eqs. (19–21) we choose $\beta = \gamma = 0$.

We have shown how to construct local linear and quadratic approximants to the manifold $\mathcal{M}$ at p, which also give local approximants to the surface $\pi(\mathcal{M})$ at $\pi(p)$. Assuming that the f_i are sufficiently differentiable, higher-order local approximants are constructed in the same way.

2. CURVATURE DETERMINATION

As shown in [48], it is possible to determine the curvature of the surface $\pi(\mathcal{M})$ at p. We recall the following

Lemma 3 Let $\mathbf{n}_i$ be the normal vector to the hypersurface f_i at the point p, for $1 \leq i \leq m$. Let α_i be such that the last $n-3$ components of

$$\mathbf{n}_0 = \sum_{i=1}^{m} \alpha_i \mathbf{n}_i = (a, b, c, 0, ..., 0)$$

are zero. Then the natural projection of $\mathbf{n}_0$ is normal to $\pi(\mathcal{M})$ at the projection of p.

The normal curvature of $\pi(\mathcal{M})$ at p will be determined in a direction that is the natural projection of a tangent $\mathbf{v}$ to $\mathcal{M}$ at p. The actual computation requires forming a linear combination of the Hessians H_i of the f_i. It also requires that the vectors $\mathbf{v}$ and $\mathbf{n}_0$ project to vectors of unit length, that is, the sum of squares of the first three components of each vector must be 1. We state how to compute the normal curvature.

Theorem 4 Let $\mathbf{n}_i$ be the normal vector to the hypersurface f_i at the point p, for $1 \leq i \leq m$. Let α_i be such that the last $n-3$ components of $\mathbf{n}_0 = \sum_{i=1}^{m} \alpha_i \mathbf{n}_i = (a, b, c, 0, ..., 0)$ are zero, and such that $a^2 + b^2 + c^2 = 1$. Let $\mathbf{v} = (v_1, v_2, ..., v_n)$ be a tangent vector to $\mathcal{M}$ at p where $v_1^2 + v_2^2 + v_3^2 = 1$. Let H_i be the Hessian of f_i. Then the normal curvature of $\pi(\mathcal{M})$ at p in the (projected) direction $\pi(\mathbf{v})$ is given by

$$\kappa = -\mathbf{v}^T (\sum_{i=1}^{m} \alpha_i H_i) \mathbf{v}$$

It is well-known that the principle curvatures and their directions can be recovered from the normal curvatures in three different directions; e.g., [61]. Thus, the principal curvatures, mean curvature, and Gauss curvature of $\pi(\mathcal{M})$ can all be determined with help of the theorem.

3. SURFACE INTERSECTION

The derivation of local approximants can be used to construct local approximants to surface intersections as well. As discussed in [54,2], at the point p we

construct an approximant of the form

$$\begin{aligned} x_1 &= h_1(s) \\ x_2 &= h_2(s) \\ &\vdots \\ x_n &= h_n(s) \end{aligned} \tag{22}$$

such that

$$p = (h_1(0), ..., h_n(0))$$

The derivation is exactly as in the surface approximations, except that the right hand sides are simpler. Once the approximant has been obtained, to some degree, it is used to derive a new point estimate

$$(h_1(s_1), ..., h_n(s_1))$$

which is refined with Newton iteration and becomes the new point at which to construct a local approximant.

This approach to evaluating surface intersections by marching can be very effective. In [46] we show the evaluation of intersections with a variable-radius blend that have an estimated degree in excess of 500. The system defining this curve consists of equations of low to moderate algebraic degree, which is an important aspect of the dimensionality paradigm.

4. GLOBAL APPROXIMATIONS

For curves such as surface intersections it is easy to derive a marching scheme like the one summarized before. In order to obtain a similar scheme for evaluating surfaces we need a device that orients the exploration in space and prevents inadvertently exploring the same neighborhood several times. In [55] this problem has been addressed in the context of the dimensionality paradigm. The technique competes well with other approaches such as Allgower's simplicial continuation method, [57], or the moving-frame method of Rheinboldt, [62]. All methods are based on the following idea. Given a manifold $\mathcal{M}$ by the system of Eq. (14) and on it a point $p = (u_1, ..., u_n)$, construct a piecewise approximation of $\mathcal{M}$, beginning at p and extending in all directions.

Rheinboldt's moving-frame method proceeds by triangulating the tangent space at p, and transferring the triangulation to $\mathcal{M}$ with Newton iteration. Each vertex of the triangulation, after projection to a point q on $\mathcal{M}$, becomes the center of a new triangulation, of the tangent space at q. The algorithm locally resolves any overlap, but cannot do so globally.

Allgower's method is based on a triangulation of ambient space and only evaluates the system of equations. At each vertex of a simplex, Eq. (14) is

evaluated and from the m-vector of function values a linear approximant is constructed. Moreover, it is then deduced through which faces the linear approximant passes, and based on this information adjacent simplices are considered in the same way. Note that adjacent simplices can be found efficiently by a triangulation schema based on an elegant enumeration. In Allgower's method the ambient n-dimensional space is triangulated. In consequence, the number of simplices in an elementary volume grows exponentially with the number of variables. However, the method does not need to assume differentiability of the f_i.

In the case of surface definitions with the dimensionality paradigm, the surface that is ultimately of interest is the projection $\pi(\mathcal{M})$. Since this surface is in a three-dimensional space, it is advantageous to construct an approximant only for the projection, and Chuang's algorithm [55] does this as follows, using a grid in 3-space to detect whether a volume of space has already been explored.

1. At p, construct a local approximant to $\mathcal{M}$ as described before.
2. Determine how the projected approximant, $(h_1(s,t), h_2(s,t), h_3(s,t))$, intersects the faces of the cube, as a function of s and t.
3. From the intersection curves, determine the coordinates (s_1, t_1) of a point on the approximant that lies in an adjacent cube.
4. Refine the estimated point with Newton iteration.

There is a tradeoff between the degree of the approximant, the mesh size of the grid, and the difficulty of determining face intersections and adjacent points in Steps 2 and 3. With increasing degree of the approximant a coarser mesh can be tolerated, so that fewer approximant calculations are needed. However, determining the intersection with the faces of the current cube becomes more difficult. The advantage of this method is the fact that the dimension of the meshed space does not depend on the number of variables used to define $\mathcal{M}$. Yet, by determining the (s,t) curves, each estimate (s_1, t_1) can be pulled back into the n-space in which $\mathcal{M}$ is given.

We illustrate the method for linear approximants. Assume that we are at a point $p = (u_1, u_2, ..., u_n)$ on $\mathcal{M}$ that projects to $\pi(p)$ in a cube, and we have constructed the linear approximant

$$\begin{aligned} L: \quad x_1 &= u_1 + v_1 s + w_1 t \\ x_2 &= u_2 + v_2 s + w_2 t \\ &\vdots \\ x_n &= u_n + v_n s + w_n t \end{aligned}$$

L intersects the faces of the cube containing $\pi(p)$, say as shown in Figure 22. Each intersection is easily determined: If the face plane is $x_1 = a$, then the

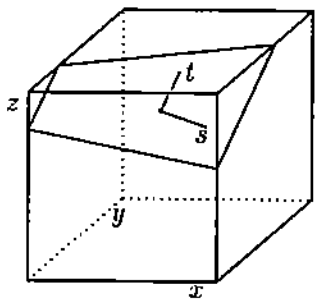


Figure 22: Linear Approximant in a Space Element and Local Frame

line in (s,t)-space corresponding to the intersection with L is simply

$$v_1 s + w_1 t = a - u_1$$

Intersections with the planes $y = b$ and $z = c$ are found analogously. Thus, the intersections define a polygon in (s,t)-space that corresponds to the area of L contained in the cube, as shown in Figure 23. Possibly with help of additional lines corresponding to the intersection of L with the faces of adjacent cubes, we can find good estimates (s_1, t_1) for new points in neighboring cells, as illustrated in Figure 24.

Note that the linear approximants found at each point do not have C^0 continuity. Assuming that the mesh is sufficiently fine, a simple solution is to triangulate the points found on $\pi(\mathcal{M})$. Since we have good adjacency information from the construction, no ambiguities have to be resolved. Other possibilities include determining the principal curvatures and directions at each point on $\pi(\mathcal{M})$. Thereupon, interpolation algorithms such as [63] could be used to provide a surface approximant that has C^1 continuity.

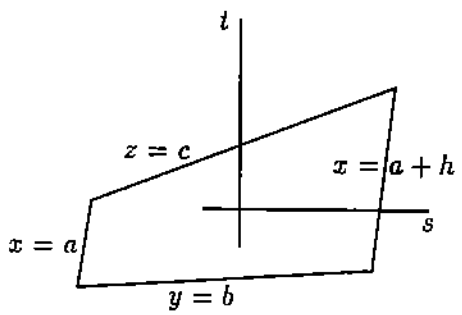


Figure 23: Corresponding Face Intersection Lines in (s,t)-Space

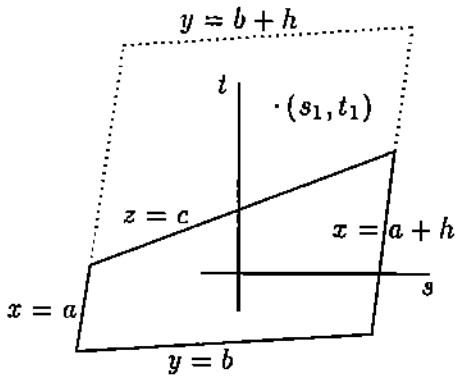


Figure 24: Finding a Point (s_1, t_1) in Adjacent Cubes

V. THE SKELETON

The *interior skeleton* of a three-dimensional solid is the locus of the centers of all inscribed maximal spheres. A sphere is *maximal* if there is no other inscribed sphere that contains it completely. Other names for the skeleton include *medial-axis transform* and *symmetric-axis transform.*

Computer vision has proposed the skeleton as a shape representation; e.g., [64,65]. Applications of the skeleton in pattern recognition are discussed; e.g., in [64,66]. In the context of geometric modeling applications, it has been argued that the skeleton in 2D is useful for generating finite-element meshes [67,68, 69], because it allows identifying areas in which the 2D domain is constricted or extended, and so provides the basis for quantifying shape parameters of relevance to the mesh density.

If we know for each skeleton point its distance from the boundary, then the skeleton is an informationally-complete representation in the sense of Requicha [70]. We discuss this representation now, and then consider some of the applications that might be facilitated by it. For the sake of a clear presentation, we begin with a discussion of the skeleton of two-dimensional objects. We then point out how to generalize the techniques to three-dimensional solids.

There is an interesting connection between skeletons and certain concepts from classical geometry and from the theory of differential equations, with consequences on how to compute the skeleton, [71]. We will discuss these connections as well.

A. THE SKELETON REPRESENTATION

Consider a bounded solid T, in $\mathbf{R}^2$ or $\mathbf{R}^3$, with compact boundary ∂T. For every point p, we define its *distance* from the boundary of T as the minimum

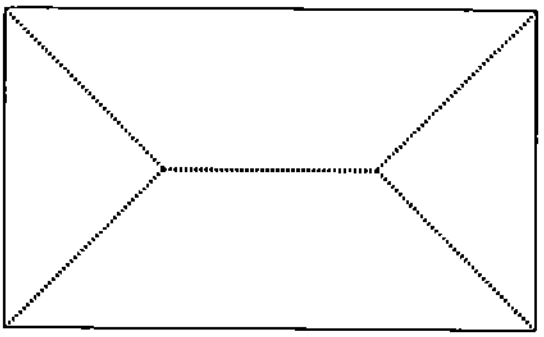


Figure 25: Interior Skeletons of a Simple Domain

Euclidean distance $d(p,q)$ where q is on ∂T. For every p, there is always at least one point q on the boundary of T such that $d(p,q)$ is minimum, and such a point q will be called a *foot point* of p on T. Then the *interior skeleton* of T consists of all points p that are interior points of T and have more than one footpoint on ∂T, as well as the limits of point sequences in this set. That is, the skeleton is the relative closure of the set of all interior points that do not have unique footpoints. An example in two dimensions is shown in Figure 25. Similarly, the *exterior skeleton* of T is the closure of all points p that are exterior to T and have more than one footpoint.

We associate with each skeleton point its distance from ∂T, as an additional coordinate. For two-dimensional solids, therefore, the skeleton point (u,v) at distance r from ∂T will be associated with (u,v,r) in 3-space. The skeleton thus becomes a three-dimensional object, as shown in Figure 26. We recover the boundary ∂T from this skeleton representation as follows. Associate with the point (u,v,r) of the skeleton the circle $(x-u)^2+(y-v)^2-r^2=0$. Then ∂T is the envelope of all such circles. We will see later that the envelope points of the circle can be determined without constructing the envelope.

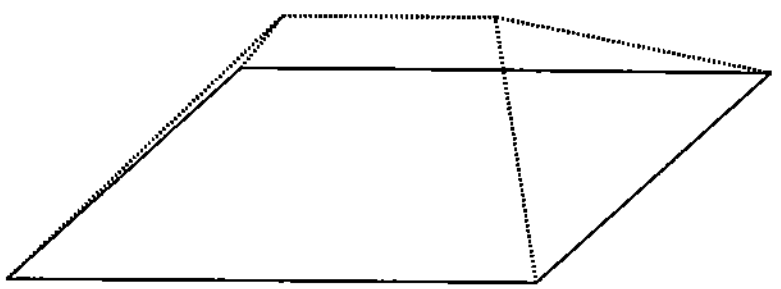


Figure 26: The Skeleton as 3D Object

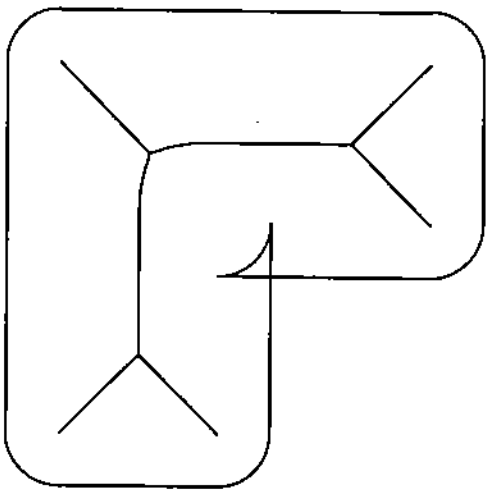


Figure 27: Self-Intersections of Positive Skeleton Translates

Likewise, the skeleton point (u, v, w) at distance r, of the three-dimensional solid T is represented by the point (u, v, w, r) in 4-space, thus considering the skeleton a four-dimensional structure. Then ∂T is the envelope of the spheres $(x-u)^2 + (y-v)^2 + (z-w)^2 - r^2 = 0$.

Since the r-coordinate gives the distance from the boundary, we expect that a translation of the skeleton in the r-direction represents an interior or exterior offset of T. More precisely, with d a signed translation distance, let the *d-translate* of the skeleton S consist of the points

$$\{(u, v, w, r+d) \,|\, (u, v, w, r) \in S,\, r + d \geq 0\}$$

Then a negative d-translate represents the interior d-offset of T, whereas, in the case of positive d-translates, self-intersections are possible when T is not convex. See also Figure 27.

The possibility that the boundaries of positive translates may contain self-intersections suggests that the interior skeleton does not contain sufficient information to generate offsets in a simple way. In fact, exterior offsets will require the exterior skeleton as well. Briefly, a positive translate of the interior skeleton is combined with a negative translate of the exterior skeleton, by the same distance. Then the exterior skeleton points that are clipped by the rule $r + d \geq 0$ identify self-intersections.

B. CYCLOGRAPHIC MAPS AND IMAGES

The concept of cyclographic maps is due to Müller [72], and has uses in descriptive geometry. Cyclographic maps provide a different conceptualization for the skeleton, and we explore this now in detail.

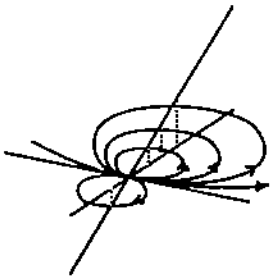


Figure 28: Cyclographic Map of All Cycles Tangent to C in a Point

We consider oriented circles in the (x,y)-plane, calling them *cycles* so as to stress that they are oriented. The plane is considered in (x,y,z)-space. A counter-clockwise orientation is *positive*, as seen from points with positive z coordinates, and a *negative* orientation is clockwise. If a cycle has the center (u,v) and radius r, we associate with it the point (u,v,r) in 3-space, where r is signed according to the cycle orientation. This association defines a bijection

$$\Phi : \mathbf{R}^3 \mapsto C$$

between points in 3-space and cycles in the (x,y)-plane. Note that the points in the plane correspond to cycles of zero radius.

Let C be an oriented curve in the plane. We investigate which cycles are tangent to C, requiring that the cycles and the curve are consistently oriented, as shown in Figure 28. Let p be any point on C. Then all cycles tangent to C at p have their centers on the normal to C and are at distance equal to the cycle radius. Hence, the points in 3-space to which those cycles correspond are on a line that is inclined to the (x,y)-plane by 45° and orthographically projects onto the normal of C at p. The set of all oriented circles tangent to the curve C therefore corresponds to a ruled surface in 3-space all of whose generators intersect C, project onto the normals of C, and have slope 1 against the (x,y)-plane. This surface is the *cyclographic map* of C.

In general, since all generators of the cyclographic map have equal slope, the ruled surface is developable [72]. In particular, if C is a line, then the cyclographic map is an inclined plane, and if C is a circle, then the cyclographic map is a right circular cone with a right angle at the vertex.

Now consider the reverse process, of investigating the cycle geometry that corresponds to a given curve in space. Intuitively, the family of cycles corresponding to the points in space will have an envelope, although the envelope could be imaginary. We call this envelope the *cyclographic image* of the space curve.

In the simplest case, the space curve is a line ℓ, and each point on this line maps to a cycle. If ℓ is parallel to the (x,y)-plane, then the cycles have equal radius and their common envelope is a pair of parallel lines in the (x,y)-plane. If ℓ is inclined, with a slope less than 45°, then the cycles are enveloped by a

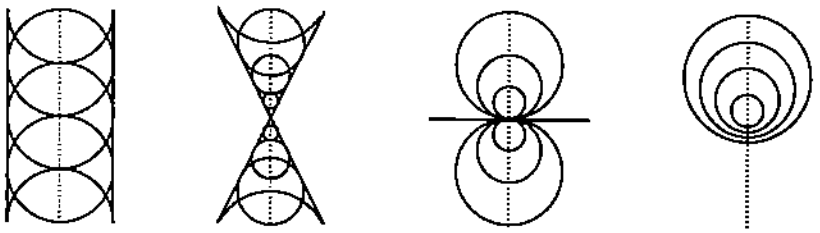


Figure 29: Cyclographic Image of a Line in Space

pair of intersecting lines. The intersection is the point at which ℓ intersects the (x, y)-plane. If the line ℓ is at an angle of 45°, then the envelope is a single line perpendicular to the projection of ℓ onto the (x, y)-plane. For greater inclination angles, the cycles have no real envelope. See also Figure 29.

If we choose a space curve C other than a line, then to each point p of C the corresponding cycle contributes two points to the cyclographic image which are real and distinct if the tangent at p has slope less than 45°, coincident if the slope is 45°, and conjugate complex otherwise. Moreover, the orthogonal projection of the space curve C onto the (x, y)-plane bisects the envelope curve, and so C is the skeleton of the cyclographic image. Furthermore, it can be proved that the tangent to C at p, and the tangents to the cyclographic image of C, at the points contributed to the envelope by the cycle corresponding to p, are concurrent, as shown in Figure 30. Note that this fact implies a direct construction of the footpoints on the boundary from each skeleton point.

There are a number of interesting relationships between the space curve C and its cyclographic image that are discussed in [72]. For example, the tangents to C form the cyclographic map of the cyclographic image. Furthermore, the edge of regression of the cyclographic map projects onto the evolute of the cyclographic image.

Consider the cyclographic map of the boundary ∂T of a 2D solid T. We will trim this map as follows: For every point $q = (u, v)$ in the (x, y)-plane, consider the normal of the plane through q. This line intersects the cyclographic map in a number of points. If q is interior of T, then we choose among the intersection

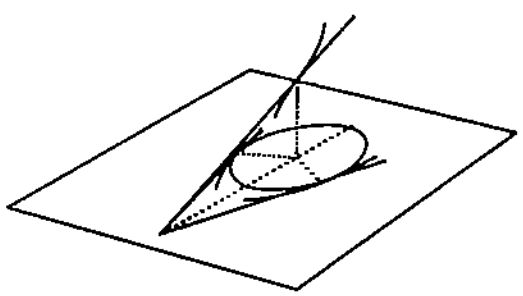


Figure 30: Cyclographic Image of a Space Curve

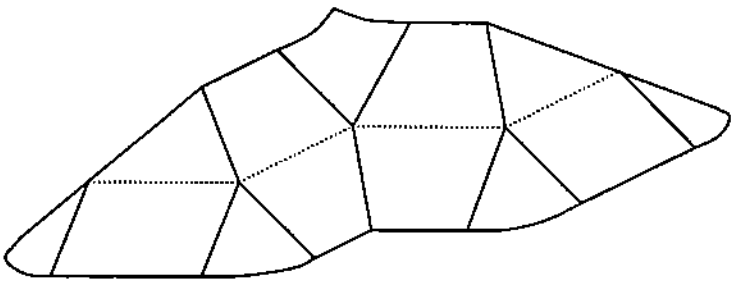


Figure 31: The Distance Function $\mathcal{S}$ Defined by the Cyclographic Map

points the one that lies above the (x, y)-plane and has the smallest z coordinate. If q is exterior to T, then we choose the intersection point below the (x, y)-plane with the largest z-coordinate. In either case, we have associated a point that lies on that line of the cyclographic map that goes through the nearest foot point of q on ∂T. With this convention, we have defined a single-valued function $\mathcal{S}$ whose domain is the (x, y)-plane and whose range is a subset of the cyclographic map of ∂T. Note that $\mathcal{S}$ is zero only on the boundary of T. We will see later that this function is characterized by the Hamilton-Jacobi equation. Note that $\mathcal{S}$ defines the minimum Euclidean distance of every point of the (x, y)-plane to the boundary of ∂T. Clearly its singular curves are the skeleton of T. See Figure 31 for an example.

Since the $\mathcal{S}$ function is the Euclidean distance function, we observe that it represents all interior and exterior offsets as follows: In order to obtain the boundary of the interior d-offset of T, intersect $\mathcal{S}$ with the plane $z = d$. The boundary of the exterior d-offset is given similarly by the intersection of $\mathcal{S}$ with the plane $z = -d$. See also Figure 31.

This construction of the offsets interprets why the interior offset can be obtained as the positive d-translate of the interior skeleton. The interior skeleton characterizes $\mathcal{S}$ in the interior of T, but does not reflect the exterior structure of $\mathcal{S}$ completely. Thus, the exterior skeleton must be translated to obtain exterior offsets without self-intersection.

It has been pointed out in [73] that a solid can be blended by a succession of interior and exterior offsets, by the rounding radius. The exterior offset of the interior offset obtains all rounds, and the interior offset of the exterior offset obtains all fillets. It is clear, however, that the translation of the (x, y)-plane first by d and then by $-d$ recovers the original boundary ∂T. It follows that the trimmed cyclographic map of an offset is not equal to the translation of the cyclographic map of ∂T. In the construction of interior offsets as negative d-translates of the skeleton, this fact finds its counterpart in the clipping effected by the condition $r + d \geq 0$.

In analogy to the two-dimensional case, we can define cyclographic maps that establish a correspondence between oriented spheres in 3-space, and the points of a four-dimensional space. The map of a solid boundary ∂T is now a curved space that is ruled, and the ruling consists of lines, through every point of ∂T, that are inclined against (x, y, z)-space by 45° in the containing 4-space. Furthermore, the lines project onto the normals in (x, y, z)-space of the boundary of T. From differential geometry we know that the normals in 3-space envelope the focal surfaces of ∂T, [74], and the various constructions and properties outlined above for the two-dimensional case generalize virtually unchanged to the three-dimensional case [75]. In particular, the skeleton of three-dimensional domains is the singular set of the trimmed cyclographic map.

C. THE HAMILTON-JACOBI EQUATION

As we discussed before, the function S is a subset of the cyclographic map, which, in turn, is ruled with each generator inclined 45° against the (x, y)-plane, or against the (x, y, z)-space in the case of 3-dimensional solids T. Denoting the distance variable with t for the moment, we can express the geometry of the ruling by the equation

$$S_t^2 = S_x^2 + S_y^2$$

for two-dimensional solids, and by

$$S_t^2 = S_x^2 + S_y^2 + S_z^2$$

for three-dimensional solids. Since we can locally express t as an explicit function of the other variables, the equations become

$$S_x^2 + S_y^2 = 1$$

and

$$S_x^2 + S_y^2 + S_z^2 = 1$$

respectively, but this is just the well-known Hamilton-Jacobi equation. The function S, therefore, satisfies this differential equation with the boundary condition

$$S = 0 \quad \text{on} \quad \partial T$$

With proper sign conventions, it is clear that the Hamilton-Jacobi equation defines the cyclographic map S, and, because of the distance interpretation of t, S defines the Euclidean minimum distance from ∂T of any point in (x, y)- and (x, y, z)-space. This fact has been noted before; [76].

We have identified the skeleton as the locus of points at which the cyclographic map is derivative discontinuous. In terms of the Hamilton-Jacobi equation, this set of points is called a *shock wave*; e.g., [77]. Thus, if we have a PDE solver for integrating the Hamilton-Jacobi equation that locates the

shocks in the solution, it follows that such a solver also finds the skeleton of T. Integration inward from ∂T evaluates the interior skeleton, whereas integration outward from the boundary evaluates the exterior skeleton.

At the time of writing, we have only limited experience with this approach towards computing the skeleton. In particular, we have implemented the scheme described in [78]. Early experiments seem to indicate that the solver is more general than necessary. More specialized solvers should be faster and more accurate. We are experimenting with alternatives that integrate the equation as an initial-value problem, augmented by a correcting step that refines the solution based on the dimensionality paradigm.

D. GEOMETRIC APPROACHES TO COMPUTING THE SKELETON

Early approaches to compute the skeleton in two dimensions have evaluated an approximate distance from the boundary, by one or two passes over a discretization of T. Examples include [66,79].

Computational geometers have investigated the skeleton of polygonal domains. The algorithm by Preparata [80] evaluates first the branching points of the skeleton. Considering certain triples of edges, the associated branch point is the intersection of the bisectors of the edge pairs. Careful sequencing of triples yields an $O(n^2)$ algorithm for convex polygons. The sequencing can be conceptualized as constructing the interior offsets that contain a branching point. Then a branch point signals that certain edges of the boundary no longer contribute to the offset. An algorithm by Lee [81] constructs the skeleton using a divide-and-conquer approach, and achieves $O(n \log(n))$. Preparata's algorithm was extended by Patrikalakis and Gürsoy to domains bounded by line segments and circular arcs, [68]. As in the polygonal algorithm, branch points are determined but are now the intersection of conics.

Using some conceptual elements of Preparata's and the Patrikalakis-Gürsoy approach, [82] gives an algorithm for constructing the skeleton of CSG solids in 3-space. The algorithm computes nearest approach points between pairs of elements of the boundary, but not between triples, for that would entail algebraic computations that are of unattractive complexity. The closest approach points found are sorted by their distance from the boundary. In this phase a proximity computation also rejects those closest approach points that are nearer to a third boundary element.

Beginning at the closest approach points, the skeleton is evaluated, by increasing distance from the boundary. At each moment throughout the computation, the skeleton is known up to a current distance. Closest approach points at a greater distance are not processed until their respective distance has been reached. Each face, edge, and vertex of the skeleton is formulated as a suitable set of nonlinear equations, using the dimensionality paradigm, and is

evaluated with help of the approximation algorithm described in Section IV.G. In particular, the grid used to orient the exploration of faces and edges also signals that different skeleton faces are about to intersect.

Intersection of two faces of the skeleton are easily formulated and evaluated. Each face involves two boundary elements from which its points are equidistant. Their intersection is thus obtained by seeking all points equidistant from all boundary elements involved in the definition of the intersecting faces, and only requires combining the equation systems that describe the faces. A proximity computation determines how to trim faces in the vicinity of an edge or a vertex.

The algorithm does not attempt to exploit special geometric configurations of boundary elements. As discussed in [49], many such configurations exist and result in skeleton faces that have a special geometry that could be utilized in the construction of the face.

Bowyer et al. [83] describes an approximate skeleton algorithm for three-dimensional solids constructed from algebraic half spaces of arbitrary degree. The algorithm begins by superimposing a grid whose elements are classified as one of inside, outside, or boundary. Boundary elements intersect the solid boundary, inside elements are in the solid interior, and outside elements are exterior to the solid. Next, for all nonboundary elements, a distance computation determines the approximate minimum distance from the boundary. Those elements that are closest to more than one boundary element constitute the approximate skeleton. A number of heuristics are employed that narrow down the search for closest boundary elements. That is, for each boundary element, the cone of normals to the contained boundary area is determined, since interior elements that do not fall into this cone cannot be nearest to this boundary element. The algorithm has been implemented and incorporated into the Bath solids modeler by David Lavender. Note that the approach generalizes to solids of any dimension.

Price et al. [84] gives an algorithm for determining the branching points of the skeleton of arbitrary planar domains. First, it is observed that a Delaunay triangulation can be used to approximate the skeleton as follows: Select a large number of points from the domain boundary, and construct their Delaunay triangulation.[4] If the points are sufficiently dense, the triangulation will respect the domain boundary. Moreover, since in a Delaunay triangulation the circumscribed circle of every triangle contains no other triangle vertices in the interior, the skeleton is approximated by the centers of the circumscribing circles.

Price et al. consider how to select a small number of boundary points such that the resulting Delaunay triangulation respects the domain boundary and such that the centers of the circumscribed circles contain all branch points of the skeleton. Several heuristics are described that make the compatibility

[4] See, e.g., [85] for an algorithm to construct Delaunay triangulations.

test more efficient. In particular, by classifying the types of triangles, it is shown that many triangles need not be tested if certain critical ones respect the boundary. Exact branch points are found later using Newton iteration. The algorithm has been implemented. Note that the approach is relatively insensitive to the types of curves bounding the domain. The ideas and heuristics should generalize to the three-dimensional case. The method is used as basis of a finite-element mesh generation algorithm [67].

E. FINITE-ELEMENT MESH GENERATION

In contrast to grid generation methods like the one we discussed earlier in Section III.D, finite-element mesh generation discretizes a domain using an irregular mesh of simple shapes, often triangles or tetrahedra. Automating finite-element mesh generation is of interest because the meshing requires much effort and considerable manual labor at present. The suitability of the mesh depends on both the geometry of the domain as well as on the nature of the physical problem and its solution. It is therefore important that the mesh not only satisfy geometric criteria, but also that it can be refined iteratively in response to the analysis results.

Systematic finite-element mesh generation has been based on three fundamental approaches:

1. Impose a spatial subdivision, say using octrees. Subdivide the interior cells in a standard way into elements, and do special processing near the boundary; see, e.g., [86].

2. Sample the domain boundary computing a set of points. Construct a Delaunay triangulation of those points, and refine it such that the elements respect the boundary; see, e.g., [87,88].

3. Partition the domain by its skeleton and some additional edges or faces into simple subdomains. Triangulate the subdomains in a way that is compatible at the boundaries between them; e.g., [67,68,69].

It is argued in [69] that Approach (1) does not address the essence of the geometric problem and merely postpones it to those spatial cells that intersect the boundary. However, when suitably subdivided, the octree cells containing parts of the boundary have a relatively simple structure.

Approach (2) is attractive in 2D where favorable aspect ratios of the triangles can be guaranteed. However, no similar guarantees exist in the 3D case, as pointed out in [67].

We will sketch first the algorithm of [69], for generating meshes of polygonal domains. The algorithm uses Approach (3), and recognizes automatically where the domain is constricted. In those areas it is usually the case that the mesh needs to be finer than elsewhere.

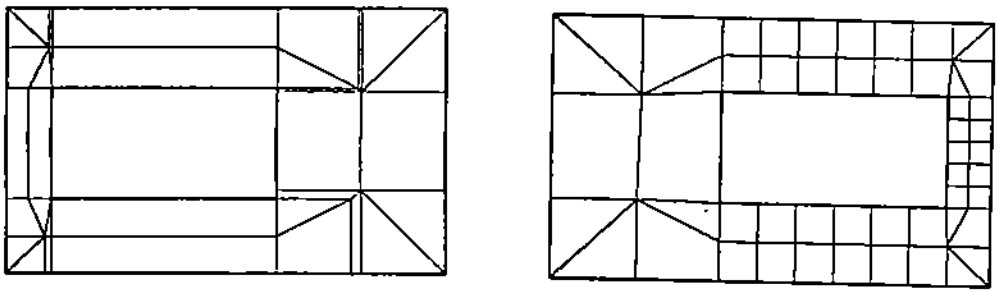


Figure 32: Meshing Algorithm of Srinivasan et al. [69]. Left: Domain with Approximate Skeleton and Initial Partitioning. Right: Result of Long and Short Edge Processing.

In the first stage, the algorithm determines the skeleton of the polygonal domain, and augments it with the shortest paths from certain skeleton points to the boundary. More precisely, at every point of the skeleton where the associated boundary entities change, the shortest paths to the boundary are added. Moreover, the parabolic arcs of the skeleton are replaced with one or two chords according to whether the angle subtended is smaller or greater than 90°. If the arc is replaced with two chords, then the shortest paths from the common chord vertex to the boundary are also added. Figure 32 shows a domain with its skeleton and the result of the first stage on the left. Note that the first stage constructs a domain partition into convex subdomains.

In the second stage, the initial partition is changed by eliminating regions that are very narrow (*slivers*), and by subdividing extended regions bounded by relatively long skeleton edges. Note that these types of region are quantitatively defined in terms of the types and relative lengths of bounding edges [69]. Figure 32 shows the result of this stage on the right.

Next, the partition of the second stage is meshed. Depending on the local geometry and the type of finite-element analysis, some edges bounding subdomains may be subdivided by additional nodes. The resulting mesh optionally may be subjected to Laplace smoothing or other heuristics, producing a final result such as shown in Figure 33.

Armstrong et al. [67] describe a different algorithm for generating meshes of planar domains. The approach is also based on the skeleton, but quadrilateral elements are generated rather than triangular ones. The skeleton is determined by the algorithm of [84]. After the skeleton has been determined, the induced partition of the domain is refined as follows. A concave boundary corner is subdivided by an additional line so as to obtain angles close to 90°, except when the corner is close to 180°. Skeleton edges that lie between complex

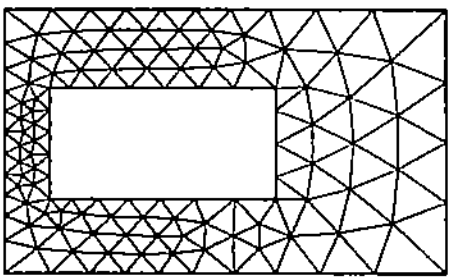


Figure 33: Mesh obtained by Srinivasan et al. after Smoothing.

branch points are also subdivided in an effort to obtain four-sided regions in the partition. The regions are then meshed with quadrilateral elements. See also Figure 34.

F. ON GEOMETRIC TOLERANCING

In many applications, a solid designed for manufacture need not have the exact shape defined by the geometric model. Rather, shapes are allowed to vary within certain tolerances. Expressing these tolerances can be extremely complex, because they may depend on the function of the solid part, and its interrelationship to other parts. Geometric tolerancing studies the subproblem of how to express the geometric variation allowed; e.g., [89].

In the simplest case, one could view the surface of the solid design as a nominal shape, and accept all solids with a shape whose surface is within a

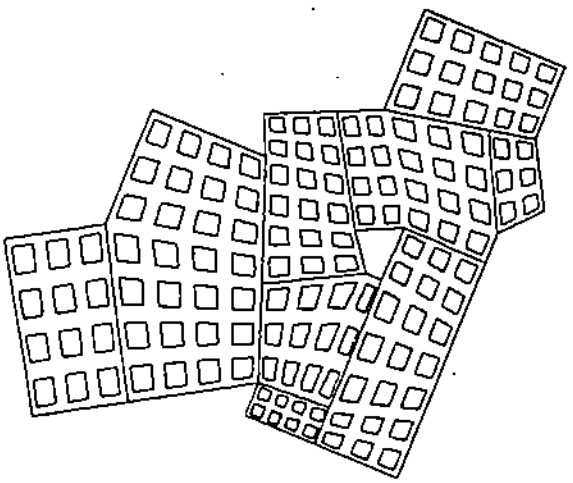


Figure 34: Mesh Obtained by Armstrong et al. [67]

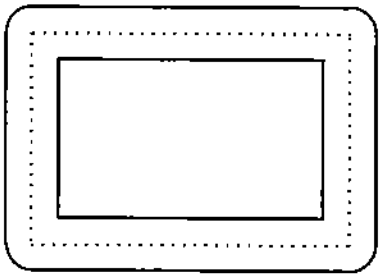


Figure 35: Tolerance Zone of a Nominal Shape

zone of constant thickness surrounding the surface of the nominal shape. See Figure 35 for a two-dimensional example. This appealingly simply idea entails some difficulties in traditional solid representations in that the measurement of distance from a curved boundary is not necessarily a simple computation.

By using a skeleton representation, however, such a distance computation simplifies dramatically. For, since dilations and contractions of shapes correspond to translates of the skeleton in the distance dimension (Section V.A), the point-boundary distance can be measured by comparing corresponding skeleton points in their last coordinate. Also, the function $\mathcal{S}$ discussed before gives a precise distance measurement, so that evaluating this function on the boundary of the variational shape at once reveals by how much the surfaces deviate.

ACKNOWLEDGEMENTS

We are indebted to H. Pottmann for bringing the work on cyclographic maps to our attention, and to R. Lynch, who noted the connection between offsets and the Hamilton-Jacobi equations. The discussions we have had with both on this subject have been most helpful.

References

[1] A. A. Requicha and R. Tilove. *Mathematical foundations of constructive solid geometry: general topology of regular closed sets.* Technical Memorandum TM-27a, Production Automation Project, University of Rochester, Rochester, NY, March 1978.

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

[3] A. A. Requicha and H. B. Voelcker. *Constructive solid geometry.* Technical Report 25, University of Rochester, Rochester, NY, November 1977.

[4] M. Mäntylä. *An introduction to SOLID MODELING.* Computer Science Press, 1988.

[5] K. Weiler. Edge–based data structures for solid modeling in curved–surface environments. *IEEE Computer Graphics & Applications*, 21–40, January 1985.

[6] T. C. Woo. A combinatorial analysis of boundary data structure schemata. *IEEE Computer Graphics & Applications*, 19–27, March 1985.

[7] B. Baumgart. *Winged–Edge Polyhedron Representation.* Technical Report CS–320, Stanford University, 1972.

[8] I. C. Braid, R. C. Hillyard, and I. A. Stroud. *Stepwise construction of polyhedra in geometric modeling*, pages 123–141. Academic Press, New York/London, 1980.

[9] F. Yamaguchi and T. Tokieda. Bridge edge and triangulation approach in solid modeling. In Tosiyasu L. Kunii, editor, *Frontiers in Computer Graphics '84*, pages 44–65, Springer-Verlag, 1985.

[10] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computations of voronoi diagrams. *ACM Transactions on Graphics*, 74–123, April 1985.

[11] P. M. Hanrahan. *Topological Shape Models.* PhD thesis, University of Wisconsin, Madison, WI, 1985.

[12] S. Ansaldi, L. De Floriani, and B. Falcidieno. Geometric modeling of solid objects by using a face adjacency graph representation. In *Proceedings ACM Siggraph '85*, pages 131–139, 1985.

[13] K. Weiler. Polygon comparison using a graph representation. In *Proceedings ACM SIGGRAPH '80*, pages 10–18, 1980.

[14] M. Mäntylä. Boolean operations on 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1):1–29, January 1986.

[15] K. Weiler. *Topological Structures for Geometrical Modeling.* PhD thesis, Rensselaer Polytechnic Institute, Troy, N.Y., August 1986. Technical Report TR-86032.

[16] M. Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, McGill University, 1988.

[17] G. Vaněček Jr. *Set Operations on Polyhedra using Decomposition Methods.* PhD thesis, University of Maryland, 1989.

[18] G. Vaněček Jr. *Protosolid: An inside look.* Technical Report CER–89–26, Purdue University, 1989.

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

[20] I. Carlbom, I. Chakravarty, and D. Vanderschel. A hierarchical data structure for representing the spacial decomposition of 3d objects. *IEEE Computer Graphics & Applications*, 5(4):24–31, 1985.

[21] I. Navazo. *Geometric Modelling of Octree encoded Polyhedrical Objects.* PhD thesis, Universitat Politecnica de Catalunya, Departament de Metodes Informatics, January 1986.

[22] G. M. Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, 1978.

[23] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *ACM Computer Graphics SIGGRAPH '87*, 21(4):153–162, July 1987.

[24] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Conf. Proc. of SIGGRAPH '80*, 14(3):124–133, July 1980.

[25] B. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer-Aided Design*, 22:250–252, 1990.

[26] G. Vaněček Jr. *Towards Automatic Grid Generation using Binary Space Partition Trees.* Technical Report CER–90–6, Purdue University, 1990.

[27] G. Vaněček Jr. A data structure for analyzing collisions of moving objects. In *Proceedings Hawaii International Conference on System Sciences, HICSS–24*, 1991.

[28] A. Bowyer, J. H. Davenport, D. A. Lavender, A geometric algebra system. In D. Kapur, editor, *Integration of Symbolic and Numeric Methods*, MIT Press, 1991.

[29] V. Shapiro and D. Vossler. *Brep to CSG Conversion I: Representations.* Technical Report CPA89-3a, Cornell University, Mechanical and Aerospace Engineering, 1990.

[30] V. Shapiro and D. Vossler. *Brep to CSG Conversion II: Solids.* Technical Report CPA89-4a, Cornell University, Mechanical and Aerospace Engineering, 1990.

[31] R. B. Tilove. *A study of geometric set-membership classification.* Master's thesis, University of Rochester, Rochester, N.Y., November 1977.

[32] A. A. Requicha and H. B. Voelcker. Boolean operations in solid modeling boundary evaluation and merging algorithms. *PIEEE*, 73(1):30–44, January 1985.

[33] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. *Siggraph '86*, 20(4):161–170, August 1986.

[34] C. M. Hoffmann, J. Hopcroft, and M. Karasick. Towards implementing robust geometric computations. In *Proceedings 4th Annual ACM Symposium on Computational Geometry*, pages 106–117, 1988.

[35] C. M. Hoffmann. The problems of accuracy and robustness in geometric computations. *IEEE Computer*, 22:31–42, 1989.

[36] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Dokl. Acad. Nauk SSSR Math*, 146(2):263–266, 1962.

[37] G. Vaněček Jr. Brep-index: a multi-dimensional spatial partitioning tree. In *1st ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Austin Texas, June 1991.

[38] J. R. Rice and R. F. Boisvert. *Solving Elliptical Problems Using ELLPACK.* Springer-Verlag, 1985.

[39] C. M. Hoffmann and J. E. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3:194–206, 1987.

[40] C. M. Hoffmann and J. E. Hopcroft. Model generation and modification for dynamic systems from geometric data. In B. Ravani, editor, *CAD Based Programming for Sensory Robots*, pages 481–492, Springer NATO ASI Series F50, 1988.

[41] R. Bartels, J. Beatty, and B. Barsky. *Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, Cal., 1987.

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

[43] J. Hoschek. *Grundlagen der Geometrischen Datenverarbeitung.* Teubner Verlag, Stuttgart, 1989.

[44] W. Boehm, G. Farin, and J. Kahmann. A survey of curve and surface methods in cagd. *CAGD*, 1:1–60, 1984.

[45] C. M. Hoffmann. Algebraic and numerical techniques for offsets and blends. In W. Dahmen, M. Gasca, S. Micelli, editor, *Computations of Curves and Surfaces*, pages 499–528, Kluwer Academic Publishers, 1989.

[46] C. M. Hoffmann. A dimensionality paradigm for surface interrogation. *CAGD*, 7:517–532, 1990.

[47] V. Chandru, D. Dutta, and C. Hoffmann. Variable radius blending with cyclides. In K. Preiss, J. Turner, M. Wozny, editor, *Geometric Modeling for Product Engineering*, pages 39–57, North Holland, 1990.

[48] J.-H. Chuang and C. Hoffmann. *Curvature Computations on Surfaces in n-Space.* Technical Report CER-90-34, Purdue University, Computer Science, 1990.

[49] D. Dutta and C. Hoffmann. A geometric investigation of the skeleton of CSG objects. In *Proc. ASME Conf. Design Automation*, Chicago, 1990.

[50] C. M. Hoffmann and P. J. Vermeer. Eliminating extraneous solutions in curve and surface operations. *IJCGA*, 1, 1991.

[51] D. Kapur. Geometry theorem proving using Hilbert's nullstellensatz. In *SYMSAC 86*, pages 202–208, Waterloo, Ont., 1986.

[52] D. Kapur. A refutational approach to geometry theorem proving. In D. Kapur and J. Mundy, editors, *Geometric Reasoning*, pages 61–93, M.I.T. Press, 1989.

[53] Wu Wen-Tsün. *On a Projection Theorem of Quasi-Varieties in Elimination Theory.* Technical Report MM Preprints 4, Inst. of Syst. Sci., Academia Sinica, 1989.

[54] C. Bajaj, C. M. Hoffmann, J. Hopcroft, and R. Lynch. Tracing surface intersections. *CAGD*, 5:285–307, 1988.

[55] J.-H. Chuang. *Surface Approximations in Geometric Modeling.* PhD thesis, Purdue University, Computer Science, 1990.

[56] E. Allgower, K. Georg, and R. Miranda. *Computing Real Solutions of Polynomial Systems.* Technical Report, Colorado State University, Mathematics Department, 1990.

[57] E. Allgower and S. Gnutzmann. An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. *SIAM Journal of Numerical Analysis*, 24:452–469, 1987.

[58] B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232, D. Reidel Publishing Co., 1985.

[59] B. Buchberger, G. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Annual Reviews in Computer Science*, 3:85–120, 1988.

[60] A. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[61] J. Pegna and F.-E. Wolter. A simple practical criterion to guarantee second order smoothnes blend surfaces. 1989. manuscript.

[62] W. C. Rheinboldt. On a moving-frame algorithm and the triangulatrion of equilibrium manifolds. In T. Küpper, R. Seydel, H. Troger, editor, *Bifurcation: Analysis, Algorithms, Applications*, pages 256–267, Birkhäuser Verlag, Basel, 1987.

[63] H. Pottmann. Interpolation on surfaces using minimum norm networks. 1990. manuscript.

[64] H. Blum. A transformation for extracting new descriptors of shape. In W. Whaten-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380, MIT Press, Cambridge, MA, 1967.

[65] V. Srinivasan and L. Nackman. Voronoi diagram of multiply connected polygonal domains. *IBM Journal of Research and Development*, 31:373–381, 1987.

[66] U. Montanari. Continuous skeletons from digitized images. *JACM*, 16:534–549, 1969.

[67] C. Armstrong, T. Tam, D. Robinson, R. McKeag, and M. Price. Automatic generation of finite element meshes. In *SERC ACME Directorate Research Conference*, England, 1990.

[68] N. Patrikalakis and H. Gürsoy. *Shape Interrogation by Medial Axis Transform.* Technical Report Memo 90-2, MIT, Ocean Engr. Design Lab, 1990.

[69] V. Srinivasan, L. Nackman, J. Tang, and S. Meshkat. *Automatic Mesh Generation using the Symmetric Axis Transformation of Polygonal Domains.* Technical Report RC 16132, IBM Yorktown Heights, 1990.

[70] A. Requicha. *Mathematical Models of Rigid Solids.* Technical Report Memo 28, University of Rochester, Production Automation Project, 1977.

[71] C. M. Hoffmann. Skeletons, cyclographic maps, and shock waves. 1991. manuscript.

[72] E. Müller and J. Krames. *Die Zyklographie.* Franz Deuticke, Leipzig und Wien, 1929.

[73] J. Rossignac and A. Requicha. Offsetting operations in solid modeling. *CAGD*, 3:129–148, 1984.

[74] K. Strubecker. *Differentialgeometrie III.* Sammlung Göschen Bd. 1180/1180a, Walter de Gruyter, Berlin, Germany, 1969.

[75] E. Müller. Zusammenhang zwischen relativer Flächentheorie und einer Verallgemeinerung der Zyklographie. *Jahresber. Dtsche. Math. Ver.*, 155–160, 1923.

[76] A. Goodman. A partial differential equation and parallel plane curves. *American Mathematical Monthly*, 71:257–264, 1964.

[77] G. Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, Wellesley, MA, 1986.

[78] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[79] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13:471–494, 1966.

[80] F. Preparata. The medial axis of a simple polygon. In *Proc. 6th Symp. Mathematical Foundations of Comp. Sci.*, pages 443–450, 1977.

[81] D. T. Lee. Medial axis transformation of a planar shape. *IEEE Trans. Pattern Anal. and Mach. Intelligence*, PAMI-4:363–369, 1982.

[82] C. M. Hoffmann. How to construct the skeleton of CSG objects. In A. Bowyer and J. Davenport, editors, *The Mathematics of Surfaces IV*, Oxford University Press, 1990.

[83] A. Wallis, D. Lavender, A. Bowyer, and J. Davenport. Computing voronoi diagrams of geometric models. In *IMPA Workshop on Geometric Modeling*, Rio de Janeiro, 1991.

[84] M. Price, T. Tam, C. Armstrong, and R. McKeag. Computing the branch points of the voronoi diagram of a object using a point Delaunay triangulation algorithm. 1991. Draft manuscript.

[85] F. Preparata and M. Shamos. *Computational Geometry.* Springer Verlag, New York, 1985.

[86] M. Shepard, F. Guerinoni, J. Flaherty, and P. Baehmann Finite octree mesh generation for automated adaptive three-dimensional flow analysis. In *Proc. 2nd Intl. Conf Num. Grid Generation in Computational Fluid Mechanics*, pages 709–718, Miami, 1988.

[87] P. Chew. *Guaranteed-Quality Triangular Meshes.* Technical Report 89-893, Cornell University, Computer Science, 1989.

[88] N. Sapidis and R. Perucchio. Domain Delaunay tetrahedrization of arbitrarily shaped curved polyhedra defined in a solid modeling system. In *Proc. ACM/SIGGRAPH Symp Solid Modeling Found and CAD/CAM Applic*, Austin, Tex, 1991.

[89] A. A. Requicha. Toward a theory of geometric tolerancing. *Intl. J. of Robotics Research*, 2:45–60, 1983.