# Dimension-Independent Modeling with Simplicial Complexes

A. PAOLUZZI, F. BERNARDINI, C. CATTANI, and V. FERRUCCI
Università "La Sapienza"

Dealing with simplicial decompositions which are dimension independent allows for the convergence of disparate viewpoints from computer graphics, solid and geometric modeling. In this framework it is possible to treat in a unified manner several geometric problems, such as solid modeling of articulated objects, simplicial approximation of curved manifolds, motion encoding and interference detection, free configuration space computation, and graphical representation of multidimensional data. In the paper the authors describe the winged scheme, a simple representation based on simplicial decompositions, which can be used for linear polyhedra of any dimension and which allows for "solid" approximation of curved manifolds when combined with curved maps (e.g., NURBS). Various operators and algorithms are discussed, including boundary evaluation, linear and screw extrusion, grid generation, simplicial maps, and set operations. A simple manipulation language is also introduced, and some nontrivial examples are discussed.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representations*; *geometric algorithms, languages, and systems*; J.6 [**Computer-Aided Engineering**]: *computer-aided design (CAD)*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Boolean operations, design languages, extrusion, $n$-dimensional triangulation, polyhedra, representation, simplicial complexes, simplicial maps

## 1. INTRODUCTION

*Statement of the problem and motivation.* In this paper several methods of 3D graphics and modeling are extended to work with polyhedra of dimension $d$ in $\mathbb{R}^n$. A simplicial-based representation is presented, and a symbolic description is given for structured sets of dimension-independent polyhedra, along with operators for object manipulation and a language to be used as an interface to the modeling system. With the presented methods it is possible to unify the geometric treatment of problems arising in different areas, including solid modeling of articulated systems, simplicial approximation of curved manifolds, motion encoding, and interference detection between mobile ob-

This work was partially supported by the PF "Edilizia" of the Italian National Research Council under grants nr. 89.00282.64, 90.01702.64, and 91.02332.64.
Authors' addresses: A. Paoluzzi, F. Bernardini, and V. Ferrucci, Dip. di Informatica e Sistemistica, Università "La Sapienza," Via Buonarroti 12, 00185 Roma, Italy; C. Cattani, Dip. di Matematica, Università "La Sapienza," P.le A. Moro 5, 00185 Roma, Italy.

jects and the environment (both static and dynamic). In addition, it becomes also possible to represent—in a single higher-dimensional polyhedron—the whole set of configurations assumed by a mobile system with some degrees of freedom.

Dimension (or order) $d$ of a polyhedron is the highest dimension of the simplices contained in it, that is the maximum number of affinely independent vertices, which span some polyhedron portion, minus one. The dimension of the embedding space coincides with the number $n \geq d$ of the coordinates of vertices. The range of interest in the paper is restricted to piecewise-linear polyhedra, which can be collected into structures together with operators on structures. The representation scheme we propose for polyhedra is based on simplicial complexes; curved polyhedra can be linearly approximated by simplicial maps, which are pairs constituted by a simplicial complex (often a $d$-dimensional grid) and a homeomorphic map.

The presented approach allows for the convergence of viewpoints from solid and curved geometric modeling. In this paper, parametric representations of curved manifolds and NURBS are defined as maps between spaces of polyhedra. In other words, a polyhedral approximation—with a desired resolution—of a curved object is seen as the mapped image of a suitable polyhedron in the space of parameters. Since a "solid" representation of the mapping argument is used, a "solid" representation of the resulting object is obtained. Where an improved resolution is required, it is sufficient to locally refine (e.g., using baricentric subdivision) the simplicial representation of the parametric argument. The separation of the mapping definition from its application to some polyhedral argument is very useful; in particular, it makes possible: (a) to map a lower-dimensional subset of the parametric domain, e.g., to evaluate a curve defined on a surface; (b) to generate a trimmed patch, e.g., a trimmed NURBS, given a polyhedral approximation of its trimmed parametric domain; (c) to build a multigrid approximation, where grids of different resolution are defined on subsets of the mapping domain.

Simplicial decompositions have seldom been seriously considered for the representation of objects in a general-purpose solid modeler because of their main drawback: the fragmentation caused by Boolean operations. It should be pointed out, that when dealing with multidimensional objects, different approaches such as B-reps with Boolean operators based on boundary classification suffer grave efficiency problems, as they need to recurse on all elements of the boundary, from maximum- to zero-dimensional ones. On the contrary, the use of simplicial complexes allows us to approximate in a natural way curved manifolds, to exploit the straightforwardness of the data structure for the design of simple algorithms, and to take advantage of linear-programming methods for the solution of geometrical problems, such as intersection, interference detection, and point classification. Furthermore, the use of simplicial decompositions can be considered as a preliminary step toward the introduction of more general complexes, where cells are convex or affine sets, both of which allow as well the use of linear-programming techniques while maintaining compact storage.

The extension of solid-modeling techniques to higher dimensions also intro-

duces very simple and general methods for the representation and rendering of fields, since a field over a manifold can be modeled by embedding the manifold in a higher-dimensional space. Consideration of the field variation within a simplex is important for achieving the desired accuracy in the piecewise approximation with simplicial complexes. In this setting, the evaluation of "geometrical queries" over fields, e.g., the computation of subsets of points which satisfy a predicate calculus formula (where atomic sentences are relational expressions with field variables), can be reduced to the computation of Boolean formulae between $d$-dimensional manifolds. As an example, consider that an iso-valued surface of the temperature in a solid results from the intersection between a temperature hyperplane $T = const$ (a 3-manifold in the $x$, $y$, $z$, $T$ space) and a model of the solid embedded in such 4D space, where the $T$ coordinate has been added to the vertices of the solid.

In addition, we believe that the present approach may help to reduce the gap between solid modeling and engineering computation with finite-element codes and multigrid methods; e.g., the domain discretization used in a FEM computation can be easily translated in a decompositive object representation, while the field values (both scalar and vector- or tensor-valued) computed at the "nodes" can be stored as additional coordinates of object vertices, when the grid used for the field computation is given. The reverse is also possible, of course: the object representation obtained with simplicial maps and set operations can be used as a domain decomposition for field computations.

In general, the triangulations we use are not minimal. The rationale for this hypothesis is that triangulations of polyhedra currently used in CAD/CAE applications are strongly redundant. The modeling triangulations can be locally improved (changed or detailed) for both robustness reasons and for the needs of finite-element computations, where triangulations are often required as a refinement of some redundant mesh. The cost of generating an optimal triangulation for a nonconvex multidimensional polyhedron may be prohibitive. A recent result by Ruppert and Seidel [60] has shown that the tetrahedralizability problem is NP-complete (even) in 3D space. This implies that the operators to be subsequently described must directly generate an (eventually redundant) triangulation of the resulting polyhedron—see, e.g., boundary and extrusion operators.

*Previous work.* New needs for visualization, manipulation, and reasoning with multidimensional surfaces and solids are emerging from scientific visualization, statistical graphics, modeling and simulation, and robotics. The extension of basic techniques of computer graphics for the wireframe display of $n$-dimensional objects was provided by the early works of Noll [42, 43]. Burton and Smith [12] describe a hidden-line algorithm for higher-dimensional scenes. Armstrong and Burton [1] present various graphical techniques based on visual cues, for improving the comprehension of "hyper-dimensional" objects. Banchoff [4] discusses the real-time presentation of four-dimensional objects and extends [5] standard algorithms of computer graphics (like Gouraud shading), in order to obtain a good rendering of 4D

objects. Glassner [28] gives efficient techniques for ray tracing of animated scenes, where static objects in a 4D space-time are rendered instead of dynamically moving objects in a 3D space. Graphical display of rational algebraic hypersurfaces is studied by Bajaj [3], and visualization techniques for surfaces in 4D space are presented by Hoffmann and Zhou [32].

Solid modeling, usually performed in three dimensions [54, 55], was recently extended to the modeling of $n$-dimensional solids ($n \geq 4$). Cameron [13, 14] started working in 4D in order to detect collisions between obstacles and a robot moving in a known environment. Cattani and Paoluzzi [16] discussed the extrusion of an object into the space-time, which exhibits more convenient properties than the sweeping within the embedding space. They also presented preliminary ideas about extrusion and set operations in $\mathfrak{R}^n$ using simplicial complexes. The higher-dimensional approach to the description of objects and motions was already proposed by Lozano-Perez and Wesley [36], where each degree of freedom is associated to an independent dimension of the embedding space. Putnam and Subrahmanyam [52] and Montgomery [39] construct a CSG approach for multidimensional linear solids. Putnam and Subrahmanyam define set operators through boundary classification techniques, assuming no distinction between different entities (vertices, edges, faces, and so on). Their representation consists of a list of oriented boundary elements, where each element recursively contains the description of its boundary. This approach is very simple, but the description is not efficient neither in space nor in time. In fact, depending on the number of elements it belongs to, each boundary element is stored in memory many times. Furthermore, since topological relationships are not explicitly stored, any topological query is answered in linear time with the size of the object description.

A complete characterization of the adjacencies in a cell decomposition of an $n$-polyhedron is given by Lienhardt [34], who extends to $n$-dimensions the concept of a topological map usually given in two dimensions, where it is used to establish dualities between graphs embedded into surfaces. Brisson [10] defines the cell-tuple structure, where a complete representation both of incidence between various order cells and of cell adjacency in a subdivided manifold of dimension $n \geq 1$ is given. Such a set of cell-tuples is a kind of "vertical" description of the object topology, where each cell-tuple is constituted by a sequence of incident cells of various dimensions $d$ ($0 \leq d \leq n$) and where each $d$-cell is contained in the boundary of the $(d + 1)$-cell. Tuples are implemented as $(n + 1)$ arrays of pairs of indices, where the first index points to the constituting cells, and the second one is used to implement a switch operator to an adjacent cell-tuple. Brisson shows that cell-tuple representation is equivalent, in dimension 2 and 3, to the quad-edge data structure of Guibas and Stolfi [29], and to the facet-edge data structure of Dobkin and Laszlo [21]. The cell-tuple representation portrays in a uniform way the dual subdivision of a topological manifold and can easily calculate ordering information from switch operators (see also [35]). Brisson's approach is elegant and powerful; a major drawback is the factorial growth with $n$ of the cardinality of the set of cell-tuples.

Nef [41] and later Bieri and Nef [9] conceive polyhedra as finite Boolean expressions using halfspaces in order to perform set operations on $n$-dimensional polyhedra. Their approach can be summarized as follows: (a) the concept of "locally adjoined pyramid" is introduced, which allows us to conveniently define the notion of face of a nonconvex polyhedron; (b) a set of elementary operations over pyramids is given, in order to define an abstract data type pyramid; (c) algorithms over pyramids lead to the implementation of set operations (complement, union, intersection, difference) over polyhedra; (d) pyramids are in turn represented by means of subsets of the cell complex induced in $\Re^n$ by the set of the face hyperplanes. Bieri and Nef's approach is very interesting, but one drawback is that the cell complex implementation of pyramids is $O(m)$ in space for each cell, where $m$ is the number of hyperfaces. A variation of CSG oriented to cell decompositions has been used by Zhang [67] in the 3D EASYCAD modeler.

Rossignac and O'Connor [57] developed the powerful dimension-independent SGC (Selective Geometric Complexes) representation, in the attempt to extend the geometric coverage of geometric and solid modeling over curved pointsets with internal structures and incomplete boundaries. Their representation supports mixed-dimensional collections of disjointed open cells of real algebraic varieties. Three fundamental algorithms concerning compatible subdivision, selection, and simplification of cells are used to easily define standard solid-modeling operations, such as interior, closure, regularized Boolean, or user-defined operations. Constructive Non-Regularized Geometry (CNRG), a modification of CSG, has been proposed by Rossignac and Requicha [58] to allow for the representation of dimensionally inhomogeneous pointsets of $\Re^n$. In a CNRG tree, the leaves correspond to regions which are nonregular semianalytic subsets of $\Re^n$, and internal nodes represent CNRG objects obtained by applying Boolean, topological, or filtering operators to the objects represented by the child nodes. Operators are carefully defined to ensure that information is not lost in manipulating objects of mixed dimensionality and/or with internal structures.

The work of the CAD Group at the University of Rome focused on the use of simplicial complexes in solid modeling with the development of the 3D solid modeler *Minerva* [17, 45], where a simplicial decomposition of the boundary was used. Such a representation has been generalized to 4D [15] and to the $n$D winged representation [16]. The winged representation is simple and useful for actual implementations: most of the materials presented here have been implemented in a prototype modeler called $Simple_X^n$ [6, 22, 49]. Various papers describe algorithms based on such a representation for motion planning [44], extrusion and boundary evaluation [24], integration of polynomials over simplicial complexes [7], Boolean operations [23], spatial indices [25], simplicial maps and rendering of fields over manifolds [48], and polyhedral approximation of parametric and NURBS maps [49]. A calculus over polyhedra (with higher-level operators, not described in this paper) has been introduced in the context of the PLASM design language [46, 47] based on Backus' functional language FL [2].

*Paper overview.* In Section 2, a minimum amount of notation and notions from algebraic topology, needed in the following, is introduced. Then, the winged representation is defined and discussed with examples. In Section 3, the syntactic rules of a simple description language are given, and the evaluation of a structure is discussed. In Section 4, the semantics of several classes of operators is discussed, including: affine transformations; connected-components extraction; boundary evaluation; straight, linear and screw extrusion; grid and map operators for piecewise-linear approximation of curved manifolds; NURBS maps and projection. Set operations are illustrated in Section 5, point-in-polyhedron testing and integration over a polyhedron in Section 6. In the conclusion the features of the presented approach are summarized; the main open problems are discussed, and some possible extensions are outlined. Examples of the winged representation of $d$-dimensional cubes are given in Appendix A, and an application to the modeling and visualization of a time-varying field over a manifold is contained in Appendix B. Finally, in Appendix C, two high-level operators for the description of the set of configurations of articulated rigid systems are introduced, and an example of free configuration space computation is provided. A glossary of mathematical symbols concludes the paper.

## 2. THE WINGED REPRESENTATION SCHEME

In the following, we give a brief summary of concepts and notation used in the following sections. An easily readable treatment of algebraic topology can be found in [27]. Other useful references are [33], [53], [59], and [65].

### 2.1 Some Background

Solid and geometric modeling find the origin of their methods within the realms of numerical analysis, differential geometry, and real algebraic geometry. Only few attempts to cross the edges of these domains can be found in the literature (see, e.g., [53], and more recently [10] and [34]). In our approach we make use of a few fundamental concepts from algebraic topology. The source of this paragraph is Dieudonné [20].

The study of compact triangulated spaces started with Poincaré's papers [51]. Given a triangulation $T$ consisting of cells homeomorphic to convex polyhedra, $p$-chains were defined as formal linear combinations with integer coefficients of oriented cells of $T$. To obtain the boundary operator on a triangulation it was sufficient to formally combine the boundary operators defined on cells of $T$. Poincaré [20] was guided in the definition of the boundary operator by the intuitive idea that "a boundary has no boundary."

As any subdivided triangulation of every cell into smaller cells gives the same "homology", it is sufficient to consider *simplicial cell complexes*, where each cell is a simplex, a sort of $d$-dimensional triangle in $\mathbb{R}^n$ ($d \leq n$). To define such a simplex it is sufficient to know its vertices. Brouwer [11] introduced the notion of *simplicial mapping* $f: K \to L$, where $K$ and $L$ are simplicial complexes, such that, for each simplex $\sigma$ of $K$, $f(\sigma)$ is a complex of

$L$, and the restriction of $f$ to $\sigma$ is affine. This implies that $f$ is entirely determined by the images $f(v_i)$ of vertices of $K$, which are vertices of $L$. An even more important idea of Brouwer concerned *simplicial approximation*: for two triangulated spaces $X$, $Y$ and *any* continuous map $f: X \to Y$ it is possible, for each $\epsilon > 0$, to find subdivisions of the triangulations of $X$ and $Y$ and a simplicial map $g: X \to Y$ relative to these subdivisions such that the distance of $f$ and $g$ is at most $\epsilon$ and such that $f$ and $g$ are "homotopic." It was proved around 1930 that $C^1$ manifolds and algebraic varieties with singularities can be triangulated, as conjectured by Poincaré.

In this paper we will not be involved with the central concepts of homology and homotopy, but just with the combinatorial instrumentation of simplicial complexes and with simplicial mapping and approximation.

### 2.2 Preliminaries

The join of two sets $P$, $Q \subset \Re^n$ is the set $PQ = \{\gamma p + \lambda q,\ p \in P, q \in Q\}$, where $\gamma$, $\lambda \in \Re$, $\gamma$, $\lambda \geq 0$, and $\gamma + \lambda = 1$. The join operation is associative and commutative. A simplex $\sigma \subset \Re^n$ of order $d$, or $d$-simplex, is the join of $d + 1$ affinely independent points, called vertices. The $n + 1$ points $p_0, \ldots, p_n$ are affinely independent when the $n$ vectors $\boldsymbol{p}_1 - \boldsymbol{p}_0, \ldots, \boldsymbol{p}_n - \boldsymbol{p}_0$ are linearly independent. A $d$-simplex can be seen as a $d$-dimensional triangle: 0-simplex is a point, 1-simplex is a segment, 2-simplex is a triangle, 3-simplex is a tetrahedron, and so on. Any subset of $s + 1$ vertices ($0 \leq s \leq d$) of a $d$-simplex $\sigma$ defines an $s$-simplex, which is called $s$-face of $\sigma$.

A simplicial complex is a set of simplices $\Sigma$, verifying the following conditions: (a) if $\sigma \in \Sigma$, then any $s$-face of $\sigma$ belongs to $\Sigma$; (b) if $\sigma$, $\tau \in \Sigma$, then either $\sigma \cap \tau = \emptyset$, or $\sigma \cap \tau$ is an $s$-face of $\sigma$ and $\tau$. Geometric carrier $[\Sigma]$ is the pointset union of simplices in $\Sigma$.

The order of a complex is the maximum order of its simplices. A complex $\Sigma^d$ of order $d$ is also called a $d$-complex. A $d$-complex is regular if each simplex is an $s$-face of a $d$-simplex. Two simplices $\sigma_1$ and $\sigma_2$ in a complex $\Sigma$ are $s$-adjacent if they have a common $s$-face; they are $s$-connected if a sequence of simplices in $\Sigma$ exists, beginning with $\sigma_1$ and ending with $\sigma_2$, such that any two consecutive terms of the sequence are $s$-adjacent. In the following, face and adjacency (without prefix) of a $d$-simplex stand for $(d-1)$-face and $(d-1)$-adjacency. $K^s(\Sigma^d)$ ($0 \leq s \leq d$) denotes the set of $s$-simplices belonging to $\Sigma^d$, and $|K^s|$ denotes their number. With some abuse of language, we call $K^s$ the $s$-skeleton. The set of vertices of $\Sigma^d$ is therefore $K^0(\Sigma^d)$, and the set of $d$-simplices is $K^d(\Sigma^d)$.

The set of all linear $d$-polyhedra embedded in $\Re^n$ will be denoted as $\mathscr{P}^{d,n}$. A polyhedron $P \in \mathscr{P}^{d,n}$ coincides with the geometric carrier of a simplicial $d$-complex, and we write $P = [\Sigma^d]$. As an extreme example, $o \in \mathscr{P}^{0,0}$ is the 0-polyhedron consisting of a single point—the set $\mathscr{P}^{0,0}$ is a singleton and contains only $o$. A polyhedron is regular if any associated complex is regular. The boundary $\partial P$ of a regular polyhedron $P = [\Sigma^d]$ is the geometric carrier of a $(d-1)$-complex whose $(d-1)$-simplices are faces of exactly one $d$-simplex in $\Sigma^d$. Notice that if $P$ is regular then $\partial\partial P = \emptyset$. The set of vertices of a polyhedron $P = [\Sigma^d]$ is defined to be $K^0(\Sigma^d)$ and is concisely indicated by

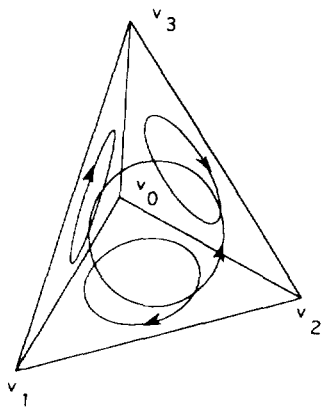



Fig. 1. Coherent orientation of the 2-faces of a 3-simplex.

$K^0(P)$. Notice that such a set of vertices may be redundant (e.g., lying in the interior), according to the use of "nodes" in FEM decompositions. For sake of brevity, we will occasionally soften the distinction between the polyhedron $P \in \mathcal{P}^{d,n}$ and an associated complex: the meaning of $K^s(P)$ is "the $s$-skeleton of $\Sigma^d$ such that $[\Sigma^d] = P$".

The choice of an ordering for the vertices of a simplex implies its orientation, according to the even or odd permutation of the ordering. The two opposite orientations will be denoted as $+\sigma$ and $-\sigma$. A complex is orientable when all its simplices can be coherently oriented. The oriented $(d-1)$-faces of the $d$-simplex $\sigma_i = \langle v_{i,0}, \ldots, v_{i,d} \rangle$ are given by the formula:

$$\sigma_{i,j} = (-1)^j \langle v_{i,0}, \ldots, v_{i,j-1}, v_{i,j+1}, \ldots, v_{i,d} \rangle, \quad 0 \le j \le d, \tag{1}$$

where $\sigma_{i,j}$ and $v_{i,j}$ denote the $j$th face and the $j$th vertex of $\sigma_i$, respectively. A similar notation for the oriented $(d-1)$-faces of a $d$-simplex is attributed by Dieudonné [20] to Eilenberg and Mac Lane. Two adjacent simplices are coherently oriented when their common face has opposite orientations (see Figure 1).

### 2.3 Winged Representation

A complete representation scheme [54] for a regular $d$-complex can be simply obtained by giving the list of its maximum order simplices, that is, its $d$-skeleton $K^d$, since any lower-order simplex can be extracted by repeated applications of Formula (1). Although complete, this representation is highly inefficient, as it takes at least $O(|K^d|)$ to answer any topological query. It is useful to enrich the scheme by explicitly storing maximum-order adjacencies, since this improves the efficiency of the traversal of the representation. Thus we state the following:

*Definition* 1. The Winged Representation $\mathcal{W}(\Sigma^d)$ of a regular complex $\Sigma^d$ is a pair $(K^d, \mathcal{A})$, where $K^d$ is the $d$-skeleton of the complex, and $\mathcal{A}: K^d \to \times_{d+1}(K^d \cup \{\perp\})$ is an adjacency function, which associates each $d$-simplex with the $(d+1)$-tuple of $d$-simplices that are $(d-1)$-adjacent to it. The

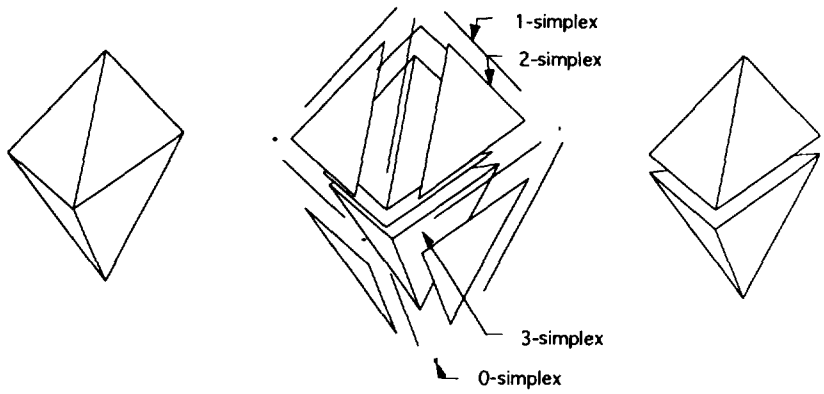



Fig. 2. A polyhedron with an associated simplicial complex, and its winged representation.

complete representation of adjacencies is restricted to the (quite large) subclass of regular complexes such that a $(d-1)$-face is adjacent at most to 2 $d$-simplices.

Here, the symbol $\perp$ stands for "no adjacency," and the notation $\times_{d+1} S$ represents the Cartesian product of $d+1$ instances of the set $S$. The notation $\mathscr{A}_h(\sigma_p)$ will be used to indicate the $h$th value in the adjacency tuple $\mathscr{A}(\sigma_p)$, i.e., the $d$-simplex adjacent to $\sigma_p$ along its $h$th face.

The winged representation of a complex (see Figure 2) can be implemented with a two-dimensional array of reals (the coordinates of vertices) and a pair of two-dimensional arrays of pointers (to represent topology). If $\Sigma^d$ is embedded in $\Re^n$, each of its $|K^0|$ vertices has $n$ real coordinates, and each element in $K^d$ requires $2(d+1)$ pointers to its vertices and its adjacencies. Therefore, the winged representation $\mathscr{W}(\Sigma^d)$ of a complex $\Sigma^d$ embedded in $\Re^n$ is stored in a memory area of size

$$n|K^0|(\#r) + 2(d+1)|K^d|(\#p),$$

where $\#r$ and $\#p$ are the sizes of the memory representation of a real number and of an integer pointer, respectively. Finding a common face between two given $(d-1)$-adjacent $d$-simplices requires to scan the adjacency pointer array for one of the two simplices, with a cost of $O(d)$.

The winged representation can be seen as a paradigm which unifies diverse kinds of representation schemes for polyhedra (see Figure 3). In particular, it can be used as a boundary representation (as in the 3D modeler *Minerva* [45]) and as a decompositive representation, needed to perform finite-element analysis in CAD/CAE applications and useful to execute extrusion operations.

*Definition* 2. The Decompositive Winged Representation $\mathscr{W}(P)$ of a polyhedron $P \in \mathscr{P}^{d,n}$, such that $P = [\Sigma^d]$, is given by $\mathscr{W}(\Sigma^d)$.

*Definition* 3. The Boundary Winged Representation $\mathscr{W}_\partial(P)$ of a polyhedron $P \in \mathscr{P}^{d,n} (d \geq 2)$, such that $\partial P = [\Sigma^{d-1}]$, is given by $\mathscr{W}(\Sigma^{d-1})$.

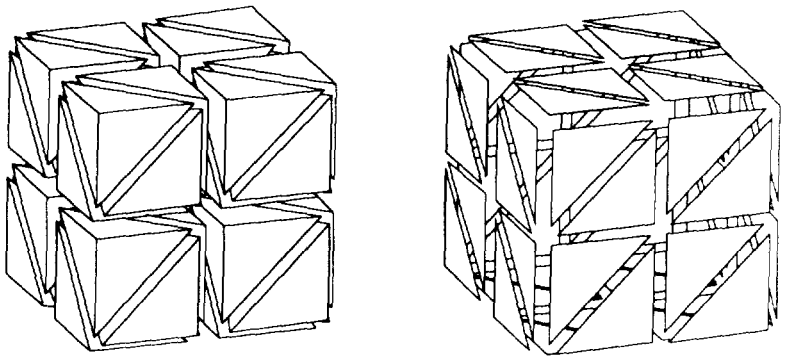


Fig. 3. Decompositive versus boundary representation.

Notice that a boundary representation of $P \in \mathcal{P}^{d,n}$ makes sense only when $d \geq 2$, as only in this case the $(d-2)$-adjacencies used in the $\mathcal{W}_\partial(P)$ representation are well defined. The evaluation of $\mathcal{W}_\partial(P)$, given $\mathcal{W}(P)$, is discussed in Section 4.3. Notice also that given $P \in \mathcal{P}^{d,n}$, a boundary winged representation of $P$ coincides with a decompositive winged representation of $\partial P$, the boundary of $P$:

$$\mathcal{W}_\partial(P) = \mathcal{W}(\partial P).$$

The $\mathcal{W}$ representation scheme is not unique, since it is possible to associate different triangulations (of the interior or of the boundary) to the same polyhedron; it is complete because only one polyhedron coincides with the geometric carrier of any $\mathcal{W}$ representation. The represented polyhedra, in both cases, may be disconnected, and each connected part may be bounded by more than one shell. The winged representation is somewhat similar to Brisson's cell-tuple [10], because both representations have tuples of length $2(d+1)$ pointing to cells and to adjacent cells. Differently from the cell-tuple data structure, the winged representation is "horizontal," maintaining an explicit storage just of $d$-simplices and of their adjacencies, while lower-order oriented simplices can be computed recursively.

In considering 3D boundary representations, one may ask why to use a triangulation of the boundary. It is the authors' opinion that to use a simplicial decomposition of the boundary makes sense because it does not require to describe recursively all the boundary faces of any dimension. In fact a 2D face can be either represented by a decompositive representation (a 2D triangulation) or by using an edge list (a 1D simplicial description), which are both of linear size with respect to the number of vertices of the face. One may argue, in this case, that there is no reason to prefer one representation to another, but when the dimension of the object grows, the difference becomes evident. In fact, in a decompositive representation of the boundary only maximum-order simplices appear. Conversely, by using a "pure" boundary representation, it is necessary to describe the boundary by extracting its hyperfaces, then to describe the boundary of each hyperface, and so on, in a recursive way.

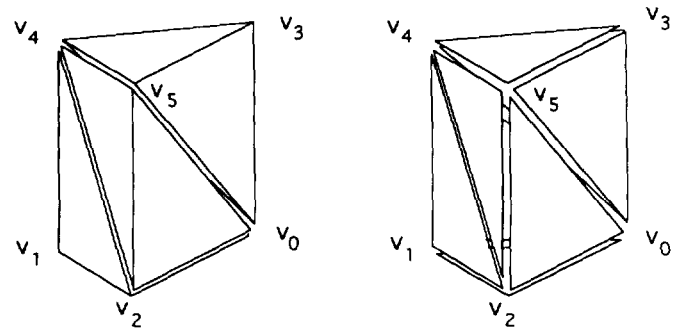



Fig. 4. A 3D wedge $P$: $\mathscr{W}(P)$ and $\mathscr{W}_\partial(P)$.

*Example* 1. The decompositive winged representation $\mathscr{W}(P)$ of the wedge $P$ in Figure 4, along with the boundary winged representation $\mathscr{W}_\partial(P)$, are given by listing the maximum-order simplices and the relative adjacency tuples. Remembering that if $\sigma_{p,h}$ is a boundary face, i.e., if $\sigma_p$ has no adjacent simplex along its $h$th face $\sigma_{p,h}$, then $\mathscr{A}_h(\sigma_p)$ is conventionally set to $\perp$, we have:

$$\mathscr{W}(P) = \begin{cases} \sigma_0 = +\langle v_1, v_2, v_0, v_4\rangle & \mathscr{A}(\sigma_0) = \langle \sigma_1, \perp, \perp, \perp\rangle \\ \sigma_1 = +\langle v_2, v_0, v_4, v_5\rangle & \mathscr{A}(\sigma_1) = \langle \sigma_2, \perp, \perp, \sigma_0\rangle \\ \sigma_2 = +\langle v_0, v_4, v_5, v_3\rangle & \mathscr{A}(\sigma_2) = \langle \perp, \perp, \perp, \sigma_1\rangle \end{cases}$$

where, for example, $\mathscr{A}_0(\sigma_0) = \sigma_1$, with $\sigma_{0,0} = +\langle v_2, v_0, v_4\rangle$, and so forth. For the boundary winged representation $\mathscr{W}_\partial(P)$:

$$\mathscr{W}_\partial(P) = \begin{cases} \sigma_{0,1} = -\langle v_1, v_0, v_4\rangle & \mathscr{A}(\sigma_{0,1}) = \langle \sigma_{2,2}, \sigma_{0,2}, \sigma_{0,3}\rangle \\ \sigma_{0,2} = +\langle v_1, v_2, v_4\rangle & \mathscr{A}(\sigma_{0,2}) = \langle \sigma_{1,1}, \sigma_{0,1}, \sigma_{0,3}\rangle \\ \sigma_{0,3} = -\langle v_1, v_2, v_0\rangle & \mathscr{A}(\sigma_{0,3}) = \langle \sigma_{1,2}, \sigma_{0,1}, \sigma_{0,2}\rangle \\ \sigma_{1,1} = -\langle v_2, v_4, v_5\rangle & \mathscr{A}(\sigma_{1,1}) = \langle \sigma_{2,0}, \sigma_{1,2}, \sigma_{0,2}\rangle \\ \sigma_{1,2} = +\langle v_2, v_0, v_5\rangle & \mathscr{A}(\sigma_{1,2}) = \langle \sigma_{2,1}, \sigma_{1,1}, \sigma_{0,3}\rangle \\ \sigma_{2,0} = +\langle v_4, v_5, v_3\rangle & \mathscr{A}(\sigma_{2,0}) = \langle \sigma_{2,1}, \sigma_{2,2}, \sigma_{1,1}\rangle \\ \sigma_{2,1} = -\langle v_0, v_5, v_3\rangle & \mathscr{A}(\sigma_{2,1}) = \langle \sigma_{2,0}, \sigma_{2,2}, \sigma_{1,2}\rangle \\ \sigma_{2,2} = +\langle v_0, v_4, v_3\rangle & \mathscr{A}(\sigma_{2,2}) = \langle \sigma_{2,0}, \sigma_{2,1}, \sigma_{0,1}\rangle \end{cases}$$

Notice that, since $\partial P$ is a closed (without boundary) polyhedron, each 2-simplex of the boundary has always three 2-simplices 1-adjacent to it.

Let us compare the sizes of three different representations of the topology of the wedge $P$ in the above example, without considering the size of the set of vertices, which is the same for all of them. Note that the topology description in $\mathscr{W}(P)$ has half the size than in $\mathscr{W}_\partial(P)$, and one third of the size of the winged-edge representation of the same object.

*Winged-edge representation.* The memory size of the main relation of Baumgart's winged-edge representation is $8|E|(\#p) = 72(\#p)$, where $E$ is the set of original edges.

*Boundary winged representation.* The size of the topology of the representation $\mathcal{W}_b(P)$ is $2 \cdot 3|K^2|(\#p) = 48(\#p)$.

*Decompositive winged representation.* The size of the topology of the representation $\mathcal{W}(P)$ is $2 \cdot 4|K^3|(\#p) = 24(\#p)$.

In the following we are mainly concerned with decompositive representations of polyhedra.

## 3. DESCRIPTION LANGUAGE

We introduce a simple language for the definition and manipulation of structures, which are defined as ordered sequences of expressions containing polyhedra, operators, and structures, together with angle delimiters. A more powerful language, embedded within Backus' FL functional language, is described in [46] and [47]. The aim of the simple language presented here is to give a framework for the modeling tools to be described in the following and to unify the presentation of examples.

A structure is defined as follows, using EBNF notation:

*structure* ::= *polyhedron*|*operator structure*|"⟨"*structure*{","*structure*}"⟩"

where *polyhedron* is a primitive object. If $\langle S_1, S_2, S_3 \rangle$ is a structure, then $S_1$, $S_2$, $S_3$ are called its substructures or elements. One can write *structure-identifier* " = " *structure* to assign a name to a structure.

In the paper we distinguish between operators, which work on structures, and mathematical operators, which work on the underlying polyhedra. Composition of mathematical operators is denoted by the infix symbol $\circ$. The syntax for operators (where, as usual, square brackets denote optional terms) is

*operator* ::= *generic-operator*[$^{indices}$][ "("*parameters*")"]|*set-operator*.

A *generic-operator* takes a structure as argument and, when evaluated, results in a structure. Let $S$ be a structure. During the evaluation of *generic-operator* over $S$, three cases may occur:

—$S$ is a polyhedron: the evaluation takes place, and the polyhedron is transformed according to the semantics of the underlying mathematical operator, resulting in a new polyhedron or structure.

—$S$ is a structure beginning with an operator: the innermost operator is evaluated, and then *generic-operator* is applied to the result.

—$S$ is a list of substructures: the evaluation results in a new structure, obtained by recursively applying *generic-operator* to each element of $S$.

The only restriction to be imposed on $S$ is that, once each substructure has been evaluated, all the resulting polyhedra must be embedded in the same space. Rules for the evaluation of *set-operator* are given in Section 5.

The optional *indices* specifier is a sequence of integers referring to the elements in the structure. If no indices are supplied, then the operator is applied to each element. If some indices are given, the operator is applied to the specified elements, while a default operator is applied to the others. The purpose of the default operator is to maintain consistency of the embedding space dimension of substructures. The default operator may be the operator Identity, that returns the same structure to which it is applied.

Notice that, for sake of simplicity and without loss of generality, an operator applies only to its argument and does not apply to each following substructure, as customary in graphics; e.g., a PHIGS-like structure as $\langle A, T, B, C \rangle$, where the transformation $T$ is applied both to $B$ and to $C$, is written, according to our syntax, as $\langle A, T\langle B, C \rangle\rangle$. A list of the operators to be described in the following is

```
generic-operator::= "Identity"|"Select"|"Translate"|"Rotate"|"Shear"|"Scale"|
"Components"|"Boundary"|"Extrude"|"Screw"|"Embed"|"Grid"|"Map"|"NURBS"|
"Project"|"Move"|"Joint"
set-operator::= "Union"|"Intersect"|"Difference"
```

The operator Identity returns its argument unchanged. The operator Select, with no parameters, returns the elements with specified *indices* in a structure. $S_{i_1,\ldots,i_n}$ will be used as syntactic "sugar" for $\mathsf{Select}^{i_1,\ldots,i_n}S$. The operators Move and Joint are described in the Appendix C.

*Example* 2. If $A \in \mathscr{P}^{2,3}$, B, C $\in \mathscr{P}^{3,3}$, $\boldsymbol{v}_1 \in \Re^4$ and $\boldsymbol{v}_2, \boldsymbol{v}_3 \in \Re^3$, an example of legal structure definition is:

$$
\begin{aligned}
S = \ & \mathsf{Union} \\
& \quad \mathsf{Extrude}^{1,2}(\boldsymbol{v}_1, 1) \\
& \quad\quad \langle A, \mathsf{Translate}(\boldsymbol{v}_2)A, B, \\
& \quad\quad \mathsf{Translate}(\boldsymbol{v}_3)\langle C, \mathsf{Rotate}(1, 2, \alpha)C \rangle\rangle
\end{aligned}
$$

$S$ is the polyhedron resulting from the union of the extrusion of a collection of some affinely transformed polyhedra.

## 4. OPERATORS

In the following subsections a description of the semantics of each operator is provided. Typewriter-like characters are used to denote operators over structures, while upper-case italic or standard mathematical symbols are used for mathematical operators over polyhedra. For the sake of clarity, an informal style is used for the semantics specification.

### 4.1 Affine Transformations

Affine transformations are used to relocate or stretch objects in space. For the examples to be developed consequently it suffices to define translation, elementary rotations, shearing, and scaling. Since the semantics of these operators is well known, only the corresponding syntax is given.

The mathematical operator for Translate, which takes as parameter the vector $\boldsymbol{t} \in \Re^n$, is the usual translation, that we consider as a bijective

mapping $T_t : \mathscr{P}^{d,n} \rightarrow \mathscr{P}^{d,n}$. Similarly, a scaling mapping $S_s : \mathscr{P}^{d,n} \rightarrow \mathscr{P}^{d,n}$ is associated to the operator Scale with parameter $s \in \Re^n$. The operator Rotate, with parameters $i$, $j$, which identify the rotation plane, and rotation angle $\alpha$, corresponds to the elementary rotation $R_{i,j,\alpha}$ in $\Re^n$, where $R_{i,j,\alpha}$ is defined, as in [42], in such a way that only the coordinates $x_i$ and $x_j$ are changed. The transformation matrix for the rotation $R_{i,j,\alpha}$ differs from the identity matrix $n \times n$ only for the elements $r_{ii} = \cos \alpha$, $r_{ji} = \sin \alpha$, $r_{ij} = -\sin \alpha$, $r_{jj} = \cos \alpha$. The operator Shear, with parameters $i$ and shearing vector $h \in \Re^{n-1}$, is associated to the shearing mapping $H_{i,h} : \mathscr{P}^{d,n} \rightarrow \mathscr{P}^{d,n}$, with respect to the coordinate $x_i$. The shearing matrix relative to $H_{i,h}$ differs from the identity matrix only for the $i$th column, which is $[h_1, \ldots, h_{i-1}, 1, h_i, \ldots, h_{n-1}]^T$.

When the *indices* specifier is supplied, the default operator to be applied to the nonspecified elements of the structure is Identity.

### 4.2 Connected Components

The operator Components, when applied to a polyhedron $P \in \mathscr{P}^{d,n}$, gives as output a structure $S$ which represents the set of its $(d-1)$-connected components.

When $\mathscr{H}(P)$ is given, connected components will be extracted by computing a set of equivalence classes of simplices under the (d − 1)-connectivity relationship. In order to determine a connected component it is therefore sufficient, starting from any $d$-simplex $\sigma$, to recursively traverse the representation and collect $d$-simplices by using the adjacency function, while marking each encountered $d$-simplex. When it is not possible to find an unmarked adjacent $d$-simplex, the extraction of the connected component is completed.

Let us denote the input polyhedron as $P \in \mathscr{P}^{d,n}$, and the set of connected components of $P$ as $S$. The algorithm CollectComponents, with input $\mathscr{H}(P)$ and output $S$, calls the recursive algorithm GetAComponent, which receives the $d$-simplex $\sigma \in K^d(P)$ as input and gives as output the $d$-skeleton and the adjacencies of the $(d-1)$-connected component containing $\sigma$. Pseudocodes are given in Table I.

Notice that the connected-component extraction has complexity $O(d|K^d|)$, since the procedure GetAComponent is called $O(|K^d|)$ times. The same kind of recursive traversal of the representation can be used to extract the polyhedron subsets satisfying different properties, e.g., to extract the shells or the hyperfaces of a boundary representation $\mathscr{H}_a(P)$, where hyperfaces are defined as maximal $(d-2)$-connected subsets of simplices belonging to the same affine subspace.

### 4.3 Boundary

The boundary operator is a mapping $\partial : \mathscr{P}^{d,n} \rightarrow \mathscr{P}^{d-1,n}$ from the set of $d$-polyhedra-with-boundary to the set of $(d-1)$-polyhedra-without-boundary. If $P = [\Sigma^d]$, then the polyhedron $\partial P$ is the geometric carrier of a $(d-1)$-complex whose $(d-1)$-simplices are faces of exactly one $d$-simplex in $\Sigma^d$.

The algorithm for computing the boundary representation $\mathscr{H}_a(P)$ when the decompositive representation $\mathscr{H}(P)$ is given is detailed in [24]. A direct

Table I. Pseudocodes for the CollectComponents and GetAComponent Functions

```
algorithm CollectComponents (input: 𝒲(P); output: S);
S := ⟨⟩;
foreach σ ∈ K^d(P) do
  if not Marked(σ) then
    K̃^d := ∅; A := ∅;
    GetAComponent (σ, K̃^d, A);
    S := S ∪ ⟨(K̃^d, A)⟩

algorithm GetAComponent(σ, K̃^d, A);
K̃^d := K̃^d ∪ {σ}; A := A ∪ 𝒜(σ);
Mark(σ);
foreach σ' ∈ 𝒜(σ) do
  if not Marked(σ') then
    GetAComponent(σ', K̃^d, A)
```

representation of boundary faces is explicitly embedded in the decompositive winged scheme: for each $\perp$ value in the adjacency tuple $\mathscr{A}(\sigma)$, the corresponding $(d-1)$-face of $\sigma$ is a boundary face of $P$. This implies that the skeleton $K^{d-1}(\partial P)$ can be evaluated in $O(ds)$ time by scanning $\mathscr{W}(P)$ and examining $(d+1)s$ adjacency values, where $s = |K^d(P)|$.

Some more work is needed to compute adjacencies between $(d-1)$-faces in $\partial P$. One possible method is to compute $\mathscr{W}_\partial(P)$ by a composition operation of the representations $\mathscr{W}(\partial\sigma_p)$, $\mathscr{W}(\partial\sigma_q)$ for each adjacent pair $\sigma_p$, $\sigma_q \in K^d(P)$. The operation, a kind of "sewing," eliminates the common $(d-1)$-face of $\sigma_p$ and $\sigma_q$ and appropriately crosses the adjacencies in $\partial\sigma_p$, $\partial\sigma_q$. Actually, it turns out that it may be necessary during the incremental construction of $\mathscr{W}(\partial P)$ to execute more than one sewing operation at a time. Such a composition can be performed efficiently because both $K^{d-1}(\partial\sigma)$ and the $(d-2)$-adjacency function $\mathscr{A}: K^{d-1}(\partial\sigma) \to \times_d K^{d-1}(\partial\sigma)$ can be computed with closed formulae [24].

Pseudocode for boundary evaluation is given in Table II, where $K^{d-1}$ and $A$ denote a set of faces and of adjacency tuples during the construction of $\mathscr{W}(\partial P)$. The set of $(d-2)$-adjacency tuples in the boundary of $\sigma_p \in K^d(P)$ is denoted as $\mathscr{A}(\partial\sigma_p)$.

The operator Boundary, with no parameters, results in the boundary mapping $\partial : \mathscr{P}^{d,n} \to \mathscr{P}^{d-1,n}$. The default operator for Boundary, when applied to a subset of elements in a structure, is Identity.

### 4.4 Extrusion

Gaspar Monge used the sweeping operation in the 18th Century, as a method for generating curves and surfaces by moving a point or a curve, respectively. When sweeping is applied to space curves or surfaces, it produces space surfaces or solids, respectively. Sweeping, extrude, and revolve operations are largely used in CAD systems (see, e.g., Pegna [50] and Weld [64]). In this section we analyze the translational extrusion of a polyhedron, which can be considered a basic operation to generate higher-dimensional polyhedra.

Table II. Pseudocode for the Boundary Function

```
algorithm Boundary(input: 𝒦_n(P); output: 𝒦_n(P));
K^{d-1} := ∅; A := ∅;
foreach σ_p ∈ K^d(P) do
  if not Marked(σ_p) then
    A := A ∪ 𝒜(∂σ_p);
    Mark(σ_p);
  foreach 𝒜_h(σ_p) ∈ 𝒜(σ_p) do
    if not MarkedAdjacency(𝒜_h(σ_p)) then
      case 𝒜_h(σ_p) of
        ⊥ :
          K^{d-1} := K^{d-1} ∪ {σ_{p,h}};
        σ_q  where 𝒜_k(σ_q) = σ_p:
          if not Marked(σ_q) then
            A := A ∪ 𝒜(∂σ_q);
            Mark(σ_q);
            Sew(A, σ_{p,h}, σ_{q,k});
            MarkAdjacency(𝒜_k(σ_q));
if K^{d-1} = ∅ then 𝒦_n(P) := (∅, ∅)
  else 𝒦_n(P) := (K^{d-1}, A)
```

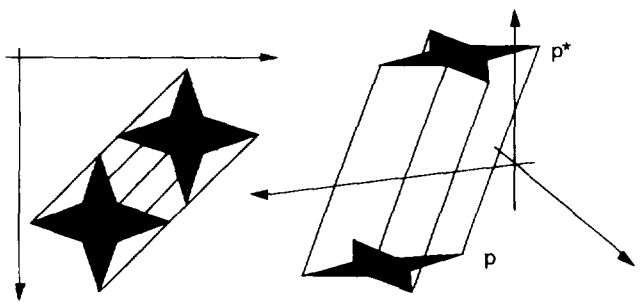



Fig. 5. Sweeping versus extrusion.

Sweeping generates the subset of the embedding $n$-space spanned by a moving $d$-dimensional object. Conversely, extrusion maps a $d$-dimensional object, embedded in $\Re^n$, into a $(d + 1)$-dimensional object in $\Re^{n+1}$, and can be used to describe pointsets that are the product of a polyhedron times an interval. The volume swept by a polyhedron coincides with the projection of its extrusion from $\Re^{n+1}$ back onto $\Re^n$.

The set obtained by extruding a regular polyhedron is a regular polyhedron. Also, boundary points are mapped into boundary points of the resulting polyhedron. This is not generally true in sweeping, where the object is swept within an embedding space of the same dimension (see Figure 5).

The single-step straight extrusion $E_1(P)$ is defined as a mapping $E_1 : \mathcal{P}^{d,n} \to \mathcal{P}^{d+1,n+1}$ such that $E_1(P) = P \times I$, where $I$ is the unit interval $[0, 1] \subset \Re$ and $K^0(E_1(P)) = K^0(P) \times \{0, 1\}$.

*Extrusion of a simplex.* In the following we show a simple combinatorial rule to generate a simplicial chain whose geometric carrier is the set $\sigma \times I$

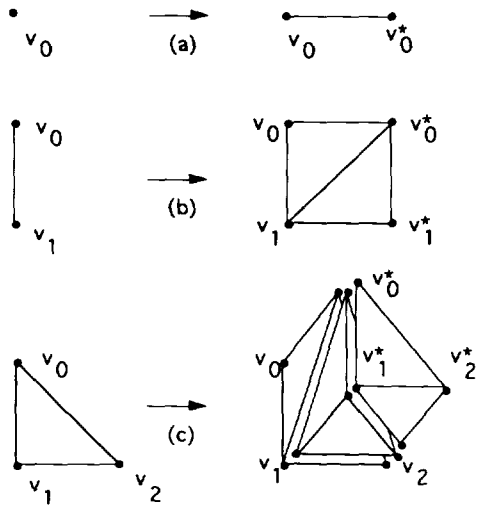



Fig. 6. Extrusion of (a) a point; (b) a straight line segment; (c) a triangle.

resulting from the extrusion of the $d$-simplex $\sigma \subset \Re^n$. The generating rule is introduced by examples. Consider the extrusion of a simple simplex: a 0-simplex (point) generates a chain with one 1-simplex; a 1-simplex (straight line segment) gives a chain with two 2-simplices; a 2-simplex (triangle) produces a chain (wedge) with three 3-simplices; and so on (see Figure 6). A generalization of the above examples suggests a rule for computing a simplicial chain which triangulates the convex set $\sigma \times I$ generated by an oriented $d$-simplex $\sigma$. In particular, the skeleton $K^{d+1}(\sigma \times I)$ contains $d + 1$ coherently oriented $(d + 1)$-simplices $\tau_i$, which are generated by the combinatorial formula:

$$K^{d+1}(\sigma \times I) = \left\{\tau_i : \tau_i = (-1)^{id}\langle v_i, \ldots, v_d, v_0^*, \ldots, v_i^*\rangle, 0 \le i \le d\right\} \quad (2)$$

where $v_i \in K^0(\sigma \times \{0\})$, $v_i^* \in K^0(\sigma \times \{1\})$. A similar triangulation rule is given by Whitney [65, p. 365].

The $i$ exponent in the factors $(-1)^{id}$ in Equation (2) is needed because otherwise the rule would generate a set of simplices having, alternatively, opposite orientations; the $d$ exponent, due to the even or odd simplex order, takes into account the number of exchanges between the simplices $\tau_i$ and $\tau_{i+1}$.

Any pair of simplices $\tau_i$, $\tau_{i+1} \in K^{d+1}(\sigma \times I)$ is $d$-adjacent, as $\tau_i$ and $\tau_{i+1}$ have a common $d$-face. This common face is constituted in $\tau_i$ by the last $d + 1$ vertices and in $\tau_{i+1}$ by the first $d + 1$ vertices. No other pairs of simplices in $K^{d+1}(\sigma \times I)$ have $d + 1$ common vertices. It follows that the complex $\Sigma^{d+1}$ associated to $\sigma \times I$ by (2) is a linear $d$-chain.

The 0-skeleton and the $(d + 1)$-skeleton of the chain $\Sigma^{d+1}$ have $2(d + 1)$ and $(d + 1)$ elements, respectively. In fact, the 0-skeleton contains only $v_0$, $v_1, \ldots, v_d$ and $v_0^*$, $v_1^*, \ldots, v_d^*$. The size of $(d + 1)$-skeleton is directly obtained from (2).

Table III. Pseudocode for the Extrusion Function

```
algorithm Extrusion(input: 𝒦(P1); output: 𝒦(P2));
{skeletons construction}
K^0(P2) := K^0(P1) × {0, 1};
K^{d+1}(P2) := ∪_{σp ∈ K^d(P1)} K^{d+1}(σp × I);
foreach σp ∈ K^d(P1) do
  {adjacencies internal to the chain}
  foreach i ∈ [p(d + 1), ..., p(d + 1) + d − 1] do
    𝒜0(τi) := τi+1;
  foreach i ∈ [p(d + 1) + 1, ..., p(d + 1) + d] do
    𝒜d+1(τi) := τi−1;
  {top and bottom face of the chain}
  𝒜d+1(τp(d+1)) := 𝒜0(τp(d+1)+d) := ⊥ ;
{adjacencies between different chains}
foreach (σp, σq) ∈ K^d(P1) × K^d(P1) where 𝒜h(σp) = σq, 𝒜k(σq) = σp do
  𝒜j1(τi1) := τr1,  (i1, j1) ∈ f(p, h), (r1, ·) ∈ f(q, k), 1 ≤ l ≤ d
  𝒜jl(τil) := τrl,  (il, jl) ∈ f(q, k), (rl, ·) ∈ f(p, h), 1 ≤ l ≤ d
{adjacencies for boundary faces}
foreach σp ∈ K^d(P1) where 𝒜h(σp) = ⊥ do
  𝒜jl(τil) := ⊥ ,  (il, jl) ∈ f(p, h), 1 ≤ l ≤ d
```

*Extrusion of a polyhedron.* The winged representation of $P_2 = E_1(P_1)$ can be computed as discussed in [24]: first, each simplex $\sigma \in K^d(P_1)$ is independently extruded, and this directly generates the skeleton $K^{d+1}(P_2)$; then, adjacencies between simplices in $K^{d+1}(P_2)$ are computed by using closed formulae. The algorithm is such that no "a posteriori" triangulation of the extruded object is required.

Pseudocode for the single-step straight extrusion $E_1$ is given in Table III. Let $P_1 \in \mathscr{P}^{d,n}$ be given, with $K^d(P_1) = \{\sigma_0, \ldots, \sigma_{s-1}\}$. Suppose $P_2 = E_1(P_1)$, $K^{d+1}(P_2) = \{\tau_0, \ldots, \tau_{s(d+1)-1}\}$, and the set of simplices generated by extruding $\sigma_p \in K^d(P_1)$—a simplicial chain—be denoted by $K^{d+1}(\sigma_p \times I) = \{\tau_{p(d+1)}, \ldots, \tau_{p(d+1)+d}\}$. Finally, $\mathscr{A}_k(\sigma_q)$ denotes the $d$-simplex $(d-1)$-adjacent to $\sigma_q$ along its $k$th face. The computation of the adjacency-generating function $f(\cdot, \cdot)$, defined in [24], is $O(d)$, and is repeated $O(ds)$ times; therefore the algorithm is $O(d^2 s)$, where $s = |K^d(P_1)|$. Since this is the size of the output, the algorithm is also $\Theta(d^2 s)$.

*More general extrusion operators.* The *straight extrusion*, with number of steps $h$, is defined as a mapping $E_h : \mathscr{P}^{d,n} \to \mathscr{P}^{d+1,n+1}$ such that

$$E_h(P) = P \times I$$

where $I$ is the unit interval $[0, 1] \subset \mathbb{R}$ and where the set of vertices of $E_h(P)$ is

$$K^0(E_h(P)) = K^0(P) \times \left\{0, \frac{1}{h}, \frac{2}{h}, \ldots, \frac{h-1}{h}, 1\right\}.$$

Two more general operators can be easily defined starting from the straight extrusion. The *linear extrusion* $LE_{v,h}(P)$, with extrusion vector $v \in \mathbb{R}^{n+1}$,

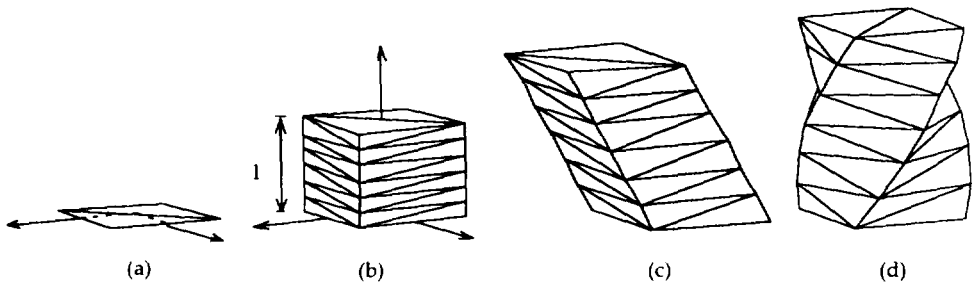



Fig. 7. (a) the square $P$ in $\mathfrak{R}^2$ to be extruded; (b) straight extrusion $E_6(P)$; (c) linear extrusion $LE_{[3/2,3/2,3/2],6}(P)$; (d) screw extrusion $SE_{\pi/2,1,2,6}(P)$. Hidden faces have been removed.

where $v_{n+1} \neq 0$, and number of steps $h$, of the polyhedron $P \in \mathscr{P}^{d,n}$, is defined as a scaling and shearing of the extruded object:

$$LE_{v,h}(P) = S_{[1,\ldots,1,v_{n+1}]} \circ H_{n+1,[v_1,\ldots,v_n]} \circ E_h(P).$$

Notice that $H_{n+1}$ does not affect the coordinate $x_{n+1}$.

The *screw extrusion* $SE_{\theta,i,j,h}(P)$, with rotation angle $\theta$, rotation plane $x_i$, $x_j$, and number of steps $h$, of the polyhedron $P \in \mathscr{P}^{d,n}$, is defined as

$$SE_{\theta,i,j,h}(P) = Z_{i,j} \circ S_{[1,\ldots,1,\theta]} \circ E_h(P)$$

where $Z_{i,j}(Q)$ is a homeomorphic mapping which applies a parametric rotation to the vertices of $Q$, while the convex coordinates of other points in $Q$ within the simplex they belong to do not change. In particular, if $q = (q_1, \ldots, q_{n+1}) \in K^0(Q)$, then $Z_{i,j}q = R_{i,j,q_{n+1}}q$. The technique for computing $E_1(P)$ is easily extended to compute the winged representation of $E_h(P)$, so that representations of the linear or screw extrusion of a $d$-polyhedron $P \in \mathscr{P}^{d,n}$ are both computed in $O(hd^2s)$, where $s = |K^d(P)|$.

*Language operators.* The operator Extrude, with parameters $v \in \mathfrak{R}^{n+1}$ and $h$, results in a linear extrusion $LE_{v,h} : \mathscr{P}^{d,n} \to \mathscr{P}^{d+1,n+1}$. The parameters for the operator Screw are $\theta$, $i$, $j$, $h$. The operator corresponds to a screw extrusion $SE_{\theta,i,j,h} : \mathscr{P}^{d,n} \to \mathscr{P}^{d+1,n+1}$.

*Example* 3. The polyhedra shown in Figures 7b, 7c, and 7d have been obtained, respectively, as

Extrude([0, 0, 1], 6)$P$
Extrude([3/2, 3/2, 3/2], 6)$P$
Screw($\pi$/2, 1, 2, 6)$P$

where $P$ is the 2-polyhedron in $\mathfrak{R}^2$ shown in Figure 7a.

The default operator for both Extrude and Screw is Embed. The operator Embed, with parameter $m$, when applied to $P \in \mathscr{P}^{d,n}$ yields

$$P \times \overbrace{\{0\} \times \cdots \times \{0\}}^{m \text{ times}} \in \mathscr{P}^{d,n+m}.$$

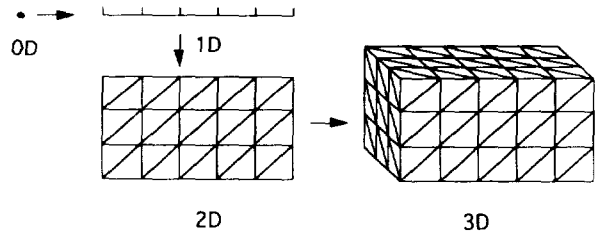



Fig. 8. Grid generation by extrusion.

For example, if $P$ is a 3D cube in $\Re^3$, then Embed(2)$P$ yields $P \times \{0\} \times \{0\}$, and the result is a 3D cube in $\Re^5$, lying in the subspace $x_4$, $x_5 = 0$.

### 4.5 Grid

A grid can be generated by a succession of straight extrusions (see Figure 8). Such an operation can be generalized in order to build a grid over a simplicial complex. This can be useful, e.g., to build a time grid for a dynamic phenomenon over a manifold for which a simplicial approximation (or triangulation) already exists. The grid generator operator, with steps $h_1, \ldots, h_m$, is therefore defined as a mapping $G^m_{h_1,\ldots,h_m} : \mathscr{P}^{d,n} \to \mathscr{P}^{d+m,n+m}$ such that

$$G^m_{h_1,\ldots,h_m}(P) = E_{h_m} \circ \cdots \circ E_{h_2} \circ E_{h_1}(P)$$

where $E_{h_i}$ is the straight extrusion previously defined.

*Example* 4. Let $C^m$ be the $m$-dimensional unit cube, and let $o \in \mathscr{P}^{0,0}$ be the 0-polyhedron consisting of a single point. It is easy to see that the grid $G^m_{1,\ldots,1}(o)$, denoted as $G^m$, coincides with $C^m$, whereas $G^m_{h_1,\ldots,h_m}(o)$, denoted as $G^m_{h_1,\ldots,h_m}$, differs from $C^m$ only because its set of vertices is a superset of $K^0(C^m)$, e.g., $K^0(G^3_{4,4,4}) \supset K^0(C^3)$, since it contains vertices which lie on the edges, faces, and also in the interior of the unit cube. Notice that

$$\left|K^0\left(G^m_{h_1,\ldots,h_m}(o)\right)\right| = (h_1 + 1)(h_2 + 1)\cdots(h_m + 1).$$

The operator Grid, with parameters $h_1, \ldots, h_m$, results in the generation of the grid $G^m_{h_1,\ldots,h_m}$. The default operator for Grid($h_1, \ldots, h_m$) is Embed($m$).

In addition, note that the structure Grid($h_1, h_2, \ldots, h_m$) $o$ corresponds to a uniform partition of the $m$-dimensional unit cube. Uniform partitions of different size and position are obtained by application of an affine transformation. Nonuniform grids can be generated by using the "product" operator described in [46].

### 4.6 Maps

Manifold is, intuitively, the extension of the concept of curve and surface obtained when $m$ independent parameters vary. A differentiable manifold $M$ is a topological space such that: (a) it can be covered by a family of open sets homeomorphic to the interior of the Euclidean hypersphere; (b) when a point

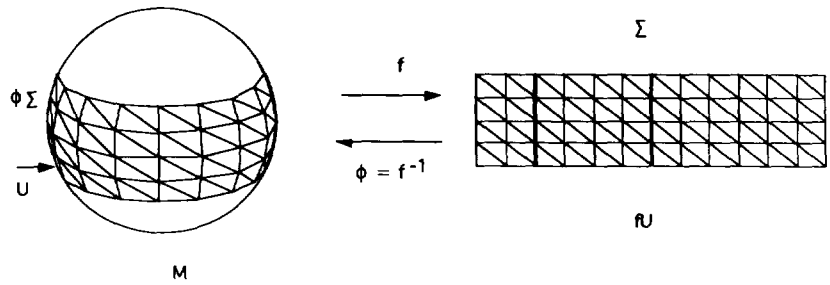



Fig. 9. The concept of simplicial map.

belongs to two such open sets, there exists a bijective smooth mapping between the two corresponding systems of coordinates. Each pair $(U, f)$, where $f: U \rightarrow \Re^m$ and $U \subseteq M$, is called a *chart* of the manifold. The set of charts is called *atlas*.

Most part of engineering computations consists in solving field equations over some differentiable manifold, using some suitable discretization of charts. In this subsection we show that when a chart $(U, f)$ is given, it is very easy to generate a simplicial approximation of $U$ starting from a simplicial decomposition or approximation of $fU$.

Let a pair $(U, f)$ be given; if $fU = [\Sigma]$, where $\Sigma$ is a simplicial complex, then we call *simplicial approximation* of $U$ the set $[\phi\Sigma]$, where $\phi = f^{-1}$ is applied only to the vertices of $\Sigma$, while other points maintain their convex coordinates w.r.t. the simplex to which they belong. This set is easily computed from $\Sigma$ by substituting the set of vertices $K^0(\Sigma)$ with its image $fK^0(\Sigma)$. We call *simplicial map* a pair $(\Sigma, \phi)$ (see Figure 9). Notice that a solid model for the approximation of $U$ is known when the topology of $\Sigma$ is known. This is the case when a winged representation of $\Sigma$ is given. $\Sigma$ is called *grid* when $fU$ is a rectangular domain, i.e., a product of intervals $I_i \subset \Re$ and $K^0(\Sigma) = V_1 \times \cdots \times V_n$, where each $V_i \subset I_i$ is a discrete subset of coordinates.

In order to specify a simplicial map we need to define both a domain discretization (often a grid) and a coordinate transformation $f: U \rightarrow \Re^m$, with $U \subseteq \Re^n$. For this purpose it is necessary to have an algebraic subsystem for the symbolic manipulation of vector-valued real functions $x_h = \phi_h(u_1, u_2, \ldots, u_m)$ of real variables $u_i (1 \leq h \leq n)$. Such a subsystem is being developed within $Simple_X^n$. It allows for: the input of function expressions in standard mathematical format; the automatic generation of partial derivatives; some matrix calculus, with both symbolic and numeric matrix elements; the interfacing with external computer algebra systems (like *Mathematica*).

The operator Map takes as parameters the functions $\boldsymbol{\phi} = \phi_1, \ldots, \phi_n$ which define the simplicial map. As a convention, letters $u_1, \ldots, u_m$ are used to name coordinates in the function domain $fU \subseteq \Re^m$. The syntax for the

specification of a map from $\mathscr{P}^{d,m}$ to $\mathscr{P}^{d,n}$, $d \le m \le n$, is therefore the following:

$$\mathsf{Map}(\phi_1(u_1, \ldots, u_m), \ldots, \phi_n(u_1, \ldots, u_m))$$

using standard mathematical format for the functions specification. The default operator for Map is Embed(n-m). Some examples of the use of the Map operator are given in Appendix B.

### 4.7 NURBS

Non-Uniform Rational B-spline (NURBS) curves and surfaces have assumed in recent years a central role in CAD systems and in standard graphics. In fact, they give both a representation with local support of free-form geometries and an exact representation of conic sections and quadrics. In addition, they exhibit nice geometrical properties, as the global and local containment in the convex hulls of control points, local continuity control, etc.

An easy generalization of NURBS curves and surfaces is the $(m, n)$-NURBS, a mapping from $\mathscr{P}^m$ to $\Re^n$ defined by $n$ rational functions depending on $m$ variables. Each of the $n$ rational functions is the parametric representation of one of the $n$ coordinates of a point in the NURBS range set. The $(m, n)$-NURBS mapping is specified by the usual set of parameters (orders, knot vectors, and control grid), whose meaning is recalled in the following paragraph.

The syntax for the NURBS mapping is NURBS($L, K, C$) where $L$ is a list of $m$ integers, specifying the orders $k_i$ (order = degree + 1) of the $m$ B-spline basis of polynomials. $K$ is the list of $m$ knot vectors (nondecreasing sequences of $l_i$ reals, $1 \le i \le m$). $C$ is an $m$-dimensional array of points in $\mathscr{P}^{n+1}$ used as shape approximation, where the $(n + 1)$-th coordinate is the weight of the control point, which acts as an "attractor" in the shape approximation. The B-spline basis corresponding to the independent variable $u_i$ contains $l_i - k_i$ polynomials, so that the size of the array $C$ of control points is $(l_1 - k_1)(l_2 - k_2) \cdots (l_m - k_m)$. For more details about NURBS curves and surfaces and their $(m, n)$-generalization, the interested reader is referred to [56] and [49].

A solid model of a polyhedral approximation of a NURBS range set is simply obtained by evaluating the expression

$$\mathsf{NURBS}(L, K, C)\ \mathsf{Grid}(h_1, h_2, \ldots, h_m)\, o$$

whose meaning is the following: Grid($h_1, h_2, \ldots, h_m$) $o$ is an $m$-dimensional grid in $m$-dimensional parameter space, with $h_i$ steps in the direction of the $i$th parameter. The vertices of the grid are substituted as parametric values in the $n$ NURBS scalar functions, in order to exactly generate a point in $n$-dimensional range space, while other grid points are affinely mapped in $\Re^n$ according to their convex coordinates $\lambda_j$ $(0 \le j \le d)$. More formally, if $\sigma$ is a

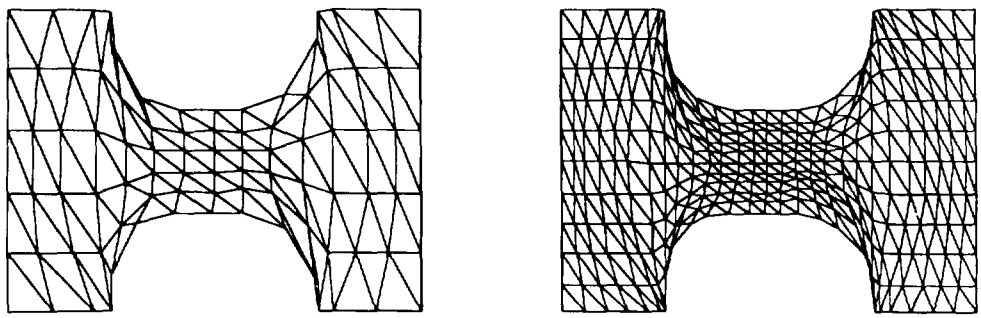


Fig. 10. Two polyhedra, $M_1$, $M_2$, approximating a planar NURBS.

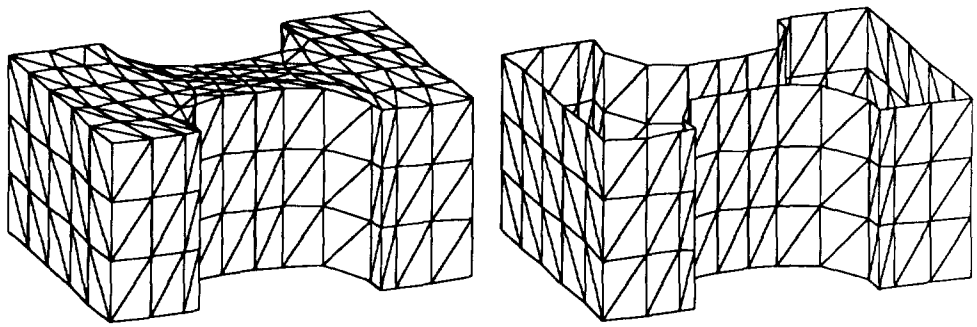


Fig. 11. The results of the expressions Boundary Grid(3) $M_1$ and Grid(3) Boundary $M_1$, where $M_1$ is the polyhedron on the left in Figure 10.

$d$-simplex:

$$\text{if } p \in \sigma \subset \Re^m, \quad \text{and } p = \sum_{v_j \in K^0(\sigma)} \lambda_j v_j,$$

$$\text{then NURBS}()p = \sum_{v_j \in K^0(\sigma)} \lambda_j \, \text{NURBS}()v_j. \tag{3}$$

It is assumed that in the NURBS mapping the topology is conserved. In other words, the polyhedron NURBS() Grid() $o$ has the same topology as Grid() $o$. Notice, that in order to ensure in any case the validity [54] of the resulting polyhedron, a check for possible self-intersections should be performed, with a cost of $O(s^2)$, where $s = m!h_1 \cdots h_m$ is the number of simplices in Grid() $o$.

*Example* 5. Two 2-polyhedra (see Figure 10) are generated by using a (2, 2)-NURBS mapping. They differ only for a refinement of the argument grid. The polyhedron on the left is generated as $M_1$ = NURBS($L, C, K$) Grid(14, 5) $o$, while the one on the right is generated as $M_2$ = NURBS($L, C, K$) Grid(28, 10) $o$. The polyhedra in Figure 11 are obtained by combining the Grid and Boundary operators: On the left, the result of the expression Boundary

Grid(3) $M_1$, where $M_1$ is the 2-polyhedron previously defined; on the right, the result of the expression Grid(3) Boundary $M_1$.

At this point, we would like to stress that in our setting we see an $(m, n)$-NURBS as a mapping from $\mathscr{P}^{d,m}$ to $\mathscr{P}^{d,n}$, $(d \leq m \leq n)$. As a consequence, the operator NURBS is used not only to generate but also to transform polyhedra, and therefore the application of NURBS to a structure makes perfect sense. Notice that the polyhedral argument of a NURBS application must be considered as embedded in parametric space. This setting is mainly useful in three cases: (a) to map a "nonsolid" subset $(d < m)$ of the NURBS domain, e.g., a NURBS curve defined on a NURBS surface; (b) to evaluate a trimmed NURBS, e.g., a trimmed surface in $\Re^3$, given a polyhedral approximation of its 2D parametric domain set; (c) to build a multigrid approximation, where grids of different resolution are defined on (even overlapping) subsets of the map domain.

### 4.8 Project

The projection $\Pi_i$ of the polyhedron $P \in \mathscr{P}^{d,n}$ is defined as the set:

$$\Pi_i(P) = \{(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n), p \in P\} \subset \Re^{n-1}.$$

Notice that $\Pi_i(P)$ is a polyhedron, but in the general case a simplicial decomposition of $\Pi_i(P)$ is not easily obtained from a simplicial decomposition of $P$. However, the projection of a single $d$-simplex $\sigma \subset \Re^n$ has particular properties. Since $\sigma$ is a $d$-simplex, there is at least one $(d-1)$-face of $\sigma$, say $\tau$, which projects in a $(d-1)$-simplex $\Pi_i(\tau)$ in $\Re^{n-1}$. Consider, with respect to $\Pi_i(\tau)$, the projection of the only vertex $v$ of $\sigma$ which is not a vertex of $\tau$. Two cases are possible, namely either $\Pi_i(v) \in \Pi_i(\tau)$ or $\Pi_i(v) \notin \Pi_i(\tau)$. In the first case, the projection of $\sigma$ onto $\Re^{n-1}$ coincides with the $(d-1)$-simplex $\Pi_i(\tau)$. Otherwise, the projection of $\sigma$ onto $\Re^{n-1}$ is the convex set $\Pi_i(\tau)\Pi_i(v)$ resulting from the join of the $(d-1)$-simplex $\Pi_i(\tau)$ and of the point $\Pi_i(v)$. This is a convex set with $(d+1)$ vertices in $\Re^{n-1}$, which can always be decomposed in at most $\lfloor (n+1)/2 \rfloor$ simplices.

When projecting a complex, the problem is complicated by the overlap of the projected simplices. Nevertheless, as in the case of a single simplex, the study of topological properties of the projections of subsets of boundary faces may be useful to devise an efficient solution to the problem.

The operator Project, with parameters $i_1, \ldots, i_m$, results in the application of $\Pi_{i_1} \circ \cdots \circ \Pi_{i_m}$. Use of the optional indices specifier is not allowed in this case. Presently this operator is not supported within $Simple_X^n$ in the general case.

## 5. SET OPERATIONS

The evaluation of a *set-operator* is accomplished differently than the evaluation of a *generic-operator* (see Section 4). As the calculation of the result of a single Boolean operation can be very time consuming, it is convenient to attempt a simplification step over a whole Boolean expression, thereby

avoiding the computation of unnecessary subexpressions. In the evaluation phase, when a *set-operator* occurs to be applied to a set of substructures, we postpone its evaluation until all the *generic-operators* contained in these substructures have been evaluated. Hence, if a *generic-operator* has as argument a substructure containing *set-operators*, then the whole process has to be first applied to this substructure. In this way we obtain an expression formed by *set-operators* and polyhedra only. Then we evaluate the whole Boolean expression, using tests to prune unnecessary parts.

Let $S$ be a structure. Then, the rules for the first phase of the evaluation of a *set-operator* over $S$ can be formally stated as follows:

—$S$ is a polyhedron or a list of polyhedra: $S$ is returned unchanged.

—$S$ is a list of substructures, and at least one of them is not a polyhedron: the evaluation results in a new structure, obtained by evaluating all the *generic-operators* contained in the substructures.

—$S$ is a structure beginning with a *generic-operator*: evaluate the innermost operator, and then apply *set-operator* to the result.

The operators Union, Intersection, and Difference correspond to the regularized set operators $\cup^*$, $\cap^*$; and $-^*$, defined as the closure of the interior of the result of the theoretic set operation. It is assumed that all operands are in $\mathscr{P}^{n,n}$.

*Example* 6. Evaluation of the expression

```
Difference
  ⟨Union⟨A, Boundary⟨B, C⟩⟩,
  D,
  Intersect⟨E, F, G⟩⟩
```

where $A, \ldots, G$ are polyhedra in $\Re^n$, yields the Boolean expression

$$(A \cup^* (\partial B \cup^* \partial C)) -^* (D -^* (E \cap^* (F \cap^* G))).$$

The Boundary generic-operator is first evaluated ($\partial B$ and $\partial C$ are the winged representations of the boundaries of polyhedra $B$ and $C$). The final expression contains polyhedra and set-operators only.

### 5.1 Evaluation of a Boolean Expression

A Boolean expression obtained with the transformations previously described is computed executing the following steps:

*Normalization.* The expression is converted into a normal form, called *P-form*, by means of symbolic manipulation techniques. The *P-form* is a union of quasidisjointed *P-constituents*, each *P-constituent* being the intersection of positive and negative polyhedra in the expression. A negative polyhedron represents a "hole" in the space, that is, the difference between the universe (the affine embedding subspace) and its interior. However, since negative polyhedra only appear in the intersection with positive polyhedra, the complement operation never needs to be executed per se, but can always be translated into a difference: an expression as $AB\bar{C}$ is always computed as

$(A \cap^* B) -^* C$. Each *P-constituent* represents a portion of space that can be not convex and not connected. During the normalization process, a pruning of the expression is executed, using a bounding-box intersection test.

*Selection.* Each polyhedron in the *P-form* is expanded in its set of simplices; the result is a new normal form, called *s-form*, that is a union of quasidisjointed *s-constituents*, each *s-constituent* being the intersection of positive and negative simplices. During the selection process, a pruning of the expression is executed, using a simplex vs. convex-hull-of-polyhedra intersection test (see Section 5.3). Although the number of *s-constituents* is exponential in the worst case, it may be expected that pruning may strongly reduce this number in practical cases (see [66]).

*Intersection.* The positive simplices in each *s-constituent* are intersected, and the vertices of the resulting convex are computed.

*Triangulation.* The resulting convex sets are triangulated and their $\mathscr{H}$ representations are computed.

*Repeated-difference.* One of the negative simplices of the *s-constituent* is subtracted from the triangulated convex, by using the TriangulateJoin algorithm described in the following section. The process is iterated for each negative simplex. At the end, we have a triangulation of the *s-constituent*. Adjacencies between *s-forms* have to be taken into account in order to obtain a valid triangulation of the whole polyhedron resulting from the Boolean expression.

*Simplification.* At this point a further reduction step of the decomposition (not yet implemented in our system) could be executed to reduce the fragmentation of the resulting representation. This (challenging!) phase of the computation of a Boolean expression is not discussed in the following.

*Example* 7. In Figure 12 an example of the evaluation process is shown. The expression to be evaluated is $A \cup^* B$. The Normalization step yields the *P-form* $(A\bar{B}, AB, \bar{A}B)$. The result of the selection step is the *s-form* $(a_1\bar{b}_1\bar{b}_2, a_2\bar{b}_1\bar{b}_2, a_1b_1, a_1b_2, a_2b_1, a_2b_2, b_1\bar{a}_1\bar{a}_2, b_2\bar{a}_1\bar{a}_2)$. At the end of the Triangulation and Repeated-difference steps the $\mathscr{H}$ representation of the resulting polyhedron is obtained.

Details of the technique can be found in [23]. Alternative approaches to Boolean operations in dimension $n$, discussed in the Introduction, can be found in [9], [52], [57], and [58]. In the following we briefly discuss the guidelines of the method we adopted to compute an *s-constituent*.

### 5.2 Computation of an *s-constituent*

As we stated above, a Boolean expression can be transformed in an *s-form*, i.e., the union of *s-constituents*. The transformation rules used in the Normalization and Selection processes guarantee that the *s-constituents* are quasidisjointed (their regularized intersection is empty). Therefore, it is possible to compute each *s-constituent* separately, and then glue them together. The

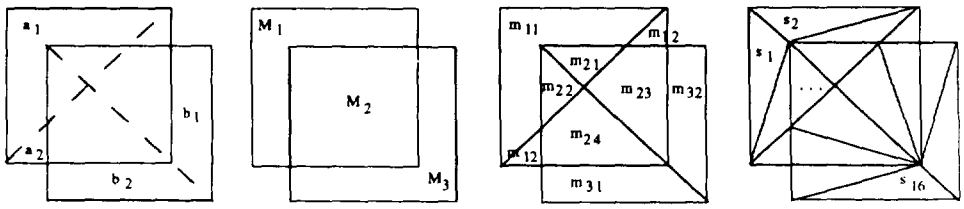



Fig. 12. Evaluation of the Boolean expression $A \cup^* B$. (a) The polyhedra $A$ e $B$, with the underlying triangulation; (b) Normalization: $M_1 = A\bar{B}$, $M_2 = AB$, $M_3 = \bar{A}B$; (c) Selection: $m_{11} = a_1\bar{b}_1\bar{b}_2$, etc.; (d) The nonsimplified triangulation of the result.

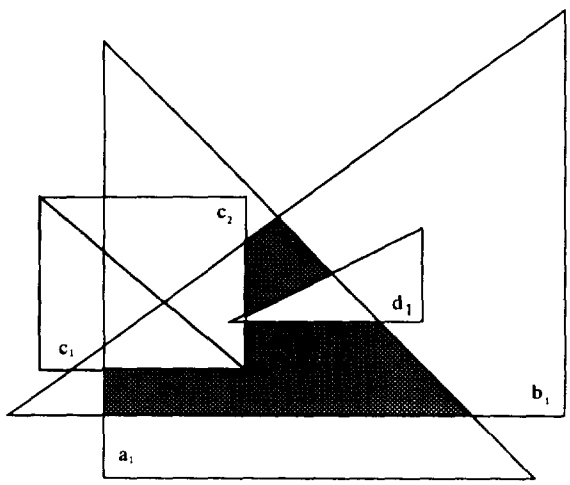



Fig. 13. An example (in 2D) of the *s-constituent* $a_1b_1\bar{c}_1\bar{c}_2\bar{d}_1$.

only interaction constraint is between adjacent cells, as the triangulation induced on the common boundary regions must be the same.

An *s-constituent* is a portion of space (eventually not convex and not connected) given by the intersection of positive and negative (complemented) simplices, belonging to different polyhedra. An example of *s-constituent* (see Figure 13) is:

$$a_1b_1\bar{c}_1\bar{c}_2\bar{d}_1,$$

where $a_1$, $b_1$, $\bar{c}_1$, $\bar{c}_2$, and $\bar{d}_1$ are simplices belonging to the polyhedra $A$, $B$, $C$, and $D$, respectively, and where simplices $\bar{c}_1$, $\bar{c}_2$, and $\bar{d}_1$ are complemented.

Given two simplices $\sigma_1$ and $\sigma_2$, belonging to the same simplicial complex, it is, by definition, $\sigma_1 \cap^* \sigma_2 = \varnothing$ and $\sigma_1 \cap^* \bar{\sigma}_2 = \sigma_1$. These simple rules (together with geometric tests) are used in the selection phase to prune and

simplify *s-constituents*. An *s-constituent* can be calculated, i.e., a winged representation can be found for it, with the following rule:

$$a_1 b_1 \bar{c}_1 \bar{c}_2 \bar{d}_1 = ((((a_1 \cap^* b_1) -^* c_1) -^* c_2) -^* d_1)$$

corresponding to the Intersection, Triangulation, and Repeated-difference steps listed before.

Simplices are convex sets; therefore the intersection of a set of (positive) simplices is a convex set and can be seen as the feasible solution set of a linear-programming problem written using convex coordinates as decision variables, where convex expressions of common points are constrained to be equal (see Section 5.3). Hence, the empty-intersection test can be performed by simply putting the LP tableau in canonical form. If the intersection is nonempty, all its vertices can be extracted with a suitable algorithm [18, 19, 38]. The interested reader is referred to the survey discussion of methods to compute all vertices of a convex polyhedral set by Matheiss and Rubin [37]. We are currently using a revised version of the algorithm described in [18]. Once the vertices of the intersection set have been computed, this convex set can be triangulated with the method described by Von Hohenbalken [63]. The algorithm has been modified [23] in order to allow the triangulation of a convex when adjacent convexes have already been triangulated. This is done by using the triangulation induced on the boundary of the convex as a start to triangulate the interior and by inserting an internal vertex when a situation occurs which is unsolvable by using only original vertices.

In the Repeated-difference phase negative simplices are subtracted one at a time. The pseudocode for the function that computes a single difference is given in Table IV. $\Sigma$ is the input complex; $\tau$ is the simplex to be subtracted; $\bar{\Sigma}$ is the output complex; $\sigma$ is a simplex of $\Sigma$; and $v_j$ is the $j$th vertex of $\sigma$. $\Sigma_1$, $\Sigma_2$ are simplicial complexes. At each step of the execution the geometric carriers $[\Sigma_1]$, $[\Sigma_2]$ are convexes. The meaning of the statement

$$\text{Glue}\big(\mathscr{W}(\bar{\Sigma}), \text{Drill}(\mathscr{W}(\Sigma_1), \mathscr{W}(\Sigma_2))\big)$$

is "add to $\bar{\Sigma}$ those simplices of $\Sigma_1$ that do not belong to $\Sigma_2$, appropriately updating adjacencies."

—TriangulateIntersection is a function that returns, given two input simplices, a simplicial complex triangulating their intersection. As anticipated before, care must be taken in order to triangulate the convex coherently with adjacent simplices.

—External returns True if the vertex is external to the simplex, False otherwise.

—TriangulateJoin is a function having as input the winged representation $\mathscr{W}(\Sigma_1)$ of a convex set and a vertex $v$ external to the set. The output is a winged representation of their join set $v[\Sigma_1]$. To compute such a decomposition it is sufficient to evaluate the boundary of the input $d$-complex and, for each $(d-1)$-face in the boundary, to join it with $v$ if $v$ is contained in the external affine subspace of the face, yielding a $d$-simplex of the output

Table IV. Pseudocode for the Difference Function

```
algorithm Difference (input: 𝒲(Σ), τ; output: 𝒲(Σ̃));
𝒲(Σ̃) := (∅, ∅);
foreach σ ∈ K^d(Σ) do
  𝒲(Σ_2) := TriangulateIntersection(σ, τ);
  𝒲(Σ_1) := 𝒲(Σ_2);
  foreach v_j ∈ K^0(σ) do
    if External(v_j, τ) then
      𝒲(Σ_1) := TriangulateJoin(𝒲(Σ_1), v_j);
  𝒲(Σ̃) := Glue(𝒲(Σ̃), Drill(𝒲(Σ_1), 𝒲(Σ_2)))
```

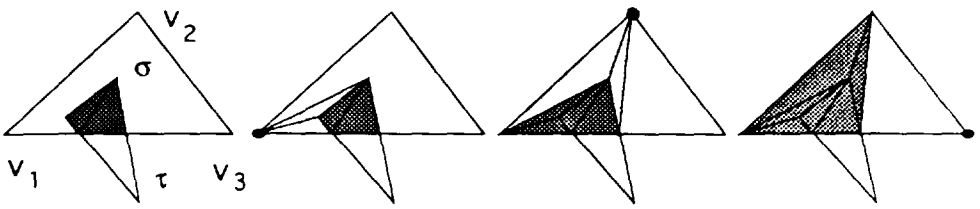



Fig. 14. A graphical description of the iterative use of the TriangulateJoin function.

decomposition. Notice that (i) the join of a vertex with a convex set is again a convex set and that (ii) the output of the function is a simplicial decomposition of a convex set, so that the function can be used iteratively (see Figure 14).

—Drill is a function which returns the winged representation of the difference between (winged representations of) two consistent simplicial complexes $\Sigma_1$ and $\Sigma_2$. By *consistent* we mean that their intersection is triangulated in the same way. The computation is performed: (a) by localizing faces of $\Sigma_1$ which coincide with faces on the boundary of $\Sigma_2$ and setting their adjacencies to $\perp$ ; (b) by discarding simplices in $K^d(\Sigma_1) \cap K^d(\Sigma_2)$.

—Glue is a function which returns the winged representation of the union between (winged representations of) two quasidisjointed complexes $\Sigma_1$ and $\Sigma_2$, where the common boundary is consistently triangulated. The computation is performed by localizing boundary faces of $\Sigma_1$ which coincide with boundary faces of $\Sigma_2$ and sewing their adjacencies. A similar operation is used in the boundary evaluation (see Section 4.3).

The approach to the computation of the intersection of two polyhedra when using decompositive representations can be parallelized more easily than when using boundary representations. The simplest strategy is to put the polyhedron of minimum size (number of simplices) on one processor and to uniformly distribute the other polyhedron on the remaining processors. The intersection problem is reduced in this way to a set of pairwise disjointed-intersection problems of smaller size. The quadratic approach can be strongly

improved by maintaining a recursive partition of the embedding space, e.g., by generalizing to $\Re^n$ the tree-shaped partition techniques known in $\Re^2$ and $\Re^3$ as quadtree and octree, or by using their dimension-independent variant, the bintree [61]. An alternative is to introduce space-cutting trees, originally proposed by Fuchs [26] for hidden-surfaces removal. This approach has been considered by Vaněček [62] for Boolean operation acceleration and entity/boundary classification problems.

### 5.3 Intersection of Convex Sets

A bold notation for points is used in this section and in Section 6.1 in order to ease the understanding of vector and matrix expressions.

Given a finite set of points $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m\} \subset \Re^n$, any point $\boldsymbol{p}$ belonging to *conv* $U$, the convex hull of $U$, can be expressed as:

$$\begin{aligned} & \boldsymbol{p} = \sum_j \lambda_j \boldsymbol{u}_j \\ \text{s.t.} \quad & \sum_j \lambda_j = 1 \\ & \lambda_j \geq 0 \qquad 1 \leq j \leq m. \end{aligned}$$

The scalars $\lambda_j$, called convex coordinates of $\boldsymbol{p}$ with respect to $U$, are unique when the points $\boldsymbol{u}_j$ are affinely independent, i.e., when they are the vertices of a simplex of order $d = m - 1 \leq n$. When $U$ coincides with the 0-skeleton of a polyhedron $P$ the point $\boldsymbol{p}$ spans the convex hull of $P$. Notice the $P$ needs to be neither convex nor connected.

Let $V = [\boldsymbol{u}_1 \ \boldsymbol{u}_2 \cdots \boldsymbol{u}_m]$ be an $n \times m$ array, where the columns are the points of $U$. Convex coordinates are denoted by the column vector $\boldsymbol{\lambda} = [\lambda_1 \ \lambda_2 \cdots \lambda_m]^T$. Using matrix notation we can write;

$$\begin{aligned} & \boldsymbol{p} = V\boldsymbol{\lambda} \\ \text{s.t.} \quad & \mathbf{1}^T\boldsymbol{\lambda} = 1 \\ & \lambda_j \geq 0 \qquad 1 \leq j \leq m. \end{aligned} \tag{4}$$

Consider two sets of points $U_1$ and $U_2$. The intersection of *conv* $U_1$ and *conv* $U_2$ is a convex set. We have $\boldsymbol{p} \in$ *conv* $U_1 \cap$ *conv* $U_2$ if and only if both $\boldsymbol{p} = V_1\boldsymbol{\lambda}_1$ and $\boldsymbol{p} = V_2\boldsymbol{\lambda}_2$ hold, with the usual constraints on convex coordinates, i.e., iff the following system admits solution:

$$\begin{aligned} & V_1\boldsymbol{\lambda}_1 = V_2\boldsymbol{\lambda}_2 \\ & \mathbf{1}^T\boldsymbol{\lambda}_1 = 1 \\ \text{s.t.} \quad & \mathbf{1}^T\boldsymbol{\lambda}_2 = 1 \\ & \lambda_{1,j}, \lambda_{2,k} \geq 0 \qquad 1 \leq j \leq m_1, 1 \leq k \leq m_2. \end{aligned}$$

Hence the set *conv* $U_1 \cap$ *conv* $U_2$ can be characterized as the solution set of

the simultaneous equalities and disequalities

$$\begin{bmatrix} V_1 & -V_2 \\ \mathbf{1}^T & \mathbf{o}^T \\ \mathbf{o}^T & \mathbf{1}^T \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} \mathbf{o} \\ 1 \\ 1 \end{bmatrix} \tag{5}$$

$$\lambda_j \geq 0 \qquad 1 \leq j \leq m_1 + m_2,$$

where $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1^T \ \boldsymbol{\lambda}_2^T]^T$. This yields a set of $n + 2$ equations in the $m_1 + m_2$ unknowns $\boldsymbol{\lambda}$. The equations (5) have the standard form of the constraint set of a linear-programming problem [40], and any linear programming solution technique can efficiently provide an answer for the emptyness test.

The formulation above can be used in executing a convex-polyhedron vs. convex-hull, or a convex-hull vs. convex-hull intersection test, as we need in the normalization and selection phases previously described. This is useful for pruning during the evaluation of the tree representing a Boolean expression. When using the above technique, the sets of points $U_1 = K^0(P_1)$ and $U_2 = K^0(P_2)$ implicitly represent the convex hulls of general nonconvex polyhedra $P_1$ and $P_2$. Notice that there is no need for explicit computation of convex hulls as minimal sets of points: linear-programming techniques by themselves recognize and discard redundant variables.

A particular case arises when the sets *conv* $U_1$ and *conv* $U_2$ are simplices. If their dimensions are $d_1$ and $d_2$, respectively, the set of linear constraints which describes their intersection has $n + 2$ equations in $d_1 + d_2 + 2$ unknowns. Hence, the set-up in (5) is also used for computing the intersection set of two simplices, as required in the intersection phase of Section 5.1, allowing for a further conceptual economy.

## 6. OTHER OPERATORS

The operators described in the previous sections have a common property: they can all be regarded as functions between two spaces of polyhedra. Some other operations are frequently needed in geometric modeling which are not of this kind. An example is the classification of a point with respect to a polyhedron, which is a mapping $\mathscr{P}^{d,n} \times \Re^n \rightarrow \{\text{True, False}\}$. This problem and the integration of polynomials over polyhedra are discussed in the following.

### 6.1 Point-Polyhedron Classification

The problem of point containment in a convex set—as a particular case, in a simplex—is easily cast into a linear-programming formulation. The query point $\boldsymbol{p}$ is contained in the convex hull of a finite set of points $U \subset \Re^n$ iff the feasible solution set associated to the linear constraint set (4) is nonempty.

The containment test is reduced to the emptyness test for a convex set in the space of the variables $\boldsymbol{\lambda}$, which can be done by using any standard linear-programming method, e.g., the artificial-variables technique [40]. This is equivalent to the solution of a particular LP problem, say to one run of the simplex method. Testing point inclusion against a convex polyhedron $P \in \mathscr{P}^{d,n}$ with this method requires just one such test over an LP problem of size

$O(|K^0(P)|)$, which is likely to range from moderate to high. For a general nonconvex polyhedron the same technique is applied to each simplex of $\mathcal{H}(P)$. This requires $|K^d(P)|$ emptyness tests, which implies the solution of $|K^d(P)|$ linear-programming problems of size $O(d+1)$. This approach seems to be more efficient in practical cases than comparing the point to the hyperplanes supporting the faces of each simplex.

With a different approach, we can devise a specialized data structure to answer inclusion queries. For the convex case, one such spatial index has been proposed by Ferrucci and Vaněček [25] which allows point classification in $O(\log^d nd)$ time.

### 6.2 Integration

A solid representation must support algorithms for numeric and/or symbolic integration. A representation based on a simplicial decomposition of the integration domain $P \in \mathcal{P}^{d,n}$ makes the integration easier, due to the domain additivity property of integrals:

$$\int_P f(p)dV = \sum_{\sigma \in K^d} \int_\sigma f(p)dV.$$

Numerical integration over simplices in $n$-space has been discussed by Hammer and Stroud [30], who obtained exact formulae for the quadratic and cubic polynomials over $n$-simplices in the $n$-space. In the related paper [31] Hammer et al. gave quadrature formulae for the $k$th-degree polynomial over the $n$-simplex. Each formulae, by using the $\mathcal{H}_0^1$ representation of 3-polyhedra, are given by Cattani and Paoluzzi [17]. Two exact algorithms and their implementation for the integration of multivariate monomials over a multidimensional polyhedron $P \in \mathcal{P}^{d,n}$ have recently been described by Bernardini [7]:

(1) When using a decompositive representation of the integration domain $P$, for each simplex $\sigma \in K^d(P)$ perform a coordinate transformation, affinely mapping $\sigma$ into the standard simplex $\langle(0,\ldots,0),(1,0,\ldots,0),\ldots,(0,\ldots,0,1)\rangle$, evaluate the integral over the standard simplex, and accumulate the result multiplied by the Jacobian of the transformation, given by the determinant of the matrix of the affine mapping.

(2) When using a boundary representation, compute the integral by repeated integration, as a summation (for each simplex $\tau \in K^{d-1}(\partial P)$) of integrals over the cylinders having $\tau$ and its projection $\Pi(\tau)$ into the plane $x_n = 0$ as bases. Such integrals can be evaluated by recursively computing integrals over the (lower-dimensional) bases of the cylinders, until the intrinsic dimension of the integration domain is one.

The first algorithm requires only the knowledge of the $d$-skeleton of the integration domain $P$, while the efficiency of the second is strongly improved when a winged representation $\mathcal{H}_0^1(P)$ is given. In fact, for each recursion step $s$ $(1 \le s \le d)$ it is necessary to (efficiently) evaluate the boundary of the integration domain and to (efficiently) extract its boundary hyperfaces, i.e., the connected subsets of the boundary which belong to the same hyperplanes.

## 7. CONCLUSIONS

A simple and general representation and a set of tools to work with simplicial decomposition of dimension-independent polyhedra have been described. Both linear polyhedra and simplicial approximations of curved polyhedra can be generated and combined in any dimension with extrusion, simplicial maps, set operations, and so on. Such objects can be collected into structures, which allow us to combine both other structures and primitive objects with affine transformations and higher-level operators. The described approach allows for a unified view and treatment of several geometric problems.

A prototype multidimensional modeler called $Simple_X^n$, which supports both decompositive, boundary, and symbolic representations, is under development [6, 22, 49]. Presently, $Simple_X^n$ allows for: the traversal, with reduction to world coordinates, of structures defined by using local modeling coordinates; the generation of higher-dimensional polyhedra, via linear or screw extrusion; the evaluation of Boolean expressions and/or their reduction to a normal form; the integration of polynomials over polyhedral domains; the definition of simplicial maps and NURBS, which allow us to generate polyhedral approximations of curved manifolds and fields over manifolds.

Whereas simplicial representations seem to be efficient in low dimension (e.g., such representations use an optimum storage when linearly approximating the boundary of 3D curved solids [45]), when dealing with extruded objects the number of simplices grows unfortunately with $O(m!)$, where $m$ is the number of dimensions added by extrusion operations. Another critical point is the high fragmentation of the simplicial representation deriving from set operations. This would require the introduction of a reduction step, currently not supported, after the evaluation of Boolean expressions. Much more can also be done to speed up set operations, e.g., by associating spatial indices [25] to the simplicial complexes to be combined.

The aim of the paper was to show a point of view on representational issues, as we believe that representations based on simplicial complexes may produce the convergence of methods of solid and (curved) geometric modeling and may actually help to reduce the gap between solid modelers and CAE systems, which make large use of cell decompositions of solids. In this framework a major research effort is still needed, as many problems need to find an efficient solution, including the computation of the projection operator, the simplification of fragmented complexes resulting from Boolean operations, and their acceleration. An alternative representation can be set up with cell decompositions where cells are convex or affine sets. With such a representation most of the implemented algorithms (based on LP techniques) can be used, with the advantage of no augmentation in the number of the extruded cells and less fragmentation. Such an approach is under study.

The need for high computing power, the use of decompositive representations, and the central role of LP techniques can take advantage of parallel computing architectures. In particular, we suggest a large-grain MIMD parallelism based on ultracomputers or hypercube machines to exploit both data and task subdivision. The authors believe that multidimensional modeling

implemented on such machines may constitute an advance for both applied geometry and computer graphics.

## APPENDIX A

### $d$-Dimensional Cubes

A $d$-dimensional cube $C^d$ can be recursively defined as the image of a lower-dimensional cube $C^{d-1}$ under an extrusion operation. For the basic case we have that $C^0$ is a single point. $C^1, C^2, C^3, C^4$ are the unit line segment, square, cube, and hypercube, respectively. Using the data definition language described in this paper, the $d$-dimensional unit cube can be defined as

Grid$(1, 1, \ldots, 1)o$

where the Grid operator has $d$ unit arguments, and $o \in \mathscr{P}^{0,0}$. The expression above is equivalent to $G^d_{1,1,\ldots,1}(o) = E_1 \circ \cdots \circ E_1 \circ E_1(o)$.

In the following, both boundary and decompositive winged representations for various order cubes are given, with the aim of presenting several constructively built winged representations. In step-wise generating the representation of the $(d+1)$-th order cube, we need to reindex the vertices generated by extruding the $d$th-order cube. For the basic case $K^0(C^0) = \{v_0\}$, and in general $K^0(C^d) = \{v_0, \ldots, v_{2^d-1}\}$, since a $d$-cube has $2^d$ vertices. Notice that $K^0(E_1(C^d)) = K^0(C^d) \times \{0, 1\}$, so that for every $v_i$ in $K^0(C^d)$ two vertices $v_i \times \{0\}$ and $v_i \times \{1\}$ arise in $K^0(E_1(C^d))$; they are denoted as $v_i$ and $v_i^*$, respectively. Since $E_1(C^d) \equiv C^{d+1}$, we use the reindexing:

$$v_i \in K^0(C^{d+1}) \text{ stands for } \begin{cases} v_i \in K^0\big(E_1(C^d)\big) & 0 \le i \le 2^d - 1 \\ v_{i-2^d}^* \in K^0\big(E_1(C^d)\big) & 2^d \le i \le 2^{d+1} - 1 \end{cases} \tag{6}$$

Therefore we have:

$$\begin{aligned} K^0(C^0) &= \{v_0\} \\ K^0(C^d) &= \{v_0, \ldots, v_{2^d-1}\} \\ K^0\big(E_1(C^d)\big) &= \{v_0, \ldots, v_{2^d-1}; v_0^*, \ldots, v_{2^d-1}^*\} \\ K^0(C^{d+1}) &= \{v_0, \ldots, v_{2^d-1}, v_{2^d}, \ldots, v_{2^{d+1}-1}\} \end{aligned}$$

Finally, remember that: (a) the decompositive representation of an open (i.e., with boundary) polyhedron is characterized by the presence of at least one $\perp$ symbol in the adjacency tuples; (b) the boundary representation of an open polyhedron is a decompositive representation of its closed (i.e., without boundary) boundary.

*Line segment.* The unit line segment $C^1$ is obtained by extruding $C^0 \equiv o = \{v_0\}$. The decompositive representation is:

$$\mathscr{H}(C^1) = \{\sigma_0 = +\langle v_0, v_0^* \rangle \quad \mathscr{A}(\sigma_0) = \langle \perp, \perp \rangle$$

A boundary representation is not given, since $\mathscr{H}_a$ is not defined for a 1-dimensional polyhedron (see Section 2.3).

*Square.* The unit square $C^2$ is obtained by extruding the unit line segment $C^1$, where $v_0^*$ has been reindexed as $v_1$, according to rule (6). Therefore we have, for the decompositive and boundary representations:

$$\mathscr{W}(C^2) = \begin{cases} \sigma_0 = +\langle v_0, v_1, v_0^* \rangle & \mathscr{A}(\sigma_0) = \langle \sigma_1, \perp, \perp \rangle \\ \sigma_1 = -\langle v_1, v_0^*, v_1^* \rangle & \mathscr{A}(\sigma_1) = \langle \perp, \perp, \sigma_0 \rangle \end{cases}$$

The corresponding boundary representation is

$$\mathscr{W}_\partial(C^2) = \begin{cases} \sigma_{0,1} = -\langle v_0, v_0^* \rangle & \mathscr{A}(\sigma_{0,1}) = \langle \sigma_{1,0}, \sigma_{0,2} \rangle \\ \sigma_{0,2} = +\langle v_0, v_1 \rangle & \mathscr{A}(\sigma_{0,2}) = \langle \sigma_{1,1}, \sigma_{0,1} \rangle \\ \sigma_{1,0} = -\langle v_0^*, v_1^* \rangle & \mathscr{A}(\sigma_{1,0}) = \langle \sigma_{1,1}, \sigma_{0,1} \rangle \\ \sigma_{1,1} = +\langle v_1, v_1^* \rangle & \mathscr{A}(\sigma_{1,1}) = \langle \sigma_{1,0}, \sigma_{0,2} \rangle \end{cases}$$

*Cube.* The 3D cube $C^3$ is obtained by extruding the $C^2$ square, where the vertices $v_0^*$ and $v_1^*$ have been reindexed as $v_2$ and $v_3$, respectively. The 3-simplices in $C^3$ are obtained, according to Section 4.4, by separately extruding 2-simplices in $C^2$.

$$\mathscr{W}(C^3) = \begin{cases} \sigma_0 = +\langle v_0, v_1, v_2, v_0^* \rangle & \mathscr{A}(\sigma_0) = \langle \sigma_1, \perp, \perp, \perp \rangle \\ \sigma_1 = +\langle v_1, v_2, v_0^*, v_1^* \rangle & \mathscr{A}(\sigma_1) = \langle \sigma_2, \perp, \sigma_3, \sigma_0 \rangle \\ \sigma_2 = +\langle v_2, v_0^*, v_1^*, v_2^* \rangle & \mathscr{A}(\sigma_2) = \langle \perp, \sigma_4, \perp, \sigma_1 \rangle \\ \sigma_3 = -\langle v_1, v_2, v_3, v_1^* \rangle & \mathscr{A}(\sigma_3) = \langle \sigma_4, \perp, \sigma_1, \perp \rangle \\ \sigma_4 = -\langle v_2, v_3, v_1^*, v_2^* \rangle & \mathscr{A}(\sigma_4) = \langle \sigma_5, \sigma_2, \perp, \sigma_3 \rangle \\ \sigma_5 = -\langle v_3, v_1^*, v_2^*, v_3^* \rangle & \mathscr{A}(\sigma_5) = \langle \perp, \perp, \perp, \sigma_4 \rangle \end{cases}$$

Notice that the number of tuples in the $\mathscr{W}_\partial$ representation is equal to the number of $\perp$ symbols in the corresponding $\mathscr{W}$ representation.

$$\mathscr{W}_\partial(C^3) = \begin{cases} \sigma_{0,1} = -\langle v_0, v_2, v_0^* \rangle & \mathscr{A}(\sigma_{0,1}) = \langle \sigma_{2,2}, \sigma_{0,2}, \sigma_{0,3} \rangle \\ \sigma_{0,2} = +\langle v_0, v_1, v_0^* \rangle & \mathscr{A}(\sigma_{0,2}) = \langle \sigma_{1,1}, \sigma_{0,1}, \sigma_{0,3} \rangle \\ \sigma_{0,3} = -\langle v_0, v_1, v_2 \rangle & \mathscr{A}(\sigma_{0,3}) = \langle \sigma_{3,3}, \sigma_{0,1}, \sigma_{0,2} \rangle \\ \sigma_{1,1} = -\langle v_1, v_0^*, v_1^* \rangle & \mathscr{A}(\sigma_{1,1}) = \langle \sigma_{2,0}, \sigma_{3,1}, \sigma_{0,2} \rangle \\ \sigma_{2,0} = +\langle v_0^*, v_1^*, v_2^* \rangle & \mathscr{A}(\sigma_{2,0}) = \langle \sigma_{5,0}, \sigma_{2,2}, \sigma_{1,1} \rangle \\ \sigma_{2,2} = +\langle v_2, v_0^*, v_2^* \rangle & \mathscr{A}(\sigma_{2,2}) = \langle \sigma_{2,0}, \sigma_{4,2}, \sigma_{0,1} \rangle \\ \sigma_{3,1} = +\langle v_1, v_3, v_1^* \rangle & \mathscr{A}(\sigma_{3,1}) = \langle \sigma_{5,2}, \sigma_{1,1}, \sigma_{3,3} \rangle \\ \sigma_{3,3} = +\langle v_1, v_2, v_3 \rangle & \mathscr{A}(\sigma_{3,3}) = \langle \sigma_{4,2}, \sigma_{3,1}, \sigma_{0,3} \rangle \\ \sigma_{4,2} = -\langle v_2, v_3, v_2^* \rangle & \mathscr{A}(\sigma_{4,2}) = \langle \sigma_{5,1}, \sigma_{2,2}, \sigma_{3,3} \rangle \\ \sigma_{5,0} = -\langle v_1^*, v_2^*, v_3^* \rangle & \mathscr{A}(\sigma_{5,0}) = \langle \sigma_{5,1}, \sigma_{5,2}, \sigma_{2,0} \rangle \\ \sigma_{5,1} = +\langle v_3, v_2^*, v_3^* \rangle & \mathscr{A}(\sigma_{5,1}) = \langle \sigma_{5,0}, \sigma_{5,2}, \sigma_{4,2} \rangle \\ \sigma_{5,2} = -\langle v_3, v_1^*, v_3^* \rangle & \mathscr{A}(\sigma_{5,2}) = \langle \sigma_{5,0}, \sigma_{5,1}, \sigma_{3,1} \rangle \end{cases}$$

*Hypercube.* According to rule (6) we have $K^0(C^3) = \{v_0, \ldots, v_7\}$. Furthermore, it is $K^3(C^3) = \{\sigma_0, \ldots, \sigma_5\}$. We have the following decompositive winged representation for $C^4 \equiv E_1(C^3)$.

$$\mathscr{W}(C^4) = \begin{cases}
\sigma_0 = +\langle v_0, v_1, v_2, v_4, v_0^* \rangle & \mathscr{A}(\sigma_0) = \langle \sigma_1, \perp, \perp, \perp, \perp \rangle \\
\sigma_1 = -\langle v_1, v_2, v_4, v_0^*, v_9 \rangle & \mathscr{A}(\sigma_1) = \langle \sigma_2, \perp, \perp, \sigma_4, \sigma_0 \rangle \\
\sigma_2 = +\langle v_2, v_4, v_0^*, v_9, v_2^* \rangle & \mathscr{A}(\sigma_2) = \langle \sigma_3, \perp, \sigma_5, \perp, \sigma_1 \rangle \\
\sigma_3 = -\langle v_4, v_0^*, v_1^*, v_2^*, v_4^* \rangle & \mathscr{A}(\sigma_3) = \langle \perp, \sigma_6, \perp, \perp, \sigma_2 \rangle \\
\sigma_4 = +\langle v_1, v_2, v_4, v_5, v_1^* \rangle & \mathscr{A}(\sigma_4) = \langle \sigma_5, \perp, \sigma_{12}, \sigma_1, \perp \rangle \\
\sigma_5 = -\langle v_2, v_4, v_5, v_1^*, v_2^* \rangle & \mathscr{A}(\sigma_5) = \langle \sigma_6, \sigma_{13}, \sigma_2, \sigma_8, \sigma_4 \rangle \\
\sigma_6 = +\langle v_4, v_5, v_1^*, v_2^*, v_4^* \rangle & \mathscr{A}(\sigma_6) = \langle \sigma_7, \sigma_3, \sigma_9, \perp, \sigma_5 \rangle \\
\sigma_7 = -\langle v_5, v_1^*, v_2^*, v_4^*, v_5^* \rangle & \mathscr{A}(\sigma_7) = \langle \perp, \sigma_{10}, \perp, \sigma_{15}, \sigma_6 \rangle \\
\sigma_8 = +\langle v_2, v_4, v_5, v_6, v_2^* \rangle & \mathscr{A}(\sigma_8) = \langle \sigma_9, \sigma_{16}, \perp, \sigma_5, \perp \rangle \\
\sigma_9 = -\langle v_4, v_5, v_6, v_2^*, v_4^* \rangle & \mathscr{A}(\sigma_9) = \langle \sigma_{10}, \perp, \sigma_6, \perp, \sigma_8 \rangle \\
\sigma_{10} = +\langle v_5, v_6, v_2^*, v_4^*, v_5^* \rangle & \mathscr{A}(\sigma_{10}) = \langle \sigma_{11}, \sigma_7, \perp, \sigma_{18}, \sigma_9 \rangle \\
\sigma_{11} = -\langle v_6, v_2^*, v_4^*, v_5^*, v_6^* \rangle & \mathscr{A}(\sigma_{11}) = \langle \perp, \perp, \sigma_{19}, \perp, \sigma_{10} \rangle \\
\sigma_{12} = -\langle v_1, v_2, v_3, v_5, v_1^* \rangle & \mathscr{A}(\sigma_{12}) = \langle \sigma_{13}, \perp, \sigma_4, \perp, \perp \rangle \\
\sigma_{13} = +\langle v_2, v_3, v_5, v_1^*, v_2^* \rangle & \mathscr{A}(\sigma_{13}) = \langle \sigma_{14}, \sigma_5, \perp, \sigma_{16}, \sigma_{12} \rangle \\
\sigma_{14} = -\langle v_3, v_5, v_1^*, v_2^*, v_3^* \rangle & \mathscr{A}(\sigma_{14}) = \langle \sigma_{15}, \perp, \sigma_{17}, \perp, \sigma_{13} \rangle \\
\sigma_{15} = +\langle v_5, v_1^*, v_2^*, v_3^*, v_5^* \rangle & \mathscr{A}(\sigma_{15}) = \langle \perp, \sigma_{18}, \perp, \sigma_7, \sigma_{14} \rangle \\
\sigma_{16} = -\langle v_2, v_3, v_5, v_6, v_2^* \rangle & \mathscr{A}(\sigma_{16}) = \langle \sigma_{17}, \sigma_8, \perp, \sigma_{13}, \perp \rangle \\
\sigma_{17} = +\langle v_3, v_5, v_6, v_2^*, v_3^* \rangle & \mathscr{A}(\sigma_{17}) = \langle \sigma_{18}, \perp, \sigma_{14}, \sigma_{20}, \sigma_{16} \rangle \\
\sigma_{18} = -\langle v_5, v_6, v_2^*, v_3^*, v_5^* \rangle & \mathscr{A}(\sigma_{18}) = \langle \sigma_{19}, \sigma_{15}, \sigma_{21}, \sigma_{10}, \sigma_{17} \rangle \\
\sigma_{19} = +\langle v_6, v_2^*, v_3^*, v_5^*, v_6^* \rangle & \mathscr{A}(\sigma_{19}) = \langle \perp, \sigma_{22}, \sigma_{11}, \perp, \sigma_{18} \rangle \\
\sigma_{20} = -\langle v_3, v_5, v_6, v_7, v_3^* \rangle & \mathscr{A}(\sigma_{20}) = \langle \sigma_{21}, \perp, \perp, \sigma_{17}, \perp \rangle \\
\sigma_{21} = +\langle v_5, v_6, v_7, v_3^*, v_5^* \rangle & \mathscr{A}(\sigma_{21}) = \langle \sigma_{22}, \perp, \sigma_{18}, \perp, \sigma_{20} \rangle \\
\sigma_{22} = -\langle v_6, v_7, v_3^*, v_5^*, v_6^* \rangle & \mathscr{A}(\sigma_{22}) = \langle \sigma_{23}, \sigma_{19}, \perp, \perp, \sigma_{21} \rangle \\
\sigma_{23} = +\langle v_7, v_3^*, v_5^*, v_6^*, v_7^* \rangle & \mathscr{A}(\sigma_{23}) = \langle \perp, \perp, \perp, \perp, \sigma_{22} \rangle
\end{cases}$$

It is well known that the 4D cube is bounded by eight 3D cubes (see, e.g., [5]). Six boundary 3-faces of the 4-cube are obtained by extruding the six squared 2-faces of the 3-cube. The missing two boundary faces are given by the polyhedra $C^3 \times \{0\}$ and $C^3 \times \{1\}$. The hypercube representation only through its 8 cubic faces is a representation of the same kind of that one, frequently used in graphics, in which a polyhedron is represented as a set of polygonal faces. Such a representation is not considered solid, because it is missing explicit storage of topology. Conversely, the winged representation above is a *solid* representation.

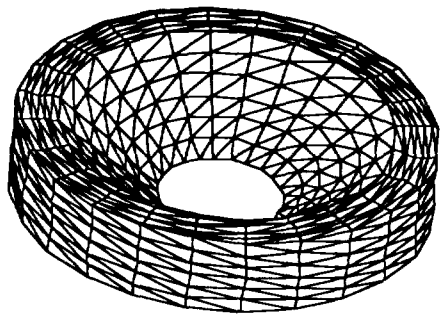


Fig. 15. The discretized domain $M$ of the example in Appendix B. The picture was attained by removing hidden faces from the polyhedron Boundary $M$.

## APPENDIX B

### Field Modeling and Rendering

In this Appendix an example is presented of the combined use of various operators in order to model both a curved-manifold object and a time-varying scalar field defined on it. The language, algorithms, and representation introduced in the paper are used to obtain several snapshots of the field at different locations and time instants (see Figure 16).

Consider the 3D polyhedron, embedded in $\Re^3$, shown in Figure 15. It has been attained with $Simple_X^n$ as a map over a 3D grid, by the expression

$$\begin{aligned} \mathrm{M} = \mathsf{Map}(&\phi_1(u_1, u_2, u_3), \phi_2(u_1, u_2, u_3), \phi_3(u_3)) \\ &\mathsf{Scale}(2\pi, 1, 1)\ \mathsf{Grid}(48, 6, 6)o \end{aligned}$$

where

$$\begin{aligned} \phi_1(u_1, u_2, u_3) &= ((1 - u_2)r_1 + (1 - u_2)u_3(r_1 - r_2) + u_2 R)\cos u_1 \\ \phi_2(u_1, u_2, u_3) &= ((1 - u_2)r_1 + (1 - u_2)u_3(r_1 - r_2) + u_2 R)\sin u_1 \\ \phi_3(u_3) &= Hu_3, \end{aligned}$$

$r_1, r_2, R, H$ are constants, and $o$ is the same as in Example 4. Suppose we now want to model a scalar field $v = v(x, y, z, t)$ where $t$ is the time. We can add information about values of $v$ over the discretization of the model by adding a time grid and then map $v$ over the whole domain. This can be accomplished with the following expression:

$$\begin{aligned} F = \mathsf{Map}&(u_1, u_2, u_3, u_4, v(u_1, u_2, u_3, u_4)) \\ &\mathsf{Grid}(12)\mathrm{M} \end{aligned}$$

having chosen for the time a discretization of 12 steps in the interval $[0, 1]$. Notice that $F \in \mathscr{P}^{4,5}$. Visualization of the field for fixed values of $t$ and $z$, or $t$ and $v$ (iso-valued surfaces), can be obtained by evaluating and displaying the expressions

$$\begin{aligned} ZT = \mathsf{Map}&(u_1, u_2, z + v(u_1, u_2, z, t)) \\ &\mathsf{Map}(\phi_1(u_1, u_2, z), \phi_2(u_1, u_2, z)) \\ &\mathsf{Scale}(2\pi, 1)\ \mathsf{Grid}(48, 6)\ o \\ VT = \mathsf{Project}&(4, 5) \\ &\mathsf{Intersect}\ F, V, T \end{aligned}$$

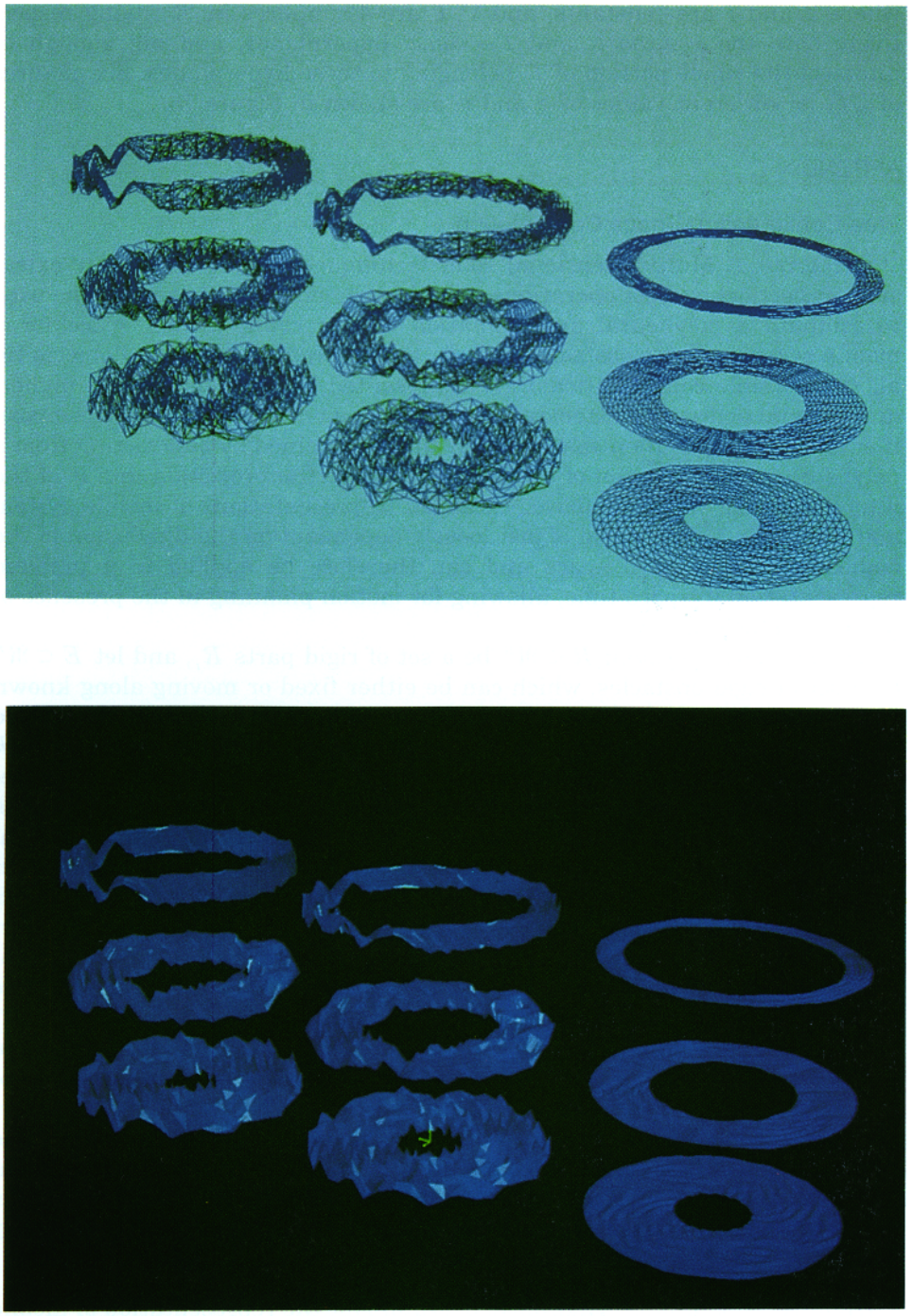


**Fig. 16.** The field $v = A\cos\{k[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2]\}\cos ft$, where $A, k, f, x_0, y_0, z_0$ are constants. Pictures correspond to three different values of the time $t$ and to three values of $z$, the height of the solid in Figure 15.

where $z$ and $t$ are constants, and $V, T$ are 4-simplices in $\Re^5$, lying respectively onto the $v = const$, and $t = const$ hyperplanes, and big enough to intersect the whole portion of $F$ belonging to these hyperplanes. Six pictures of $ZT$, for different values of $z$ and $t$, are shown in Figure 16.

## APPENDIX C

### Free Configuration Space Computation

*Background.* Multidimensional solid techniques, and in particular extrusion, projection, and set operations in higher-dimensional spaces, can be used to compute a polyhedral approximation of free configuration space for a mobile system amidst obstacles [44]. In this approach, dynamical objects (in all the possible configurations) are represented as static objects in higher-dimensional spaces, by extensively using extrusions. A simplicial decomposition of free configuration space is then simply obtained by subtracting from a polyhedral representation of configuration space the projection into it of the intersection between the above "extended" representations of mobile system and obstacles. This method is just loosely dependent on the dimension of the embedding geometrical space and can therefore be applied to a problem representation in space-time, allowing for motion planning in the presence of moving obstacles.

Let the moving system $R \subset \Re^n$ be a set of rigid parts $R_i$, and let $E \subset \Re^n$ be a set of rigid obstacles, which can be either fixed or moving along known trajectories. In practical cases we have $n \leq 4$, with $n = 4$ when $R \cup E$ are modeled in space-time. The parts of the moving system have a total number $k$ of degrees of freedom, which equates the minimal number of scalar parameters necessary to uniquely determine their position and orientation. It is assumed that the degrees of freedom are restricted to be either translational or rotational. The Configuration Space $CS = I_1 \times \cdots \times I_k \subset \Re^k$ is the product space of range intervals $I_i$ of configuration parameters. Each point in CS represents a distinct configuration of the moving system, but just a compact subset $FP \subseteq CS$, called free configuration space, contains feasible configurations, where the $R$ elements neither intersect nor intersect the obstacles. The Extended Configuration Space is defined in [44] as $ECS = \Re^n \times CS \subset \Re^{n+k}$.

An ECS representation of $R \cup E$ encodes, within a higher-dimensional polyhedron, the whole set of all the feasible and unfeasible configurations of $R \cup E$. In particular, a polyhedron obtained by linearly extruding $R_i$ represents all the possible positions of $R_i$ in its translational motion. The same property holds for a screw extrusion of $R_i$, when dealing with a rotational degree of freedom. The ECS representation of $R \cup E$ can be used to compute, with solid-modeling tools (intersection, difference, projection), the free configuration space FP, as shown in [44].

*High-level operators.* High-level operators for the representation of mobile systems can easily be defined in terms of the extrusion operators. Consider an object represented by the structure $S$, and suppose that $S$ is free to translate with translation given by $T_{\lambda t}$, with $\lambda \in [0, 1]$. The ECS representa-

tion of this system can be expressed as Move($\boldsymbol{t}$)$S$, where the operator Move, with parameter $\boldsymbol{t} = [t_1, \ldots, t_n]$ results in a linear extrusion $LE_{[t_1,\ldots,t_n,1],1}: \mathscr{P}^{d,n} \to \mathscr{P}^{d+1,n+1}$. Conversely, suppose that only some elements of the structure $S$ are able to translate. In this case all the configurations of $S$ are represented by applying a linear extrusion $LE_{[t_1,\ldots,t_n,1],1}$ to the translating substructures and a straight extrusion $E_1$ to the remaining elements. Hence the default operator for Move is Extrude with parameters $([0, \ldots, 0, 1], 1)$, i.e., a straight extrusion.

Recalling the definition of the operators with production rules, it is easy for us to see that the ECS representation for such a structure can be expressed by using Move with the optional *indices* specifier; e.g., if $S = \langle S_1, S_2, S_3, S_4 \rangle$ and if only $S_1$ and $S_3$ are free to translate, with translation vector $\boldsymbol{t}$, then the ECS representation of $S$ is given by $\text{Move}^{1,3}(\boldsymbol{t})S$.

A rotational degree of freedom is modeled by using the Joint operator. The Joint operator, with parameters $c = (c_1, \ldots, c_n)$, $\theta$, $i$, $j$, $h$, results in the mapping $\mathscr{P}^{d,n} \to \mathscr{P}^{d+1,n+1}$ given by

$$T_{[c_1,\ldots,c_n,0]} \circ SE_{\theta,i,j,h} \circ T_{-[c_1,\ldots,c_n]}$$

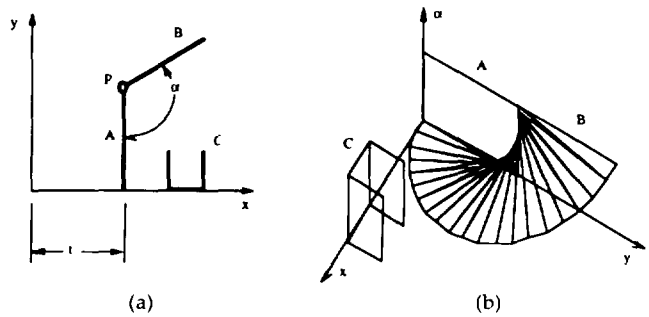
This corresponds to moving the point $c$ (one of the fixed points of the isometry) to the origin, then performing a screw extrusion, thereafter moving $c$ back. According to the previous discussion, the default operator for Joint is Extrude $([0, \ldots, 0, \theta], 1)$.

The ECS representation of a moving system which is able to rotate by an angle $\theta$ in the plane $x_i$, $x_j$ with a fixed point $c$ is *approximated* (with a linear polyhedron, in $h$ steps) as Joint$(c, \theta, i, j, h)S$, where $S$ is a structure which represents the geometry of the system. When some portions of the system do not move with such rotation, a screw extrusion and a linear extrusion $LE_{[0,\ldots,0,\theta],1}$ have to be applied to the rotating and nonrotating substructures, respectively. This can be accomplished by using Joint with the optional *indices* specifier.

*Example.* Consider the 2D simple robot, constituted by two schematic rigid arms, shown in Figure 17. The robot is free to translate along the $x$ axis in the interval [0, 10], while the arm B is able to rotate around the point $p = (0, 5)$ with maximum angle $\pi$. An obstacle C is located in the workspace of the robot. The ECS description of the whole system (robot and obstacle) can be obtained by evaluating the expression

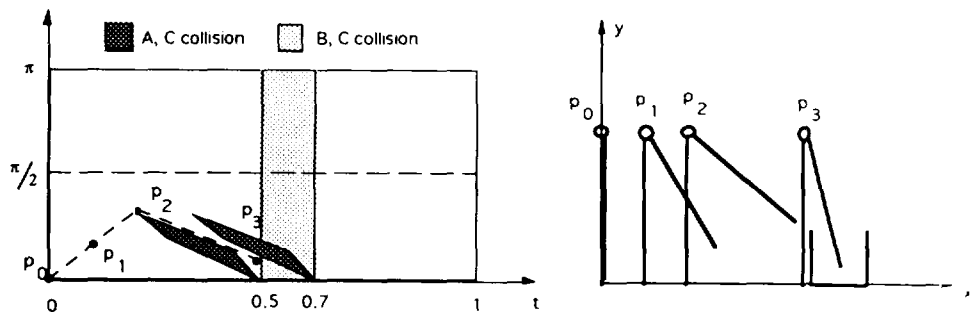
$$S = \text{Move}^{1,2}([10, 0, 0])$$
$$\text{Joint}^{2}((0, 5), \pi, 1, 2, 20)\ \langle \text{A}, \text{B}, \text{C} \rangle$$

where both A and B are the segment in $\Re^2$ with vertices (0, 0) and (0, 5) and where C is the 1-dimensional polyhedron in $\Re^2$ representing the obstacle. Notice that B, in its initial position, coincides with A. The structure $S$ resulting from the evaluation of the expression is composed of three 3D polyhedra in $\Re^4$. Each system configuration can be obtained by intersecting $S$ with the sets $\alpha = a$, $t = b$, where $0 \le a \le \pi$ and $0 \le b \le 10$.




(a) (b)

Fig. 17. (a) The robot and the obstacle used in the example of Appendix C; (b) The result of the first extrusion.




(a) (b)

Fig. 18. (a) The free configuration space for the robot and the obstacle of Figure 17a; (b) The collision-free task planned in (a).

The first step in the evaluation of the expression above is the screw extrusion of B. A and C are straight extruded. It is possible to see in Figure 17b that taking a section with a hyperplane $\alpha = cost$ and projecting the section onto the $x$, $y$ subspace, we obtain a "snapshot" of the system in a configuration where the arm B is rotated by $\alpha$ degrees, while A and C are in their original positions. The second step is the linear extrusion of both A and B, representing the translational degree of freedom of the robot. A straight extrusion is again applied to C. The whole expression results in three 3D polyhedra embedded in $\Re^4$. Two projections in $\Re^3$ of each polyhedron have been depicted in Figure 19.

In order to compute the free configuration space for this robot, we first have to compute the intersections between the ECS representations of the arms:

$E$ = Intersect $S_{1,2}$.

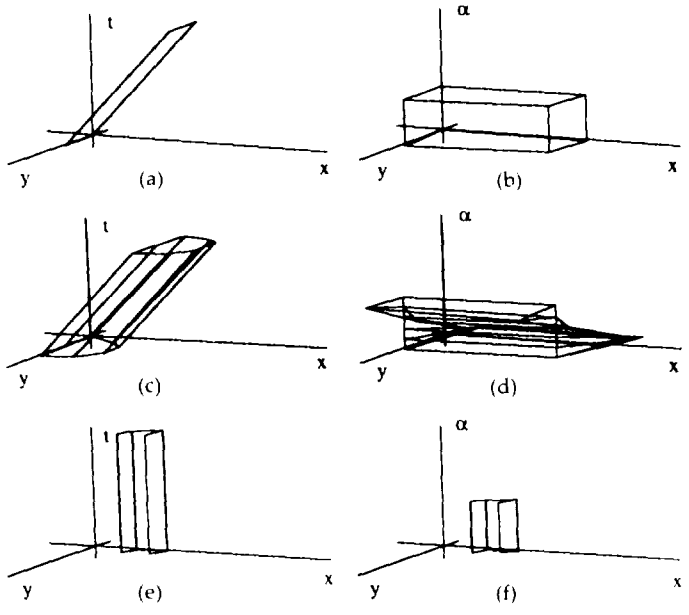



Fig. 19. The result of the second extrusion. Each object, a 3D polyhedron in $\Re^4$, is represented by means of two projections in $\Re^3$; (a), (b): extrusion of $A$; (c), (d): extrusion of $B$; (e), (f): extrusion of $C$.

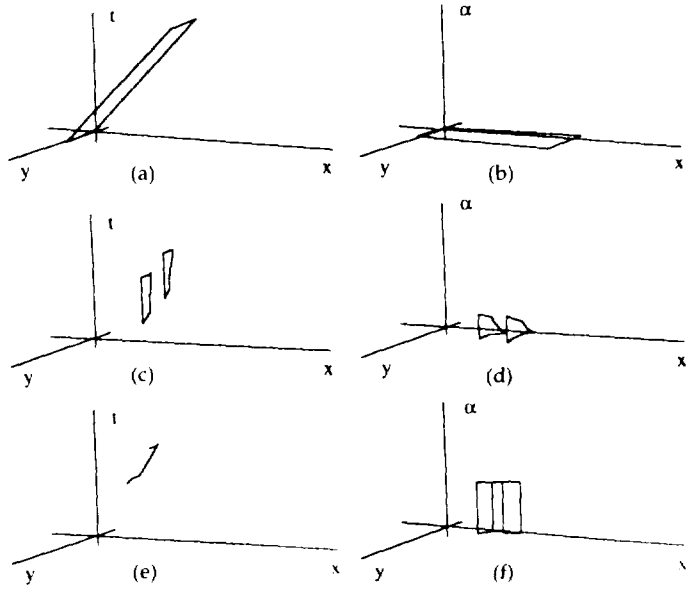



Fig. 20. The results of intersections between the objects shown in Figure 19. Two projections in $\Re^3$ for each polyhedron are shown; (a), (b): intersection of $A$, $B$; (c), (d): intersection of $B$, $C$; (e), (f): intersection of $A$, $C$.

The intersection between the whole robot and the obstacle is given by

$F$ = Intersect ⟨Union $S_{1,2}, S_3$⟩.

Projections in $\Re^3$ of these intersections are shown in Figure 20. Finally, the free configuration space FP can be computed, according to the method proposed in [44], as

$FP$ = Difference
⟨Scale ([1, $\pi$]) Grid(1, 1) $o$,
Project(1, 2) Union⟨$E, F$⟩⟩

where $o$ is defined in Section 2.2. In Figure 18a the free configuration space FP is shown, together with the collision-free path in FP for the robot task shown in Figure 18b.

## GLOSSARY

| | |
|---|---|
| $\Re^n$ | Euclidean $n$-dimensional space |
| $PQ$ | join of the two sets $P$, $Q \subset \Re^n$ |
| $\langle \cdots \rangle$ | ordered set |
| $\sigma, \tau$ | simplices |
| $\Sigma^d$ | simplicial $d$-complex |
| $[\Sigma^d]$ | geometric carrier of the complex $\Sigma^d$ |
| $K^s(\Sigma^d)$ | $s$-skeleton of a complex, i.e., the set of $s$-simplices in $\Sigma^d$ |
| $P$ | polyhedron |
| $\mathscr{P}^{d,n}$ | set of all linear $d$-polyhedra embedded in $\Re^n$ |
| $o$ | the only polyhedron contained in $\mathscr{P}^{0,0}$, a point |
| $\partial P$ | boundary of $P$ |
| $K^o(P)$ | set of vertices of $P$ |
| $K^d(P)$ | set of simplices of a simplicial decomposition of $P \in P^{d,n}$ |
| $\partial P$ | boundary of $P$ |
| $K^0(P)$ | set of vertices of $P$ |
| $K^d(P)$ | set of $d$-simplices of a simplicial decomposition of $P \in \mathscr{P}^{d,n}$ |
| $\sigma_{i,j}$ | $j$th face of the simplex $\sigma_i$, as defined by Formula (1) |
| $\times$ | Cartesian product |
| $\times_i S$ | Cartesian product of $i$ instances of the set $S$ |
| $\|S\|$ | cardinality of the set $S$ |
| $\mathscr{A}$ | adjacency function |
| $\mathscr{A}_h(\sigma_p)$ | $h$th value in the adjacency tuple $\mathscr{A}(\sigma_p)$, i.e., the $d$-simple adjacent to $\sigma_p$ along its $h$th face |
| $\perp$ | "undefined" adjacency |
| $\mathscr{W}(P)$ | decompositive winged representation of $P$ |
| $\mathscr{W}_\partial(P)$ | boundary winged representation of $P$ |
| $\lambda$ | scalar |

| | |
|---|---|
| $\boldsymbol{\lambda}$ | vector |
| $p$ | point |
| $T, S, H$ | translation, scaling, and shearing |
| $R_{i,j,\alpha}$ | rotation of $\alpha$ in the $x_i$, $x_j$ plane |
| $E_h(P)$ | straight extrusion of $P$ with number of steps $h$ |
| $LE_{\mathbf{v},h}(P)$ | linear extrusion of $P$ with extrusion vector $\mathbf{v}$ and number of steps $h$ |
| $SE_{\theta,i,j,h}(P)$ | screw extrusion of $P$ with rotation angle $\theta$, rotation plane $x_i$, $x_j$, and number of steps $h$ |
| $G^m_{h_1,\ldots,h_m}(P)$ | grid over $P$ |
| $\boldsymbol{f}, \boldsymbol{\phi}$ | vector-valued functions |
| NURBS | Non-Uniform Rational B-spline mapping |
| $\Pi_i(P)$ | projection of $P$ along the $x_i$ axis |
| $\cup^*, \cap^*, -^*$ | regularized set operations |
| $conv\, U$ | convex hull of the set $U$ |

ACKNOWLEDGMENTS

The authors have been working on the ideas discussed in this paper for some years. At this point, where diverse materials have been collected and made coherent, we wish to point out that significant preliminary steps which led to this work are represented by [16], where the background ideas appeared, and [8], where a more complete scenery was designed.

The authors wish to sincerely thank J. Rossignac for many comments, suggestions, and advice on previous versions of this paper, which led to a great improvement of our work, and the anonymous referees for their careful reading and comments. Several discussions with A. Di Carlo about atlases of manifolds and simplicial approximation of differential operators are also gratefully acknowledged. Finally, we wish to thank the "Edilizia" Project Committee of the Italian National Research Council for supporting this research. The $Simple^n_X$ kernel was implemented by F. Bernardini and V. Ferrucci, the mapping subsystem by V. Pascucci and E. Vietri, a prototype interpreter for the language interface by G. Sindoni and M. Vicentino.

REFERENCES

1. ARMSTRONG, W. P., AND BURTON, R. P. Perception cues for $n$-dimensions. *Comput. Graph. World* (Mar. 1985), 11–28.
2. BACKUS, J., WILLIAMS, J., WIMMES, E., LUCAS, B., AND AIKEN, A. FL language manual, parts 1 and 2. Tech. Rep. RJ7100 (67163), IBM Almaden Res. Center, 1989.
3. BAJAJ, C. Rational hypersurface display. *Comput. Graph. 24*, 2 (1990), 117–128.
4. BANCHOFF, T. F. Realtime computer graphics analysis of figures in four-space. In *Hypergraphics*. Westview Press, Boulder, Col., 1978.
5. BANCHOFF, T. F. Discovering the fourth dimension. Tech. Rep., Dept. of Mathematics, Brown Univ., Providence, R.I., 1987.
6. BERNARDINI, F. $Simple^n_X$: User manual and implementation notes. Part 2. Tech. Rep. 20-90, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1990.

7. BERNARDINI, F. Integration of polynomials over $n$-dimensional polyhedra. *Comput. Aided Des. 23*, 1 (Feb. 1991), 51–58.
8. BERNARDINI, F., FERRUCCI, V., AND PAOLUZZI, A. Working with dimension-independent polyhedra. Tech. Rep. 07-91, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1991.
9. BIERI, H., AND NEF, W. Elementary set operations with $d$-dimensional polyhedra. In *Computational Geometry and its Applications*, H. Noltemeier, Ed. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1988, 97–112.
10. BRISSON, E. Representing geometric structures in $d$ dimensions: Topology and order. In *ACM Symposium on Computational Geometry*. ACM Press, New York, 1989.
11. BROUWER, L. *Collected Works*, Vol. 1. North Holland, Amsterdam, 1975.
12. BURTON, R. P., AND SMITH, D. R. A hidden-line algorithm for hyperspace. *SIAM J. Comput. 11*, 1 (1982), 71–80.
13. CAMERON, S. A. Modelling solids in motion. Ph.D. dissertation, Univ. of Edinburgh, U.K., 1984.
14. CAMERON, S. A. Collision detection by four-dimensional intersection testing. *IEEE Trans. Robotics Automat. 6*, 3 (June 1990), 291–302.
15. CATTANI, C., AND PAOLUZZI, A. A topological approach to space-time modeling. In *IMACS International Symposium on System Modeling and Simulation* (Cetraro, Italy, Sept., 1988), T. Tzafestas, A. Eisenberg, and L. Carotenuto, Eds. Elsevier Science, 1989, 61–66.
16. CATTANI, C., AND PAOLUZZI, A. Solid modeling in any dimension. Tech. Rep. 02-89, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1989.
17. CATTANI, C., AND PAOLUZZI, A. Boundary integration over linear polyhedra. *Comput. Aided Des. 22*, 2 (Mar. 1990), 130–135.
18. CHEN, P., HANSEN, P., AND JAUMARD, B. On-line and off-line vertex enumeration by adjacency lists. Tech. Rep. RUTCOR 9-90, Rutgers Center for Operations Research, Rutgers Univ., New Brunswick, N.J., 1990.
19. CHERNIKOVA, N. V. Algorithm for finding a general formula for the nonnegative solutions of a system of linear equations. *U.S.S.R. Comput. Math. Math. Phys. 5* (1965), 228–233.
20. DIEUDONNÉ, J. *A History of Algebraic and Differential Topology 1900–1960*. Birkhäuser, Boston, 1989.
21. DOBKIN, D. P., AND LASZLO, M. J. Primitives for the manipulation of three-dimensional subdivisions. In *ACM Symposium on Computational Geometry*. ACM, New York, 1987, 86–99.
22. FERRUCCI, V. *Simple$^n_X$*: User manual and implementation notes. Part 1. Tech. Rep. 06-90, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1990.
23. FERRUCCI, V., AND BERNARDINI, F. Boolean operations over multidimensional polyhedra using linear programming, symbolic manipulation and simplicial decompositions. Tech. Rep. 13-91, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1991.
24. FERRUCCI, V., AND PAOLUZZI, A. Extrusion and boundary evaluation for multidimensional polyhedra. *Comput. Aided Des. 23*, 1 (Feb. 1991), 40–50.
25. FERRUCCI, V., AND VANĚČEK, G., JR. A spatial index for convex simplical complexes in $d$ dimensions. In *Advances in Spatial Databases*, O. Günther and H.-J. Schek, Eds. In Lecture Notes in Computer Science, vol. 525. Springer-Verlag, Berlin, 1991, 361–380.
26. FUCHS, H. On visible surface generation by a priori tree structures. *Comput. Graph. 14* (1980). 124–133.
27. GIBLIN, P. J. *Graphs, Surfaces and Homology*. Chapman and Hall, London, 1977.
28. GLASSNER, A. S. Spacetime ray tracing for animation. *IEEE Comput. Graph. Appl., 8*, 2 (1988), 60–70.
29. GUIBAS, L., AND STOLFI, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph. 4*, 2 (1985), 74–123.
30. HAMMER, P. C., AND STROUD, A. H. Numerical integration over simplexes. *Math. Tables Aids Comput. 10* (1956), 137–139.
31. HAMMER, P. C., MARLOWE, O. J., AND STROUD, A. H. Numerical integration over simplexes and cones. *Math. Tables Aids Comput. 10* (1956), 130–137.

32. HOFFMANN, C. M., AND ZHOU, J. Some techniques for visualizing surfaces in four-dimensional space. *Comput. Aided Des., 23*, 1 (Feb. 1991), 83–91.
33. LEFSCHETZ, S. *Introduction to Topology*. Princeton University Press, Princeton N.J., 1949.
34. LIENHARDT, P. Subdivisions of $n$-dimensional spaces and $n$-dimensional generalized maps. In *ACM Symposium on Computational Geometry*. ACM Press, New York, 1989, 228–236.
35. LIENHARDT, P. Topological models for boundary representation: A comparison with $n$-dimensional generalized maps. *Comput. Aided Des. 23*, 1 (Feb. 1991), 59–82.
36. LOZANO-PEREZ, T., AND WESLEY, M. A. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM 22*, 1 (1979), 560–570.
37. MATHEISS, T. H., AND RUBIN, D. S. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Math. Oper. Res. 5*, 2 (1980), 167–185.
38. MAÑAS, M., AND NEDOMA, J. Finding all vertices of the convex polyhedron. *Numer. Math. 12* (1968), 226–229.
39. MONTGOMERY, K. M. A constructive solid geometry scheme for representing multidimensional graphical information. Master's thesis, Brigham Young Univ., 1984.
40. MURTY, K. G. *Linear Programming*. Wiley, New York, 1983.
41. NEF, W. *Beiträge zur Theorie der Polyeder—mit Anwendungen in der Computergrafik*. Herbert Lang, Bern, 1978. In German.
42. NOLL, A. M. Computer technique for displaying $n$-dimensional hyperobjects. *Commun. ACM 10*, 8 (Aug. 1967), 469–473.
43. NOLL, A. M. Computer animation and fourth dimension. In *AFIPS Conference Proceedings*. AFIPS, 1968.
44. PAOLUZZI, A. Motion planning + solid modeling = motion modeling. Tech. Rep. 17-89, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1989.
45. PAOLUZZI, A., RAMELLA, M., AND SANTARELLI, A. Boolean algebra over linear polyhedra. *Comput. Aided Des. 21*, 8 (1989), 474–484.
46. PAOLUZZI, A., AND SANSONI, C. Programming language for solid variational geometry. *Comput. Aided Des. 24*, 7 (July 1992), 349–366.
47. PAOLUZZI, A., AND SANSONI, C. Solid modeling of architectural design with PLASM language. In Proceedings *of the CAAD Futures '91 Conference* Vieweg, Weisbaden, 1992.
48. PAOLUZZI, A., AND VIETRI, E. Representation and rendering of manifolds and fields. In Proceedings *of IcoGraphics '91* (Milan, Italy, Mar., 1991). Mondadori Informatica, Milano, 541–552. In Italian.
49. PASCUCCI, V. *Simple$^n_X$*: User manual and implementation notes. Part 3. Tech. Rep. 18-91, Dip. di Informatica e Sistemistica, Univ. di Roma "La Sapienza," Rome, Italy, 1991.
50. PEGNA, J. Variable sweep geometric modeling. Ph.D. dissertation, Stanford Univ., Dept. of Mechanical Engineering, Stanford, Calif., 1987.
51. POINCARÉ, H. *Oeuvres*, vol. 6. Gauthier-Villars, Paris, 1953. In French.
52. PUTNAM, L. K., AND SUBRAHMANYAM, P. A. Boolean operations on $n$-dimensional objects. *IEEE Comput. Graph. Appl. 6*, 6 (1986), 43–51.
53. REQUICHA, A. A. G. Mathematical models of rigid solid objects. Tech. Rep. 28, Production Automation Project, Univ. of Rochester, Rochester, N.Y., 1977.
54. REQUICHA, A. A. G. Representations for rigid solids: Theory, methods and systems. *ACM Comput. Surv. 12*, 4 (Dec. 1980), 437–464.
55. REQUICHA, A. A. G. Solid modeling: A 1988 update. In *CAD Based Programming for Sensory Robots*, B. Ravani, Ed. In Nato ASI Series, vol. 50. Springer-Verlag, New York, 1988, 3–22.
56. ROGERS, D. F., AND ADAMS, J. A. *Mathematical Elements for Computer Graphics*. 2nd ed. McGraw-Hill, New York, 1990.
57. ROSSIGNAC, J., AND O'CONNOR, M. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In *Geometric Modeling for Product Engineering* (Rensselaerville, N.Y., Sept., 1988). M. J. Wozny, J. Turner, and K. Preiss, Eds. In Proceedings *of the 1988 IFIP/NSF Workshop on Geometric Modelling*. North Holland, Amsterdam, 1990, 145–180.
58. ROSSIGNAC, J. R., AND REQUICHA, A. A. G. Constructive non-regularized geometry. *Comput. Aided Des. 23*, 1 (Feb. 1991), 21–32.

59. ROURKE, C. P., AND SANDERSON, B. J. *Introduction to Piecewise-Linear Topology*. Springer-Verlag, Berlin, 1972.
60. RUPPERT, J., AND SEIDEL, R. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. In *ACM Symposium on Computational Geometry*. ACM Press, New York, 1989.
61. SAMET, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Mass., 1990.
62. VANĚČEK, G. JR. Brep-index: A multidimensional space partitioning tree. In *ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications* (Austin, Tex., 1991). J. Rossignac and J. Turner, Eds. ACM Press, New York.
63. VON HOHENBALKEN, B. Finding simplicial subdivisions of polytopes. *Math. Program. 21* (1981), 233–234.
64. WELD, J. Geometric representation of swept volumes with application to polyhedral objects. Ph.D. dissertation, Cornell Univ., Sibley School of Mechanical and Aerospace Engineering, Ithaca, N.Y., 1987.
65. WHITNEY, H. *Geometric Integration Theory*. Princeton University Press, Princeton, N.J., 1957.
66. WOODWARK, J. R. Eliminating redundant primitives from set-theoretic solid models by a consideration of constituents. *IEEE Comput. Graph. Appl. 8*, 3 (1988), 38–47.
67. ZHANG, S. Boolean operations in cell decomposition based constructive solid geometry. In *International Conference on Die and Mould Technology* (Shanghai, May 1990).

Received April 1991; accepted March 1992; revised September 1992