PRIMITIVES FOR THE MANIPULATION OF THREE-DIMENSIONAL SUBDIVISIONS

David P. Dobkin
Michael J. Laszlo

CS-TR-089-87

April 1987

Primitives for the Manipulation of Three-Dimensional Subdivisions

David P. Dobkin
Michael J. Laszlo

Department of Computer Science
Princeton University
Princeton, New Jersey 08540

1. Introduction

A major impediment to the implementation of algorithms that manipulate 3-dimensional cell complexes and subdivisions is the lack of a suitable data structure. What is needed is a data structure powerful enough to model such objects yet simple enough to allow their manipulation in well defined ways. We focus attention here on the development of such a data structure. Our structure is analogous (though one dimension higher) to the winged-edge [Ba], [BHS], [EW] and quad-edge [GS] data structures which are widely accepted for modelling 2-manifolds. Just as these structures can be used to represent both planar polygonal cell complexes in R^2 and surfaces of polyhedra, our data structure can model polyhedral complexes in R^3 and surfaces of 4-polyhedra.

Our results can be viewed as similar to the work done by Guibas and Stolfi in deriving the quad-edge structure. Lifting the results one dimension higher increases the complexity of our data structure. They consider an edge as their atom, and consider the edge rings to which it belongs. We consider a polygon-edge pair as an atom, and consider the two polygon rings and two edge rings to which it belongs. The quad-edge atom could be considered to connect two vertices and two polygons. Similarly, our atom connects two polyhedra and two vertices. We simplify our structure by treating only complexes that are orientable, and whose cells do not puncture the interior of other cells.

There are numerous applications we envision for such a data structure. One application we consider is that of decomposing a polyhedron into tetrahedra [Ch, Wö]. We rederive and model one of the applications [Wö] in our system. A second application we consider is the implementation of an algorithm for incrementally computing the Delaunay triangulation of a 3-dimensional point set [AB], [Bh]. Further applications are also possible. For one, our data structure provides an approach to an efficient divide-and-conquer algorithm for building 3-dimensional Voronoi diagrams. A second possibility is for modelling the motion of a 3-dimensional polyhedron through time, which can be viewed as a 4-dimensional polyhedron (in x, y, z, t space) where hidden surface removal is done by projecting into x, y, t space and taking t cross-sections to determine individual scenes.

What we attempt to achieve in this paper is a blend between a derivation of the data structure and a small set of primitive operators for its manipulation, the development of macro operations from these primitives, and the use of these macros in the first two applications mentioned above. The results of this paper are implementable and an effort to build them is currently underway.

2. Definitions and Prerequisites

In this section we define the class of objects to be manipulated by our data structure. It is assumed the reader is familiar with some basic concepts of point-set topology.

2.1 Basic Definitions

Where T is a topological space, a *k-cell* is a closed subspace of T whose interior is homeomorphic to R^k, and whose boundary is non-null. In this paper, we assume that $T = R^3$, though our results hold for more general T. We call a 0-cell a *vertex*, a 1-cell an *edge*, a 2-cell a *polygon* or a *facet*, and a 3-cell a *polyhedron*. Note that a cell may be unbounded; for instance, an edge can be a closed segment (bounded by two vertices) or a ray (bounded by one vertex). We also assume that each k-cell c lies in a k-dimensional affine subspace, denoted *aff* c.

This work supported in part by the National Science Foundation under Grant DCR85-05517.

A *(closed) cell complex* of T is a finite collection C of cells of T such that

(i) the relative interiors of cells of C are pairwise disjoint, and

(ii) for each cell $c \in C$, the boundary $bd\, c$ of cell c is the union of elements of C, and

(iii) if $c, d \in C$ and $c \cap d \neq \emptyset$, then $c \cap d$ is the union of elements of C.

We let $\mathcal{U}(C)$ be the union of the cells of C, and consider C a subdivision of $\mathcal{U}(C)$. An n-dimensional complex for which every every k-cell is contained in (the boundary of) some n-cell is called an *n-complex.* Informally, an n-complex is a complex possessing no struts, superfluous cells that do not help to define polyhedra.

The combinatorial boundary of cell c of C, denoted ∂c, is defined to be the set of cells of C contained in $bd\, c$. Note that $\mathcal{U}(\partial c) = bd\, c$. The combinatorial boundary ∂C of complex C is defined as the set of cells of C contained in $bd\, \mathcal{U}(C)$. An open cell $d \subset c$ is said to be a *face* of c; if in addition $c \neq d$, then d is a *proper face* of c. If one of c or d is a proper face of the other, c and d are said to be *incident*. For instance, a polyhedron is incident with each vertex, edge and facet that lies in its boundary. The *star* of a cell c, denoted $star\, c$, is the subset of C consisting of the cells of which c is a face.

In this paper we consider two types of 3-complexes. One type, called *ball complexes*, are subdivisions of space homeomorphic to the closed ball $D^3 = \{x \in R^3 \mid |x| \leq 1\}$. We regard a ball complex as a subspace of R^3. The other type of 3-complexes we consider are the subdivisions of R^3. Observe that in such complexes, the star of each edge contains a facet-polyhedron cycle, each cell of which occurs in the cycle exactly once. Similarly, each facet is bounded by a simple vertex-edge cycle.

Given n-complex C, by convention there exists one (null) $n+1$-cell of which every n-cell of C is a face; likewise there exists one (null) -1-cell which is a face of every vertex. Distinct k-cells c and d (for $0 \leq k \leq n$) are then said to be adjacent if (i) there exists some $k-1$-cell of C that is a face of both c and d, and (ii) there exists some $k+1$-cell of C of which each of c and d is a face. For instance, two vertices connected by an edge are adjacent; in addition, two facets incident to the same polyhedron and the same edge are adjacent.

A closed disk $D^2 = \{x \in R^2 \mid |x| \leq 1\}$ can be oriented in either of two ways. One orientation turns counter-clockwise about the disk, while the other turns clockwise about the disk, as the disk is viewed from one of its sides. We henceforth refer to the orientation of a disk as *clocking*, and a polygon with orientation as a *clocked polygon*.

A closed ball D^3 can be (space)-oriented in either of two ways. If we imagine a diameter d of the ball directed from point p on the sphere $bd\, D^3$ to antipodal point p' on $bd\, D^3$, a right-handed orientation (RH orientation) turns counter-clockwise about d, while a left-handed-orientation (LH orientation) turns clockwise about d, as d is viewed from p' to p. A directed edge with orientation is called an *oriented edge*.

2.2 Space-Duality

The space-dual of a complex C of space T is a second complex C^* of T for which there exists a one-to-one mapping Ψ from C onto C^* such that

(i) the image of a k-cell under Ψ is an $n-k$-cell, and

(ii) cells c and d are adjacent in C iff cells $\Psi(c)$ and $\Psi(d)$ are adjacent in C^*.

In particular, with respect to 3-complexes C and C^*, each vertex (edge) of one corresponds to a polyhedron (facet) of the other, and adjacency relations between cells are preserved. The space-dual of cell c, denoted c^*, is that cell which corresponds to c under Ψ: $c^* = \Psi(c)$ for $c \in C$, and $c^* = \Psi^{-1}(c)$ for $c \in C^*$.

The complex C^* space-dual to C is by no means unique. However, up to the topological property which we intend our data structure to represent — adjacency relations between cells — the numerous complexes that serve as space-dual to C in T are identical. For our purposes, C^* is well-defined. Furthermore, $(C^*)^* = C$.

Given ball complex C, we denote by p^∞ that unbounded "polyhedron" whose boundary coincides with that of C. The space-dual of C is a subdivision C^* of R^3, one of whose vertices is a point at infinity whose space-dual is p^∞.

3. Traversal Functions

In this section we present the five traversal functions *Fnext*, *Enext*, *Rev*, *Clock* and *Sdual*. We call these *traversal* functions because they provide the means of traversing or moving about the cells of a complex. The first two traversal functions are used to move from cell to adjacent cell of a complex. *Rev* and *Clock* are used to change a local sense of direction, so *Fnext* and *Enext* know the direction in which each is to traverse. The function *Sdual* is used to move between a complex and its space-dual. Since edges and facets interchange in the space-dual, the roles of *Fnext* and *Enext* are interchanged in going between C and C^*.

Also presented are the vertex functions *Org*, *Dest*, *Ppos* and *Pneg*, with which one can determine the two vertices incident to an edge (its endpoints), and the two polyhedra incident to a facet. They are called *vertex* functions since the polyhedra incident to some facet in a complex correspond to the vertices incident to some edge in the complex's space-dual.

3.1 Basic Traversal Functions

Let f be a facet of complex C. The combinatorial boundary of f contains a ring of edges $e^0 \dots e^{n-1}$ where edges e^i and e^{i+1} are adjacent in C (addition modulo n). We call this ring, denoted $\mathcal{E}_f$, the edge-ring of facet f. $\mathcal{E}_f$ can be assigned either of two directions whereby we can distinguish between the two edges belonging to the ring that are adjacent to edge e^i. We write $\mathcal{E}_f = (e^0 \dots e^{n-1})$ to indicate the edge-ring with direction such that, of the two edges e^{i-1} and e^{i+1} of $\mathcal{E}_f$ adjacent to e^i, e^{i-1} precedes and e^{i+1} follows, edge e^i.

Similarly we define the facet-ring of edge e, denoted $\mathcal{F}_e$, to be the ring of facets $\mathcal{F}_e = (f^0 \dots f^{m-1})$ incident to e in C. Facets f^{i-1} and f^{i+1} of $\mathcal{F}_e$ are adjacent to facet f^i, and f^{i-1} precedes, while f^{i+1} follows, facet f^i.

The atomic unit on which queries are formulated is called a *facet-edge pair*. This is a pair consisting of a facet f and an edge e, such that f and e are incident. The *edge component* e of a is denoted e_a, and the *facet component* f of a is denoted f_a. The facet-edge pair a determines two rings in C, these being edge-ring $\mathcal{E}_{f_a}$ and facet-ring $\mathcal{F}_{e_a}$. There are four versions of a which derive from the two directions that each of its two rings can assume. Henceforth by facet-edge pair we mean one such version — each of the two rings determined by the facet-edge pair has direction. $\mathcal{E}_a$ denotes the edge-ring $\mathcal{E}_{f_a}$ with direction determined by a; facet-ring $\mathcal{F}_a$ is similarly defined.

Given facet-edge pair a, it is useful to distinguish between the two vertices incident to e_a (its endpoints), and between the two polyhedra incident to f_a. To distinguish between the endpoints, observe that the direction of $\mathcal{E}_a$ directs edge e_a in a natural way. We call that vertex which serves as endpoint both to e_a and to the edge that precedes e_a in $\mathcal{E}_a$, the *origin* of e_a, denoted $a\mathit{Org}$. Similarly, the *destination* $a\mathit{Dest}$ of e_a is that vertex incident both to e_a and to the edge that follows e_a in $\mathcal{E}_a$.

To distinguish between the polyhedra incident to f_a, we assume e_a to possess an orientation. Where its orientation is RH (LH), we define H_a^+ to be that open half-space determined by the plane *aff* f_a from which the direction of $\mathcal{E}_a$ appears counter-clockwise (clockwise). We then define the *positive polyhedron* of a, denoted $a\mathit{Ppos}$, to be that polyhedron p of C incident to f_a for which points of the interior of p arbitrarily close to the relative interior of f_a lie in H_a^+. The *negative polyhedron* $a\mathit{Pneg}$ of a is the other polyhedron of C incident to f_a. If facet f_a lies on the boundary of a ball complex so the positive (or negative) polyhedron of a does not exist, $a\mathit{Ppos}$ (or $a\mathit{Pneg}$) has value p^∞. Figure 1 illustrates some of the notions presented so far in this section.

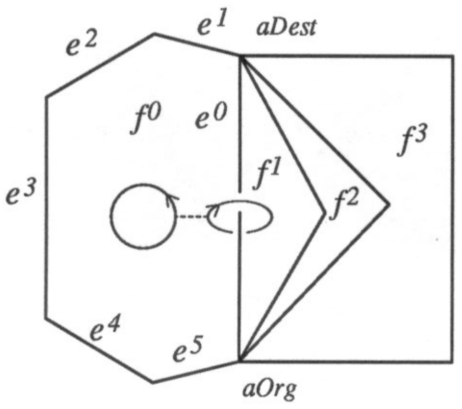

Fig. 1. We call this a handcuff diagram. It pictures a region of some complex. The "handcuff" represents facet-edge pair a. The placement and direction of its circular loop indicates the clocked facet component f_a, and its elliptical loop the space-oriented edge component e_a. In this example, $\mathcal{F}_a = (f^0 \dots f^3)$ and $\mathcal{E}_a = (e^0 \dots e^5)$, where $f_a = f^0$ and $e_a = e^0$. Polyhedron $a\mathit{Ppos}$ lies above the page and contains facets f^0 and f^1, while $a\mathit{Pneg}$ lies behind the page and contains f^0 and f^3.

We are now able to define the traversal functions *Fnext*, *Enext*, *Rev* and *Clock*. Each is applied to some facet-edge pair and returns a new facet-edge pair.

Fnext is defined by $a' = a\mathit{Fnext}$ where $e_{a'} = e_a$ and facet $f_{a'}$ follows f_a in the facet-ring $\mathcal{F}_a$. The rings of a are directed so that $\mathcal{F}_{a'} = \mathcal{F}_a$ and $a'\mathit{Org} = a\mathit{Org}$.

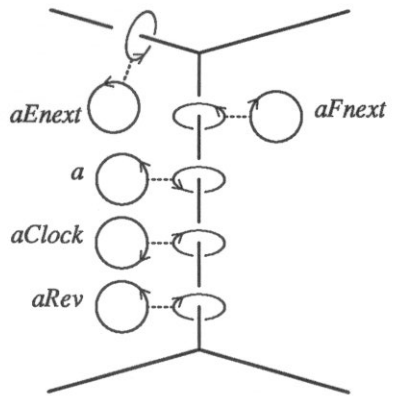

Fig. 2. This handcuff diagram illustrates the four traversal functions *Clock*, *Rev*, *Fnext* and *Enext*. The region pictured is a winged-edge, consisting of five edges and (part of) two facets (to the left and right of the vertically drawn line). We assume these two facets to belong to a common polyhedron that lies behind the plane of the page, this being a*Ppos* in this figure.

Enext is defined by $a' = aEnext$ where $f_{a'} = f_a$ and edge $e_{a'}$ follows edge e_a in the edge-ring $\mathcal{E}_a$. The rings of a' are directed so that $\mathcal{E}_{a'} = \mathcal{E}_a$ and $a'Ppos = aPpos$. Observe that a and $aEnext$ necessarily have the same orientation.

Rev is defined by $a' = aRev$ where a' and a are different versions of the same facet-edge pair — that is, $e_{a'} = e_a$ and $f_{a'} = f_a$ — for which the direction of $\mathcal{F}_{a'}$ is opposite that of $\mathcal{F}_a$, and the directions of $\mathcal{E}_{a'}$ and $\mathcal{E}_a$ are the same.

Clock is defined by $a' = aClock$ where a' and a are different versions of the same facet-edge pair, for which the directions of $\mathcal{E}_{a'}$ and $\mathcal{F}_{a'}$ are opposite those of $\mathcal{E}_a$ and $\mathcal{F}_a$, respectively.

Figure 2 illustrates these various traversal functions Traversal functions *Rev* and *Clock* can be viewed as follows. Given a, assume f_a to possess a clocking and e_a a (space)-orientation — call these the clocking and orientation of a. We insist upon the invariants that the direction of $\mathcal{E}_a$ agree with a's clocking, and that the direction of $\mathcal{F}_a$ move from $aPneg$ towards $aPpos$. The effect of *Rev* is to reverse orientation but leave clocking the same; since a's positive polyhedron is $aRev$'s negative polyhedron (and vice versa), the direction of $\mathcal{F}_{aRev}$ is opposite that of $\mathcal{F}_a$. The effect of *Clock* is to flip clocking but leave orientation the same; a's clocking is opposite $aClock$'s clocking, and a and $aClock$'s positive and negative polyhedra are swapped, so the directions of $\mathcal{E}_{aClock}$ and $\mathcal{F}_{aClock}$ are opposite those of $\mathcal{E}_a$ and $\mathcal{F}_a$, respectively. Each of the four versions of a facet-edge pair has a unique clocking and orientation. The clocking and orientation of a facet-edge pair are used as handles to manipulate the direction of its two rings.

The following relations hold among the traversal functions.

(A1) $aRev^2 = a$
(A2) $aClock^2 = a$
(A3) $aRev\,Clock = aClock\,Rev$
(A4) $aFnext^{-1} = aClock\,Fnext\,Clock$
(A5) $aFnext^{-1} = aRev\,Fnext\,Rev$
(A6) $aEnext^{-1} = aClock\,Enext\,Clock$
(A7) $aRev\,Enext = aEnext\,Rev$
(A8) $aClock\,Fnext^i \neq a$ for any i
(A9) $aRev\,Enext^i \neq a$ for any i
(A10) $aClock\,Enext^i \neq a$ for any i
(A11) $aRev\,Fnext^i \neq a$ for any i
(A12) $a \in C$ iff $aFnext \in C$
(A13) $a \in C$ iff $aClock \in C$
(A14) $a \in C$ iff $aRev \in C$

3.2 Space-Duality

The traversal function *Sdual* is applied to a facet-edge pair a of complex C, and returns a second facet-edge pair $aSdual$ belonging to C^*. The edge component of $aSdual$ is $e_{aSdual} = {f_a}^*$, and its facet component is

$f_{aSdual} = e_a{}^*$. In order to define the particular version of $aSdual$ — that is, the direction of its two rings — we first extend the notion of space-duality to facet- and edge-rings.

Given edge-ring $\mathcal{E}_a = (e_a^0 \ldots e_a^{n-1})$ of C, its space-dual is the facet-ring $(\mathcal{E}_a)^* = (e_a^{0*} \ldots e_a^{n-1*})$ of C^*. The space-dual of a facet-ring is similarly defined. The rings of $aSdual$ are then directed such that

$$\mathcal{E}_{aSdual} = (\mathcal{F}_a)^*, \text{ and}$$
$$\mathcal{F}_{aSdual} = (\mathcal{E}_a)^*.$$

The relation between a and $aSdual$ can be grasped by imagining the two facet-edge pairs superimposed, edge e_a piercing facet f_{aSdual} orthogonally, and facet f_a pierced by edge e_{aSdual} orthogonally. Edge e_{aSdual} is directed from $aPneg$ towards $aPpos$. Facet-ring $\mathcal{F}_{aSdual}$ moves from $aOrg$ towards $aDest$, so $aDest$ is the space-dual of $aSdualPpos$. Facet-edge pairs a and $aSdual$ necessarily have the same orientation. This is depicted in Figure 3.

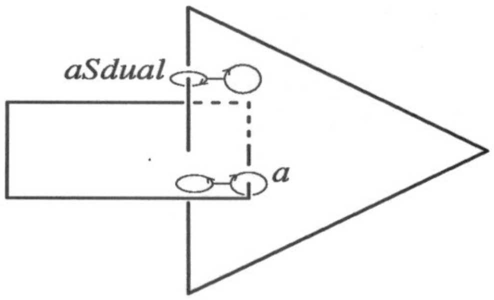

Fig. 3. This diagram depicts the relation between facet-edge pairs a and $aSdual$. Facet f_a is a square protruding from the page, and so appears foreshortened.

The following relations hold between $Sdual$ and the other traversal functions.

(A15) $aSdual^2 = a$
(A16) $aClockSdual = aSdualClock$
(A17) $aRevSdual = aSdualClockRev$
(A18) $aFnext = aSdualEnextSdual$
(A19) $aEnext = aSdualFnextSdual$
(A20) $a \in C$ iff $aSdual \in C$

Relation (A17) indicates that a change in the direction of $\mathcal{F}_a$ in C corresponds to a change in the direction of $\mathcal{E}_{aSdual}$ in C^*. Relation (A18) indicates that the rings $\mathcal{F}_a$ and $\mathcal{E}_{aSdual}$ move in the same direction. Relation (A19) defines *Enext* in terms of *Fnext* and *Sdual*, so maintaining the facet-rings in both C and C^*, as well as the correspondence between each cell and its space-dual, is sufficient to maintain the edge-rings in both complexes. The facet-edge data structure uses this fact.

3.3 The Vertex Functions

It is worthwhile to characterize every vertex of C and C^* in terms of the class of facet-edge pairs for which the vertex serves as origin. Where facet-edge pair a belongs to C, the origin and destination of a can be represented by two such classes. Also, the positive and negative polyhedra of a can be represented by two such classes corresponding to vertices of C^*.

We define an *origin partition* to be a partition of the set of facet-edge pairs comprising C and C^*. The equivalence class $aOrg$ is the class of facet-edge pairs b whose origin is identical to the origin of a. Class $aOrg$ may be known by numerous names, since $bOrg = aOrg$ if $b \in aOrg$. Each vertex of C and C^* is represented by a unique class of the partition.

Dest, *Ppos* and *Pneg* are intended to provide even more names for the classes of the origin partition. Formally, these additional names are superfluous; however, they coincide with our notions of destination vertex, and positive and negative polyhedra, and so are useful. For instance, $aDest$ corresponds to the destination vertex of the directed edge e_a, and is in some contexts more suggestive than $aClockOrg$ (another name for the same class).

The classes of the origin partition are related as follows.

(B1) $aOrg$, $aDest$, $aPpos$ and $aPneg$ are all distinct

(B2) $a Rev Org = a Org$
(B3) $a Clock Org = a Dest$
(B4) $a Rev Clock Org = a Dest$
(B5) $a Fnext^i Org = a Org$ for all integer i

(B6) $a Rev Ppos = a Pneg$
(B7) $a Clock Ppos = a Pneg$
(B8) $a Rev Clock Ppos = a Ppos$
(B9) $a Enext^i Ppos = a Ppos$ for all integer i

(B10) $a Sdual Org = a Pneg$
(B11) $a Sdual Dest = a Ppos$

4. The Facet-Edge Data Structure

Facet-edge pairs are partitioned into groups of eight. Each group consists of the four clocked and oriented versions of a facet-edge pair, plus the four versions of the facet-edge pair's space-dual. Where $\overline{a}$ is the canonical facet-edge pair of the group, the group is then of the form $\overline{a} Sdual^d Clock^c Rev^r$ where $d, c, r \in \{0,1\}$.

A group is represented by a *facet-edge node* n, which is a 2×2 matrix of structures. Element $n[d,c]$, called a *quarter-node*, corresponds to the two facet-edge pairs $\overline{a} Sdual^d Clock^c Rev^r$ where $r \in \{0,1\}$. A facet-edge pair is represented by a tuple (n, d, c, r, x), called a *facet-edge reference.* Components n, d and c address quarter-node $n[d,c]$, one of whose two corresponding facet-edge pairs is being referenced. Component r has value 0 if the facet-edge pair being referenced has the same orientation as the canonical facet-edge $\overline{a}$ (that is, $\overline{a} Sdual^d Clock^c$), and value 1 if its orientation is reverse that of $\overline{a}$ (that is, $\overline{a} Sdual^d Clock^c Rev$). Component x has value 0 if the facet-edge pair being referenced has RH orientation, and value 1 if it has LH orientation. Note that components r and x have different significance: component r describes the orientation of the facet-edge pair being referenced with respect to the orientation of the canonical facet-edge pair of the group to which it belongs, whereas component x describes its orientation in absolute terms.

The quarter-node $n[d,c]$ has two fields: *data* and *next.* Field *data* holds non-topological information which depends upon the application, and does not concern us. Field *next* contains the facet-edge reference to facet-edge pair $\overline{a} Sdual^d Clock^c Fnext$. Informally, the first row of matrix n contains a reference to each of the two facets of $\mathcal{F}_{\overline{a}}$ adjacent to facet $f_{\overline{a}}$ — a circular-list representing facet-ring $\mathcal{F}_{\overline{a}}$ is effectively threaded through the node. The second row of n contains a reference to each of the two facets of $\mathcal{F}_{\overline{a} Sdual}$ adjacent to facet $f_{\overline{a} Sdual}$. This facet-ring corresponds (under duality) to the edge-ring $\mathcal{E}_{\overline{a}}$, so a circular-list representing this edge-ring is effectively threaded through the node.

The traversal primitives are defined in terms of this data structure as follows (where all addition in this section is computed modulo 2):

$$
\begin{aligned}
(n,d,c,r,x) Fnext &= (n[d, c+r].next) Clock^r Rev^r \\
(n,d,c,r,x) Sdual &= (n, d+1, c+x, r, x) \\
(n,d,c,r,x) Clock &= (n, d, c+1, r, x) \\
(n,d,c,r,x) Rev &= (n, d, c, r+1, x+1)
\end{aligned}
$$

Two remarks concern these definitions. First, observe that

$$(n,d,c,0,x) Fnext = (n[d,c].next)$$

and

$$
\begin{aligned}
(n,d,c,1,x) Fnext &= (n[d,c+1].next) Clock Rev \\
&= (n, d, c+1, 0, x+1) Fnext Clock Rev \\
&= (n, d, c, 0, x+1) Clock Fnext Clock Rev \\
&= (n, d, c, 0, x+1) Fnext^{-1} Rev.
\end{aligned}
$$

The latter relation indicates that moving around $\mathcal{F}_a$ is the same as moving backwards around $\mathcal{F}_{a Rev}$. Second, the use of component x accommodates relation (A17): $a Rev Sdual = a Sdual Clock Rev$, which is equivalent to $a Rev Sdual Rev Clock Sdual = a$ by (A1, A2, A15). Thus

$$\begin{aligned}
&(n,d,c,r,x) RevSdualRevClockSdual \\
&\qquad = (n,d,c,r+1,x+1) SdualRevClockSdual \\
&\qquad = (n,d+1,c+x+1,r+1,x+1) RevClockSdual \\
&\qquad = (n,d+1,c+x+1,r,x) ClockSdual \\
&\qquad = (n,d+1,c+x,r,x) Sdual \\
&\qquad = (n,d,c,r,x)
\end{aligned}$$

The reader can confirm that the remaining relations (A1-20) are satisfied by this implementation, assuming the next field of each quarter-node is correct.

The vertex functions are implemented by any data structure for representing and maintaining a partition — in this case, the origin partition. A vertex function call, say *aPpos*, is first translated to the call *a′Org* according to one of the relations

$$\begin{aligned}
&aDest = aClockOrg, \\
&aPpos = aSdualClockOrg, \\
&aPneg = aSdualOrg.
\end{aligned}$$

The class of the origin partition that contains facet-edge pair a' is then sought. Associated with this class is information pertaining to vertex *a′Org*, such as its location in R^3 or properties of the polyhedron *aPpos* that is its space-dual.

5. Primitive Construction Operators

In this section we present the primitive construction operators *make_facet_edge*, *splice_facets*, *splice_edges* and *transfer*. The first operator obtains and initializes a new facet-edge node, and returns a facet-edge reference to one of the eight facet-edge pairs represented by the node. Operators *splice_facets* and *splice_edges* are used to modify the facet- and edge-rings of a complex. Operator *transfer* is used to change incidence relations involving vertices and polyhedra, by modifiying the origin partition.

Two caveats accompany these operators. First, no class of complexes is closed with respect to these operators: their use does not guarantee that complexes are produced. Operator *make_facet_edge* does not, in fact, create a complex at all — edge e_a of the facet-edge pair a it returns is incident to facet f_a and to no other facet, and so does not belong to the boundary of a polyhedron. Furthermore, little imagination is needed in using *splice_facets* or *splice_edges* to create the most exquisite garbage. Second, these primitives are not easy to use in constructing complexes of complexity. The reader need not be vexed. In section 6, we define higher-level operators in terms of these primitives which make the task of construction quite feasible (if not also easy).

To help describe these primitives, we introduce some notation for manipulating rings. The notation allows us to describe the manipulation of complexes in terms of the essentially one dimensional manipulation of rings. Let $\Phi = (a_1 \ldots a_m)$ and $\Phi' = (a_{m+1} \ldots a_n)$ be two rings with all a_i distinct. Then $concat(\Phi, \Phi')$ represents the ring

$$concat(\Phi, \Phi') = (a_1 \ldots a_n).$$

The operation $split(\Phi, a_p)$ represents the pair of rings

$$split(\Phi, a_p) = \big((a_1 \ldots a_{p-1}), (a_p \ldots a_m) \big)$$

where $0 < p \leq m+1$. Operations *first* and *second* are used to access the first and second rings of the pair $split(\Phi, a_p)$, respectively. Furthermore, rings Φ and Φ' are equivalent, denoted $\Phi \equiv \Phi'$, if they represent the same cycle of elements — that is, where $|\Phi| = |\Phi'| = n$, there exists an integer j such that, for each $1 \leq i \leq n$, the i^{th} element of Φ is identical to the $(i+j)^{th}$ element of Φ', modulo n. By convention, $\mathcal{F}_a$ denotes the ring $\mathcal{F}_a = (aFnext^0\ aFnext^1 \ldots aFnext^{n-1})$ where $|\mathcal{F}_a| = n$. $\mathcal{E}_a$ is similarly defined.

5.1 Make_facet_edge

Construction primitive *make_facet_edge* returns a facet-edge reference to a new (canonical) facet-edge $\overline{a}$. The relations (A) and (B) of section 3 hold among the eight facet-edge pairs $\overline{a}Sdual^d Clock^c Rev^r$, where $d, c, r \in \{0, 1\}$.

Primitive *make_facet_edge* is implemented as follows. Operation *make_facet_edge(orientation)* obtains a free node n. Quarter-node $n[d,c]$ is assigned the facet-edge reference $(n,d,c,0,x)$ where x has value 0 if argument *orientation* is RH and value 1 if *orientation* is LH, for $d,c \in \{0,1\}$.

5.2 Splice_facets

The operation $splice\_facets(a,b)$ takes as arguments two facet-edge pairs, and returns no value. The operation affects the facet-rings $\mathcal{F}_a$ and $\mathcal{F}_b$ as follows:

(a) if the two rings are distinct, it combines them into one ring;
(b) if the rings are identical, it breaks the ring into two distinct rings.

The arguments determine where the facet-rings are to be cut and joined. In rings $\mathcal{F}_a$ and $\mathcal{F}_b$, the cuts occur immediately after facets f_a and f_b, respectively. If the two rings are distinct, the distinct edges e_a and e_b are coalesced into one edge, and the two rings combined at the cuts. If the two rings are identical, the edge e_a $(= e_b)$ is cleaved lengthwise into two new edges, and each serves as pivot to one of the two new facet-rings resulting from the cuts. The operator is illustrated in Figure 4.

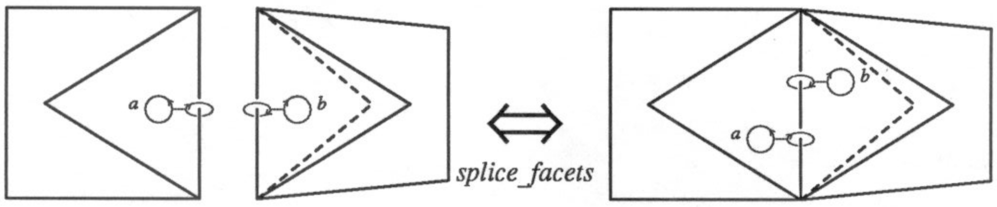

Fig. 4. This diagram illustrates the effect operator *splice_facets* has upon facet-rings.

The operation can be viewed as a way of replacing certain facet-rings with others.

```
$splice\_facets(a,b)$
  {
  if $(\mathcal{F}_a \equiv \mathcal{F}_b)$
    replace $\mathcal{F}_a$ by the two rings $split(\mathcal{F}_{aFnext}, bFnext)$;
  else
    replace $\mathcal{F}_a$ and $\mathcal{F}_b$ by $concat(\mathcal{F}_{aFnext}, \mathcal{F}_{bFnext})$;
  }
```

Operation $splice\_facets(a,b)$ is accomplished by interchanging the value of $aFnext$ with $bFnext$. The operation affects the $Fnext$ relation in complexes C_a and C_b (where by C_a we mean the complex to which cells f_a and e_a belong). Let $\overline{Fnext}$ denote the $Fnext$ relation immediately after the operation is performed. Where $\alpha = aFnextClock$ and $\beta = bFnextClock$, relations $Fnext$ (immediately before the operation) and $\overline{Fnext}$ are related as follows.

(C1) $a\overline{Fnext} = bFnext$
(C2) $b\overline{Fnext} = aFnext$
(C3) $\alpha\overline{Fnext} = \beta Fnext$
(C4) $\beta\overline{Fnext} = \alpha Fnext$
(C5) $aClockRev\overline{Fnext} = \beta Rev$
(C6) $bClockRev\overline{Fnext} = \alpha Rev$
(C7) $\alpha ClockRev\overline{Fnext} = bRev$
(C8) $\beta ClockRev\overline{Fnext} = aRev$
(C9) $\gamma\overline{Fnext} = \gamma Fnext$ for all other facet-edge pairs γ.

The implementation for *splice_facets* is quite simple. In the following, a is represented by facet-edge reference (n, d, c, r, x) and b by (n', d', c', r', x'). We assume the (last four) quarter-node assignments are performed simultaneously; in practice, some temporary variables would be used as is customary when swapping values.

```
$splice\_facets\big((n,d,c,r,x), (n',d',c',r',x')\big)$
  {
  $(\nu, \delta, \kappa, \rho, \chi) \leftarrow aFnext\,Clock;$
  $(\nu', \delta', \kappa', \rho', \chi') \leftarrow bFnext\,Clock;$
  $n[d, c+r].next \leftarrow bFnext\,Clock^{r}\,Rev^{r};$
  $n'[d', c'+r'].next \leftarrow aFnext\,Clock^{r'}\,Rev^{r'};$
  $\nu[\delta, \kappa+\rho].next \leftarrow bClock^{\rho+1}Rev^{\rho};$
  $\nu'[\delta', \kappa'+\rho'].next \leftarrow aClock^{\rho'+1}Rev^{\rho'};$
  }
```

Correctness of implementation is shown by proving that relations (C) hold. Below we show that the assignments above result in (C1), (C3), (C5) and (C7). The remaining relations (except (C9)) are shown similarly, while (C9) follows because the implementation affects no more than four quarter-nodes.

$$\begin{aligned}
a\overline{Fnext} &= \big(n, d, c, r, x\big) && (C1)\\
&= n\big[d, c+r\big].next\\
&= bFnextClock^{r}Rev^{r}Clock^{r}Rev^{r}\\
&= bFnext.\\
\alpha\overline{Fnext} &= \big(\nu, \delta, \kappa, \rho, \chi\big) && (C3)\\
&= \nu\big[\delta, \kappa+\rho\big].nextClock^{\rho}Rev^{\rho}\\
&= bClock^{1-\rho}Rev^{\rho}Clock^{\rho}Rev^{\rho}\\
&= \beta Fnext.\\
aClockRev\overline{Fnext} &= \big(n, d, c, r, x\big) && (C5)\\
&= \big(n, d, c+1, r, x+1\big)\overline{Fnext}\\
&= \big(n\big[d, c+r\big].next\big)Clock^{r}Rev^{r}\\
&= bFnextClockRev\\
&= \beta Rev.\\
\alpha ClockRev\overline{Fnext} &= \big(\nu, \delta, \kappa, \rho, \chi\big) && (C7)\\
&= \big(\nu, \delta, \kappa, \rho, \chi\big)ClockRev\overline{Fnext}\\
&= \big(\nu, \delta, \kappa+\rho+1, \rho+1, \chi+\rho+1\big)\overline{Fnext}\\
&= \big(\nu\big[\delta, \kappa+\rho\big].next\big)Clock^{\rho+1}Rev^{\rho+1}\\
&= bRev.
\end{aligned}$$

5.3 Splice_edges

The operation *splice_edges*(a, b) takes as arguments two facet-edge pairs, and returns no value. The operation modifies the edge-rings $\mathcal{E}_a$ and $\mathcal{E}_b$ as follows:

(a) if the two rings are distinct, it combines them into one ring;
(b) if the rings are equivalent, it breaks the ring into two rings;

As with *splice_facets*, the arguments to *splice_edges* determine where edge-rings are to be cut and joined. In rings $\mathcal{E}_a$ and $\mathcal{E}_b$, cuts occur immediately after edges e_a and e_b, respectively. Figure 5 illustrates the effect of *splice_edges*.

The operator can be seen as replacing certain edge-rings with others.

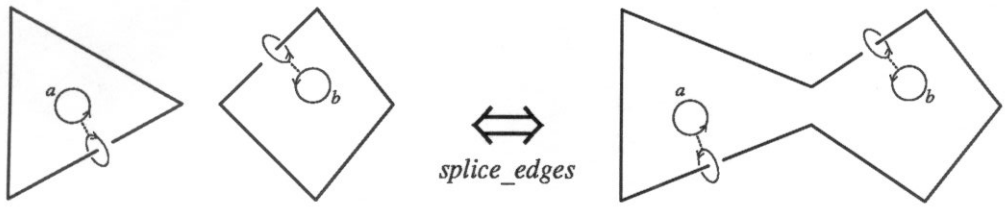

Fig. 5. This diagram illustrates the effect operator *splice_edges* has upon edge-rings.

```
splice_edges(a, b)
  {
  if ($\mathcal{E}_a \equiv \mathcal{E}_b$)
    replace $\mathcal{E}_a$ by the two rings $split(\mathcal{E}_{aEnext}, bEnext)$;
  else
    replace $\mathcal{E}_a$ and $\mathcal{E}_b$ by $concat(\mathcal{E}_{aEnext}, \mathcal{E}_{bEnext})$;
  }
```

Operation *splice_edges*(a, b) affects the *Enext* relation in complexes C_a and C_b (or equivalently, the *Fnext* relation in C_a^* and C_b^*). Let $\overline{Enext}$ denote the *Enext* relation immediately after the operation. Where $\alpha = aEnextClock$ and $\beta = bEnextClock$, *Enext* (before the operation) and $\overline{Enext}$ are related as follows.

(D1) $a\overline{Enext} = bEnext$
(D2) $b\overline{Enext} = aEnext$
(D3) $\alpha\overline{Enext} = \beta Enext$
(D4) $\beta\overline{Enext} = \alpha Enext$
(D5) $aRev\overline{Enext} = bRevEnext$
(D6) $bRev\overline{Enext} = aRevEnext$
(D7) $\alpha Rev\overline{Enext} = \beta RevEnext$
(D8) $\beta Rev\overline{Enext} = \alpha RevEnext$
(D9) $\gamma\overline{Enext} = \gamma Enext$ for all other facet-edge pairs γ.

As we might expect, an edge-ring of one complex can be modified by operating on the corresponding facet-ring of the dual complex. Indeed, *splice_edges* is implemented in terms of *splice_facets*.

```
splice_edges(a, b)
  {
  splice_facets(aDual, bDual);
  }
```

To show correctness of this implementation, it suffices to show that the (D) relations are satisfied by *splice_facets*$(aDual, bDual)$. Below we show this for (D2), (D4), (D6) and (D8); the remaining relations (except for (D9)) are shown similarly, whereas (D9) holds since *splice_facets* affects no more than four quarter-nodes.

Operation *splice_facets*$(aDual, bDual)$ establishes the following relations, where $\overline{Fnext}$ denotes the *Fnext* relation immediately after the operation.

(C2') $bDual\overline{Fnext} = aDualFnext$
(C4') $bDualFnextClock\overline{Fnext} = aDualClock$
(C6') $bDualRevClock\overline{Fnext} = aRevDualFnext$
(C8') $bDualFnextRev\overline{Fnext} = aDualRev$

Relation (Ci'), which derives from relation (Ci) of section 5.2, is then used in showing (Di) below.

$$\begin{aligned}
(D2)\qquad b\overline{Enext} &= bDual\overline{Fnext}Dual \\
&= aDualFnextDual \\
&= aEnext
\end{aligned}$$

$$\begin{aligned}
(D4)\qquad \beta\overline{Enext} &= bEnextClock\overline{Enext} \\
&= bDualFnextDualClockDual\overline{Fnext}Dual \\
&= bDualFnextClock\overline{Fnext}Dual \\
&= aDualClockDual \\
&= aClock \\
&= \alpha Fnext
\end{aligned}$$

$$\begin{aligned}
(D6)\qquad bRev\overline{Enext} &= bRevDual\overline{Fnext}Dual \\
&= bDualRevClock\overline{Fnext}Dual \\
&= aRevDualFnextDual \\
&= aRevEnext
\end{aligned}$$

$$\begin{aligned}
(D8)\qquad \beta Rev\overline{Enext} &= bEnextClockRev\overline{Enext} \\
&= bDualFnextDualClockRevDual\overline{Fnext}Dual \\
&= bDualFnextRev\overline{Fnext}Dual \\
&= aDualRevDual \\
&= aRevClock \\
&= \alpha EnextRev
\end{aligned}$$

5.4 Modifying the Origin Partition

By each operator's definition, neither *splice_facets* nor *splice_edges* affects incidence relations involving vertices and polyhedra. For instance, when *splice_facets* is used to split a single facet-ring $\mathcal{F}_e$, thereby cleaving edge e length-wise, the two resulting edges share a common origin and destination vertex. Higher-level construction operators generally need to modify incidence relations involving vertices and polyhedra. To permit this, we introduce the operator *transfer*.

Let P be a partition of some universe U, let A be an equivalence class of P, and let $B \subset U$. The operation *transfer*(A, B) modifies partition P by transfering each element $b \in B$ from its respective equivalence class *class*(b) into A.

```
transfer(A, B)
  {
  for each $b \in B$ {
    $class(b) \leftarrow class(b) - b$;
    $A \leftarrow A \cup b$
    }
  }
```

If $A = \emptyset$, then B becomes an equivalence class in the new partition. If B is an equivalence class of P, then *transfer*(A, B) simply forms the union of the two classes.

The usefulness of *transfer* becomes evident if we recall that each vertex and polyhedron is represented by an equivalence class of the origin partition (over the facet-edge pairs that comprise a complex and its space-dual). Incidence relations are implied by the partition. For instance, vertex v is an endpoint of edge e iff for some facet-edge pair a, we have $aOrg = v$ where $e_a = e$. If this is the case, it follows that $aRevOrg = v$ also, and that none of the six remaining versions of a have origin v. Edge e can then be given a new endpoint $v' = bOrg$ by the operation *transfer*$(bOrg, \{a, aRev\})$.

6. Manipulating Individual Polyhedra

It is worthwhile to be able to manipulate the individual polyhedra of a complex that is represented by the facet-edge structure. First, we would permit the traversal of the combinatorial boundary of an arbitrary polyhedron, while ignoring the rest of the complex that contains the polyhedron. Such traversal should be accomplished using functions appropriate for moving around a 2-dimensional subdivision — the facet-edge functions are too general to be appropriate. In this section, we reduce each of a small set of such functions to the facet-edge functions. The set of functions we choose to work with are the edge (traversal) functions of the quad-edge structure.

Second, we would permit the construction of the facet-edge representation of a single (connected) polyhedron. The construction should be accomplished using operators appropriate to the task — the use of the facet-edge operators would be overkill. In this section, we reduce the edge (construction) operators of the quad-edge structure to the facet-edge operators. Using these edge operators implemented in terms of the facet-edge operators, polyhedra can be built that are represented by the facet-edge structure. Each such polyhedron can be regarded as a primitive ball complex.

Third, we would permit two polyhedra to be glued together along a polygon of each. The complexes to which each belongs would thus be combined, or modified if they are one and the same. With the *meld* operator primitive ball complexes built with the edge operators can be combined to form non-trivial complexes.

6.1 Traversing the Boundary of a Polyhedron

In this subsection we concern ourselves with traversal in the combinatorial boundary ∂p of an arbitrary polyhedron p belonging to complex C or to C^*. We first briefly present as background the elements of the quad-edge structure that we will need. The presentation is intended as a reminder to the reader, and at places applies only to traversal of *orientable* surfaces; the reader is encouraged to read [GS] if not already familiar with this seminal work. We then present an *edge representation scheme* whereby an edge $e \in \partial p$, viewed as a cell of the 2-dimensional complex ∂p, can be represented in terms of the facet-edge structure that models C and C^*. The edge representation scheme is then used as a basis for describing each edge function in terms of its affect upon the facet-edge structure.

6.1.1 The edge functions

Where p is a polyhedron of complex C, the 2-complex $Q = \partial p$ is a subdivision of the sphere. Given edge $e \in Q$, the surface-orientation (with respect to Q) and direction of e can be chosen independently, so there are four surface-oriented, directed versions of e. We write $\hat{e}_p$ to denote any such version, or simply $\hat{e}$ when polyhedron p is known by context. Edge $e \in C$ is said to *underlie* edge $\hat{e}$ in C.

The direction of $\hat{e} \in Q$ determines the edge's vertex of origin ($\hat{e}\,Org$) and vertex of destination ($\hat{e}\,Dest$), in the natural way. In addition, the surface-orientation — what we have been calling clocking — and direction of $\hat{e}$ together determine the edge's left polygon ($\hat{e}\,Left$) and right polygon ($\hat{e}\,Right$). Specifically, where Q is coherently oriented under the clocking of $\hat{e}$, $\hat{e}\,Left$ is that polygon of Q incident to $\hat{e}$ whose clocking agrees with $\hat{e}$'s direction; $\hat{e}\,Right$ is the other polygon of Q incident to $\hat{e}$. The clocking of the cells $\hat{e}\,Org$, $\hat{e}\,Dest$, $\hat{e}\,Left$ and $\hat{e}\,Right$ are taken by definition to agree with that of $\hat{e}$. Note that $\hat{e}\,Org \neq \hat{e}\,Dest$ and $\hat{e}\,Left \neq \hat{e}\,Right$ since p is a polyhedron.

There are three primitive edge functions — *Flip*, *Sym* and *Onext* — in terms of which the remaining edge functions of the quad-edge structure (except for *Dual*) are defined. The flipped version $\hat{e}\,Flip$ of edge $\hat{e}$ has clocking opposite that of $\hat{e}$, but the two edges have the same direction. The symmetric version $\hat{e}\,Sym$ of $\hat{e}$ has direction opposite that of $\hat{e}$, but the edges have the same clocking. Furthermore, considering the cycle of edges (in Q) incident to $\hat{e}\,Org$, we define $\hat{e}\,Onext$ to be that edge that immediately follows $\hat{e}$ in that cycle, where the direction of the cycle is induced by the clocking of $\hat{e}$.

The *dual* of $Q = \partial p$ is defined to be a 2-complex Q^* obtained from Q by interchanging vertices and polygons, and which preserves incidence relations. The dual of edge $\hat{e} \in Q$ is an clocked and directed edge $\hat{e}\,Dual \in Q^*$, for which

(E1) $\hat{e}\,DualDual = \hat{e}$
(E2) $\hat{e}\,DualSym = \hat{e}\,SymDual$
(E3) $\hat{e}\,DualFlip = \hat{e}\,FlipSymDual$
(E4) $\hat{e}\,DualOnext = \hat{e}\,OnextSymDual$

(This definition of dual is equivalent to that of [GS] where $\hat{e}Lnext$, the edge following $\hat{e}$ in $\hat{e}Left$, is defined by $\hat{e}Lnext = \hat{e}SymOnext^{-1}$). *Dual* is extended to vertices and polygons by defining $(\hat{e}Org)Dual = \hat{e}DualLeft$ and $(\hat{e}Left)Dual = \hat{e}DualOrg$. *Dual* establishes a correspondence between the vertices (edges, polygons) of Q, and the polygons (edges, vertices) of Q^*.

Since $\hat{e}$ and $\hat{e}Dual$ have opposite clockings, it is convenient to define a rotated version $\hat{e}Rot$ of $\hat{e}$, given by

$$\hat{e}Rot = \hat{e}FlipDual = \hat{e}DualFlipSym.$$

Edge $\hat{e}Rot$ is the dual of $\hat{e}$, directed from $\hat{e}Right$ to $\hat{e}Left$, and clocked so that moving around $\hat{e}Right$ corresponds to moving around $\hat{e}RotOrg$.

To later describe how a subdivision may be modified, it is convenient to define $\hat{e}Org$ to be the ring of edges in Q incident to $\hat{e}$'s vertex of origin. More formally, $\hat{e}Org$ is the cycle under *Onext* of e. Polygon $\hat{e}Left$ is defined in terms of the ring of edges in Q^* incident to the vertex dual to $\hat{e}$'s left polygon — $\hat{e}Left$ is defined to be the ring $\hat{e}OnextRotOrg$.

6.1.2 The edge representation scheme

Let p be a polyhedron of complex C or C^*, where C and C^* are represented by the facet-edge structure. Where primal edge $\hat{e}_p \in \partial p$ and $d \in \{0,1\}$, edge $\hat{e}_p Dual^d$ is represented by the pair $\langle a, d\rangle$, called an *edge reference*. The first component is a reference to the facet-edge pair a, determined by the following:

(i) edge e_a underlies $\hat{e}_p$ in C
(ii) $f = \hat{e}_p Left$
(iii) the clocking of a coincides with the clocking of $\hat{e}_p Left$
(iv) $aPpos = p$

The second component d, called a *duality bit*, has value 0 (1) iff the edge being represented is primal (dual), and is identical to the exponent d of $\hat{e}_p Dual^d$. The scheme is depicted in Figure 6.

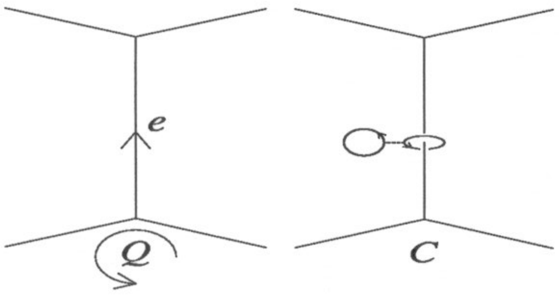

Fig. 6. This diagram illustrates the edge-reference scheme. The winged-edge corresponds to a region of $Q = \partial p$, where p lies behind the page. The facet-edge pair a depicted by the handcuff is such that edge $\hat{e}$ is given by $\langle a, 0\rangle$, and $\hat{e}Dual$ by $\langle a, 1\rangle$.

Each edge $\hat{e}_p Dual^d$, as p ranges over the polyhedra of $C \cup C^*$ and $d \in \{0,1\}$, is uniquely and unambiguously represented by an edge reference. Given edge $\hat{e}_p Dual^d$, conditions (i) and (ii) uniquely determine the components of a, then (iii) determines the clocking of a, then (iv) the clocking of a. On the other hand, given $\langle a, d\rangle$, consider first the edge that $\langle a, 0\rangle$ represents. Conditions (i) and (iv) together determine $\hat{e}_p$, then (iii) determines the clocking of $\hat{e}_p$, then (ii) the direction of $\hat{e}_p$. Finally, since $\hat{e}_p$ is unambiguously represented by $\langle a, 0\rangle$ and edge $\hat{e}_p Dual$ is well-defined, $\langle a, d\rangle$ unambiguously represents edge $\hat{e}_p Dual^d$.

6.1.3 Implementation of the edge functions

One purpose of the edge representation scheme is to enable the traversal of the boundary of polyhedron p using the underlying facet-edge structure. Each edge function can be described in terms of how it affects an edge reference. More precisely, for edge function Op, there exists a sequence of facet-edge functions Op' for which

$$\langle a, d\rangle Op = \langle aOp', d'\rangle.$$

The following characterizes each edge operator in this fashion.

(F1) $\langle a, 0\rangle Flip = \langle aFnextRev, 0\rangle$
(F2) $\langle a, 0\rangle Sym = \langle aFnextClock, 0\rangle$
(F3) $\langle a, 0\rangle Onext = \langle aEnext^{-1}FnextClock, 0\rangle$
(F4) $\langle a, d\rangle Dual = \langle a, 1-d\rangle$
(F5) $\langle a, 1\rangle Flip = \langle aRevClock, 1\rangle$
(F6) $\langle a, 1\rangle Sym = \langle aFnextClock, 1\rangle$
(F7) $\langle a, 1\rangle Onext = \langle aEnext^{-1}, 1\rangle$

The correctness of this scheme can be verified by showing that the edge operators so characterized possess the properties stated in [GS, section 2.3]. For instance, where $\hat{e}$ is represented by $\langle a, 0\rangle$, we have

$$\begin{aligned}\hat{e}Flip^2 &= \langle a, 0\rangle FlipFlip\\ &= \langle aFnextRevFnextRev, 0\rangle\\ &= \langle a, 0\rangle\\ &= \hat{e}.\end{aligned}$$

The derivation of (F1)–(F7) is straightforward. Figure 7 pictorially motivates equations (F1)–(F3). Equation (F4) follows from the edge representation scheme. Equations (F5)–(F7) follow from the relations developed so far in this section. For instance, (F5) is derived as follows:

$$\begin{aligned}\langle a, 1\rangle Flip &= \langle a, 0\rangle DualFlip && (F4)\\ &= \langle a, 0\rangle FlipSymDual && (E3)\\ &= \langle aFnextRev, 0\rangle SymDual && (F1)\\ &= \langle aFnextRevFnextClock, 0\rangle Dual && (F2)\\ &= \langle aRevClock, 0\rangle Dual\\ &= \langle aRevClock, 1\rangle. && (F4)\end{aligned}$$

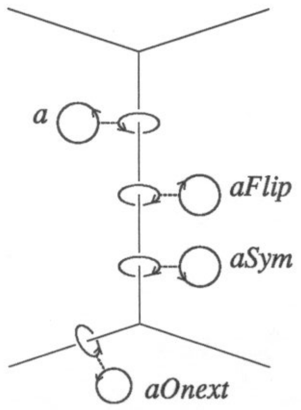

Fig. 7. This diagram illustrates the use of facet-edge pairs to represent directed, clocked edges in the boundary Q of a polyhedron. Each handcuff stands for facet-edge pair a where $\langle a, 0\rangle$ represents the directed, clocked edge $\hat{a}Op$ with which the handcuff is labelled.

6.2 Constructing a Polyhedron

A polyhedron can be characterized by its combinatorial boundary, this being a 2-dimensional subdivision of the sphere. A facet-edge structure representing a single polyhedron is most easily created and modified by manipulating the polyhedron's boundary. We choose the edge operators of the quad-edge structure as the means of performing these manipulations. These edge operators handle (in particular) the class of *open subdivisions* of the sphere, of which the (closed) subdivisions may be regarded as a special case. The construction of a polyhedron involves using the edge (construction) operators to incrementally build open subdivisions until one is produced which coincides with the boundary of the target polyhedron. In this section, we describe the effect each edge operator has upon the facet-edge structure by giving an implementation of each operator in terms of the facet-edge operators.

During the construction of polyhedron p, open subdivision Q is maintained under the edge-representation scheme. It is designated the *primal* subdivision, and its cells, and only its cells, may eventually belong to the target ∂p.

6.2.1 Open subdivisions

An *open k-cell* (for our purposes) is an open subspace of the sphere S^2 homeomorphic to R^k. An *open complex* S of S^2 is a finite collection of open cells of S such that

(i) the cells of S are pairwise disjoint,
(ii) for each cell $c \in S$, $bd\, c$ is the union of elements of S, and
(iii) if $c, d \in S$ and $cl\, c \cap cl\, d \neq \emptyset$, then $cl\, c \cap cl\, d$ is the union of elements of S.

Here $cl\, c$ denotes the closure of cell c. An open complex whose union is S^2 is an *open subdivision* (of the sphere).

Let S be an open subdivision such that for each cell $c \in S$, the closure $cl\, c$ is a (closed) cell. S corresponds to a (closed) subdivision of the sphere, obtained by replacing each cell $c \in S$ by its closure $cl\, c$. To build a polyhedron, the edge operators are applied successively to construct new elementary open subdivisions, and to combine and modify existing open subdivisions. The process proceeds until an open subdivision is produced that corresponds to (the subdivision identical to) the boundary of the target polyhedron.

6.2.2 Elementary open subdivisions of a sphere

There are two elementary subdivisions of the sphere. The first consists of a single edge $\hat{e}$ that is not a loop, and is denoted S_e (subscript e stands for 'edge'). Where $\hat{e} \in S_e$ is some arbitrary (but fixed) clocked and directed edge, we have $\hat{e}\,Org \neq \hat{e}\,Dest$ and $\hat{e}\,Left = \hat{e}\,Right$. The following properties hold in S_e:

(G1) $\hat{e}\,Onext = \hat{e}$
(G2) $\hat{e}\,Sym\,Onext = \hat{e}\,Sym$
(G3) $\hat{e}\,Flip\,Onext = \hat{e}\,Flip$
(G4) $\hat{e}\,FlipSym\,Onext = \hat{e}\,FlipSym$

The other elementary open subdivision of the sphere consists of a single edge $\hat{e}'$ that is a loop, and is denotes S_ℓ (subscript ℓ stands for 'loop'). S_ℓ is dual to open subdivision S_e; there exists a version of $\hat{e}'$ for which edges $\hat{e}'$ and $\hat{e}\,Sdual$ represent identical open subdivisions. Since $\hat{e}'$ is a loop, we have $\hat{e}'\,Org = \hat{e}'\,Dest$ and $\hat{e}'\,Left \neq \hat{e}'\,Right$. Writing $\hat{e}\,Dual$ for $\hat{e}'$, the following properties hold in S_ℓ:

(G5) $\hat{e}\,DualOnext = \hat{e}\,DualSym$
(G6) $\hat{e}\,DualSym\,Onext = \hat{e}\,Dual$
(G7) $\hat{e}\,DualFlip\,Onext = \hat{e}\,DualFlipSym$
(G8) $\hat{e}\,DualFlipSym\,Onext = \hat{e}\,DualFlip$

The operator *make_edge* builds a data structure representing both S_e and S_ℓ, and returns an edge reference to one version of S_e's (non-loop) edge. Open subdivision S_e is primal. Its implementation is given as follows.

```
make_edge()
  {
  a ← make_facet_edge(LHO);
  b ← make_facet_edge(LHO);
  splice_facets(a, b);
  splice_edges(a, bClock);
  return( ⟨a, 0⟩ );
  }
```

The operation $\hat{e} \leftarrow$ *make_edge*() obtains two new facet-edge nodes. In one of the nodes, it designates a facet-edge pair a for which $\langle a, 0\rangle$ represents $\hat{e}$, while in the other node it designates facet-edge pair b for which $\langle b\,Rev, 0\rangle$ represents $\hat{e}\,Flip$. Operation *splice_facets*(a, b) results in

(i) $a\,Fnext = b$
(ii) $b\,Fnext = a$,

while operation *splice_edges*$(a, b\,Clock)$ results in

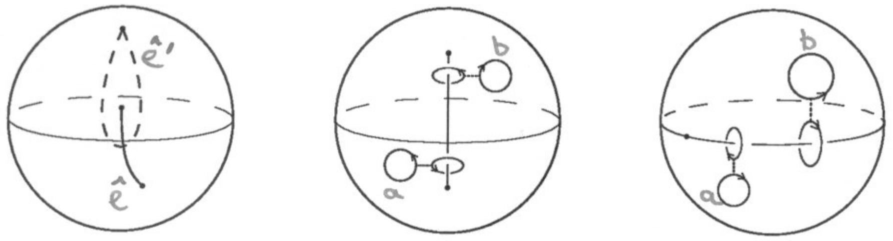

Fig. 8. This diagram depicts the open subdivisions under the edge-reference scheme. The left figure depicts edges $\hat{e} \in S_e$ and $\hat{e}' \in S_\ell$, superimposed on the same sphere to suggest how they are related. The center figure depicts the facet-edge representation for $\hat{e}$ where S_e is primal (constructed by *make_edge*), while the right figure depicts the facet-edge representation for loop $\hat{e}'$ where S_ℓ is primal (constructed by *make_loop*).

(iii) $aEnext = bClock$
(iv) $aEnext^{-1} = bClock$.

Edge $\hat{e}$ under the edge-representation scheme is depicted in Figure 8.

The relations (i)–(iv) ensure that $\hat{e} \leftarrow$ *make_edge*() does indeed build a facet-edge structure representing both S_e and S_ℓ, and that edge $\hat{e}$ is represented by $\langle a, 0\rangle$. This is verified by showing that relations (G1)–(G8) are satisfied. For instance, (G1) is shown as follows:

$$\begin{aligned}
\hat{e}Onext &= \langle a, 0\rangle Onext \\
&= \langle aEnext^{-1} FnextClock, 0\rangle \\
&= \langle bClockFnextClock, 0\rangle && (iv) \\
&= \langle bFnext^{-1}, 0\rangle \\
&= \langle a, 0\rangle && (i) \\
&= \hat{e}.
\end{aligned}$$

The operator *make_loop* also builds a facet-edge structure representing both S_e and S_ℓ, but *it* returns an edge-reference to that version of S_ℓ's loop that corresponds to $\hat{e}Dual$ (where $\hat{e} \leftarrow$ *make_edge*()). Open subdivision S_ℓ is primal. Its implementation is given as follows.

```
make_loop()
  {
  $a \leftarrow$ make_facet_edge(RHO);
  $b \leftarrow$ make_facet_edge(RHO);
  splice_facets(a, b);
  return( $\langle a, 0\rangle$ );
  }
```

The operation $\hat{e}Dual \leftarrow$ *make_loop*() obtains two new facet-edge nodes. In one node it designates facet-edge pair a for which $\langle a, 0\rangle$ represents edge $\hat{e}Dual$, and in the other node a facet-edge pair b for which $\langle bRev, 0\rangle$ represents $\hat{e}DualFlip$. Operation *splice_facets*(a, b) results in

(i) $aFnext = b$
(ii) $bFnext = a$

while the absence of a call to *splice_edges* results in

(iii) $aEnext = a$
(iv) $bEnext = b$.

Edge $\hat{e}Dual$ under the edge-representation scheme is depicted in Figure 8.

6.2.3 Modifying open subdivisions

The operator *splice* is used to modify open subdivisions. The operation $splice(\hat{a},\hat{b})$ takes as arguments two edges $\hat{a}$ and $\hat{b}$, and returns no value. The operation affects the two rings $\hat{a}Org$ and $\hat{b}Org$ and, independently, the two rings $\hat{a}Left$ and $\hat{b}Left$. In each case, if the two rings are distinct, *splice* combines them into one ring; and if the two rings are identical, *splice* breaks it into two distinct rings. The arguments $\hat{a}$ and $\hat{b}$ determine where the rings will be cut and joined. For rings $\hat{a}Org$ and $\hat{b}Org$, the cuts occur immediately after $\hat{a}$ and $\hat{b}$; for rings $\hat{a}Left$ and $\hat{b}Left$, the cuts occur immediately after $\hat{a}OnextRot$ and $\hat{b}OnextRot$.

Operation $splice(\hat{a},\hat{b})$ is performed by interchanging the values of $\hat{a}Onext$ with $\hat{b}Onext$, and $\hat{\alpha}Onext$ with $\hat{\beta}Onext$, where $\hat{\alpha} = \hat{a}OnextRot$ and $\hat{\beta} = \hat{b}OnextRot$. More formally, where $\overline{Onext}$ denotes the *Onext* relation immediately after the operation, $splice(\hat{a},\hat{b})$ establishes the following relations between $\overline{Onext}$ and *Onext*:

(H1) $\hat{a}\overline{Onext} = \hat{b}Onext$
(H2) $\hat{b}\overline{Onext} = \hat{a}Onext$
(H3) $\hat{\alpha}\overline{Onext} = \hat{\beta}Onext$
(H4) $\hat{\beta}\overline{Onext} = \hat{\alpha}Onext$
(H5) $\hat{\gamma}\overline{Onext} = \hat{\gamma}Onext$ for all other edges $\hat{\gamma} \in Q \cup Q^*$.

Operation $splice(\hat{a},\hat{b})$ is implemented in terms of the facet-edge operator *splice_edges* as follows.

```
splice( $\langle a,d\rangle, \langle b,d\rangle$ )
  {
  if ($d = 0$)
    splice_edges($aClockRev, bClockRev$);
  else
    splice_edges($aEnext^{-1}, bEnext^{-1}$);
  }
```

The duality bit d of the two arguments to *splice* are assumed to be identical — $splice(\hat{a},\hat{b})$ is defined only if $\hat{a}$ and $\hat{b}$ are both primal, or both dual.

To show the correctness of the implementation, let $\overline{Onext}$ ($\overline{Enext}$) denote the *Onext* (*Enext*) relation immediately after $splice(\hat{a},\hat{b})$, given primal edges $\hat{a}$ and $\hat{b}$ represented by $\langle a,0\rangle$ and $\langle b,0\rangle$. Operation $splice\_edges(aClockRev, bClockRev)$ establishes

(i) $a\overline{Enext}^{-1} = bEnext^{-1}$
(ii) $b\overline{Enext}^{-1} = aEnext^{-1}$
(iii) $aEnext^{-1}ClockRev\overline{Enext}^{-1} = bClockRev$
(iv) $bEnext^{-1}ClockRev\overline{Enext}^{-1} = aClockRev$

Relations (i), (ii), (iii) and (iv), which follow from the (D) relations of section 5.3, are used to show that values have been correctly swapped. To show (H1), we have

$$\begin{aligned}
\hat{a}\overline{Onext} &= \langle a,0\rangle\overline{Onext} \\
&= \langle a\overline{Enext}^{-1}FnextClock, 0\rangle \\
&= \langle bEnext^{-1}FnextClock, 0\rangle && (i) \\
&= \langle b,0\rangle Onext \\
&= \hat{b}Onext.
\end{aligned}$$

Similarly (ii) is used to show relation (H2). To show relation (H3), we have

$$\begin{aligned}
\hat{\alpha}\overline{Onext} &= \langle a, 0\rangle OnextRot\overline{Onext} \\
&= \langle aEnext^{-1}FnextClockFnextRev, 1\rangle\overline{Onext} \\
&= \langle aEnext^{-1}ClockRev, 1\rangle\overline{Onext} \\
&= \langle aEnext^{-1}ClockRev\overline{Enext}^{-1}, 1\rangle \\
&= \langle bClockRev, 1\rangle \qquad\qquad (iii) \\
&= \langle bEnext^{-1}ClockRevEnext^{-1}, 1\rangle \\
&= \langle bEnext^{-1}ClockRev, 0\rangle DualOnext \\
&= \langle bEnext^{-1}FnextClockFnextRev, 0\rangle DualOnext \\
&= \langle b, 0\rangle OnextFlipDualOnext \\
&= \hat{b}OnextRotOnext \\
&= \hat{\beta}Onext.
\end{aligned}$$

Similarly (iv) is used to show relation (H4). Notice that *splice_edges*($aClockRev, bClockRev$) only modifies facet-rings of the complex dual to the complex to which $\hat{a}$ and $\hat{b}$ belong. Since each occurence of $Fnext$ in the derivations above apply only to facet-rings of the complex to which $\hat{a}$ and $\hat{b}$ belong, we have been free to assume (in the derivations) that $Fnext$ has not been changed by *splice_edges*; no $\overline{Fnext}$ is necessary in the derivations.

We have shown correctness of an implementation for *splice* when its arguments are primal edges. Assume now that *splice* is passed two dual edges $\hat{a}$ and $\hat{b}$. To show correctness of implemenation in this case, we note that operations *splice*($\hat{a}, \hat{b}$) and *splice*($\hat{\alpha}, \hat{\beta}$) are equivalent, where $\hat{\alpha} = \hat{a}OnextRot$ and $\hat{\beta} = \hat{b}OnextRot$. Since edges $\hat{\alpha}$ and $\hat{\beta}$ are primal, it is sufficient to show that *splice_edges*($\hat{\alpha}ClockRev, \hat{\beta}ClockRev$) — which implements *splice*($\hat{\alpha}, \hat{\beta}$) — and *splice_edges*($\hat{a}Enext^{-1}, \hat{b}Enext^{-1}$) are equivalent, an easy exercise.

6.3 Meld

The operator *meld* is used to glue a ball complex C_a to a second complex C_b. With its use one melds an n-sided polygon $f_a \in \partial C_a$ to an n-sided polygon $f_b \in C_b$, thereby locating ball complex C_a in the polyhedron $bPneg$ of C_b. More formally, it establishes the topological relations for

$$\mathcal{U}(C_a) \subset bPneg, \text{ and}$$
$$\mathcal{U}(C_a) \cap \mathcal{U}(\partial bPneg) = f_b.$$

A two-dimensional analogue of the situation is depicted in Figure 9.

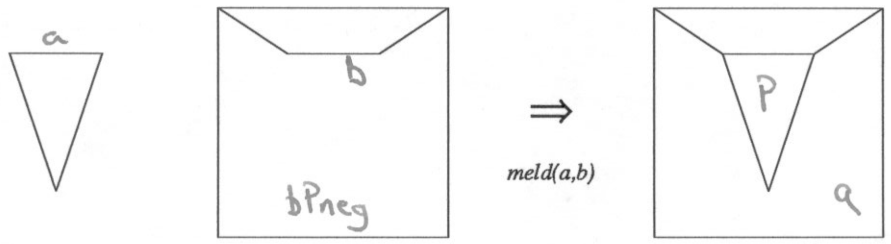

Fig. 9. This figure depicts a 2-dimensional analogue of the effect of *meld*. Edges of the figure correspond to facets, and polygons to polyhedra. Note that $C_a \neq C_b$ in the figure, but this need not be the case.

Let a and b be facet-edge pairs, and assume that

(i) $|f_a| = |f_b| = n$, and
(ii) $aPpos = p^{\infty}$.

Condition (ii) ensures that C_a is a ball complex, and that $f_a \in \partial C_a$. Where $a_i = aEnext^i$ and $b_i = bEnext^i$, $meld(a,b)$ identifies polygon f_a with f_b, and edge e_{a_i} with e_{b_i} for $0 \le i \le n-1$. The operation first coalesces distinct edge rings $\mathcal{E}_a$ and $\mathcal{E}_b$, forming a "pillow" consisting of the edges of $\mathcal{E}_a$ $(= \mathcal{E}_b)$ and the polygons f_a and f_b, and then removes polygon f_a from the complex. The two polyhedra that end up incident to f_b are $bPpos$ and p (which is essentially $aPneg$).

The boundary of polyhedron $aPneg$ is slightly changed to produce polyhedron p — facet f_a is replaced by f_b. In addition, polyhedra $aPpos$ $(= p^\infty)$ and $bPneg$ are combined to form a new polyhedron q, the effect of locating C_a inside $bPneg$. We have

$$facets\_of(p) = facets\_of(aPneg) - f_a + f_b, \text{ and}$$
$$facets\_of(q) = facets\_of(bPneg) \cup facets\_of(aPpos) - f_a - f_b.$$

Also, additional edges are made incident to each vertex $v_i = b_iOrg$ of f_b. To the edges already incident to v_i are added those edges of C_a incident to a_iOrg, less the edges of $\mathcal{E}_a$. We have

$$edges\_of(v_i) = edges\_of(b_iOrg) \cup edges\_of(a_iOrg) - e_{a_i} - e_{a_iEnext^{-1}}, \text{ for } 0 \le i \le n-1.$$

To build the facet-rings of the "pillow" formed by coalescing $\mathcal{E}_a$ and $\mathcal{E}_b$, facet-rings of C_a and C_b are combined as follows.

```
for $i = 0, \cdots, n-1$
  if $\mathcal{F}_{a_i} \not\equiv \mathcal{F}_{b_i}$
    replace $\mathcal{F}_{a_i}$ and $\mathcal{F}_{b_i}$ by $concat(\mathcal{F}_{a_iFnext}, \mathcal{F}_{b_iFnext})$;
```

Facet f_a is removed as follows.

```
for $i = 0, \cdots, n-1$
  replace $\mathcal{F}_{a_i}$ by $first(split(\mathcal{F}_{a_iFnext}, b_iFnext))$;
```

Operator *meld*, given in Figure 10, is implemented by a single loop in which the construction of the "pillow" and removal of f_a are interleaved. The facets incident to polyhedra p and q, and edges incident to vertices v_i, are specified by *transfer* operations. The necessary facet-ring manipulations are done with *splice_facets*. Note that the facet-edge pairs $a_iSdual^dClock^cRev^r$ (where $0 \le i \le n-1$,and $c, d, r \in \{0,1\}$) are effectively deleted from the data structure; the n facet-edge nodes representing these could be garbage-collected, or used again elsewhere.

```
meld(a, b)
  {
  firsta ← a;
  transfer(bPneg, aPpos);
  do {
    if($\mathcal{F}_a \not\equiv \mathcal{F}_b$)
      splice_facets(a, $bFnext^{-1}$);
    splice_facets(a, $aFnext^{-1}$);
    transfer(bOrg, aOrg);
    transfer(aPneg, {bSdual, bSdualRev});
    transfer($\emptyset$, {$aSdual^dClock^cRev^r | d, c, r \in \{0,1\}$});
    a ← aEnext
    b ← bEnext
    } until a = firsta;
  }
```

Fig. 10. Procedure *meld*.

7. Decomposing a Polyhedron

The process of partitioning a polyhedron into simpler constituent polyhedra is called *decomposition*. One reason for decomposing a polyhedron $\overline{p}$ is that $\overline{p}$ may possess properties that preclude certain algorithms from being applied to it — for instance, it may be non-convex, or possess cavities or handles. Sometimes the difficulty may be overcome by decomposing $\overline{p}$ into more amenable pieces, and then applying the algorithm to *these*[CD]. Alternatively, $\overline{p}$ may be well behaved but its volume might be too large to allow efficient solution of equations via finite element methods, and further decomposition may be desired [JB].

There are various strategies for performing decomposition. We will concern ourselves with an incremental strategy in which polyhedra are iteratively detached from the original polyhedron until nothing of the original remains. Each simpler piece split off from the original is not subject to further decomposition, and satisfies whatever "simplicity" criteria is required of the algorithm. Such an algorithm maintains a current polyhedron S (initially $\overline{p}$), and a current collection of constituent polyhedra C (initially $\emptyset$). The algorithm iteratively detaches a polyhedron p_i (in iteration i) from S and transfers it to C. The process stops when S represents a null polyhedron — collection C then represents the decomposition of $\overline{p}$. In the present section, we show how collection C assembled during the course of decomposition can be represented by the facet-edge structure. Each polyhedron detached from S is attached to C by meld operations. For simplicity, we assume that $\overline{p}$ is polyhedral in the sense we have been using the word (that is, having genus zero and no cavities), and that C always consists of zero or more ball complexes.

Wördenweber uses this incremental strategy in [Wö] to decompose a polyhedron into tetrahedra. He makes no attempt to assemble the pieces, but allows the sequence of operations by which they were detached to represent the resulting decomposition. Chazelle uses a more general divide-and-conquer strategy in [Ch] to decompose a polyhedron into convex parts — each time a polyhedron is split into two, *both* pieces are subject to further decomposition. He represents each complex created during the decomposition with collections of edge-to-facets, facet-to-edges, and vertex-adjacency pointers. Facet-edge operations that accommodate the divide-and-conquer strategy will be presented in [La].

We briefly describe Wördenweber's algorithm to make these notions more concrete, and to introduce new notions. We refer the reader to [Wö] for a description of how the algorithm selects a tetrahedron to be detached from the current polyhedron S in each iteration. The actual removal of the tetrahedron from S is accomplished by one of the four operators ***op0***, ***op1***, ***op2*** or ***op3***. Each ***opk*** modifies the polyhedron S to reflect the removal of the tetrahedron. The index k of ***opk*** indicates how the tetrahedron t that ***opk*** is designed to detach is connected to the rest of S: k is the number of triangular facets that connect t to the rest of S, while $4-k$ is the number of t's exposed facets (that is, belonging to the boundary of S). In modifying S, ***opk*** removes each of the $4-k$ exposed facets of t, a consequence of detaching t from S. In general, some of these facets correspond to facets of C (in fact, to facets of ∂C); where f is such a facet, $\tilde{f}$ denotes that facet of C to which f corresponds. Facet $\tilde{f}$ will have been created when S was modified in some earlier iteration (say iteration j where $j < i$); the polyhedron p_j transfered to C at that time contains $\tilde{f}$ which was then cleaved from f. The rest of the $4-k$ exposed facets of t belong to the boundary of the original polyhedron $\overline{p}$, and so correspond to no facets of C. Let S' denote S immediately after being modified by ***opk***. S' contains k new facets, the "connecting" facets of t; these facets are created by ***opk*** — again the consequence of deleting t from S — and replace the $4-k$ exposed facets removed by ***opk***. Each of the k facets revealed by ***opk*** corresponds to an exposed facet of the tetrahedron that has been added to C. Figure 11 illustrates the effect each ***opk*** has upon S.

A decomposition algorithm employing the incremental strategy requires the use of an operator ***op***, analogous to Wördenweber's ***opk*** operators, for transfering a polyhedron p_i from S to C. The operator must build a facet-edge representation for p_i, attach p_i to C (using calls to *meld*), and modify S (to reflect it's loss of p_i). The operator's most formidable task is in determining exactly how p_i is to be attached to C — that is, in determining the arguments to each of its calls to *meld*. To guide ***op*** in attaching p_i to C, each facet $f \in S$ possesses a *link pointer link*(f) which references that facet $\tilde{f} \in \partial C$ to which f corresponds. The facets of S that belong to the boundary of the original polyhedron $\overline{p}$ do not correspond to any facet of C, and so have null link pointers. To elaborate, in iteration i, ***op*** performs the following steps in succession:

(i) constructs a facet-edge representation for polyhedron p_i, to be transferred from S to C,
(ii) attaches p_i to C, thereby forming C' (to serve as C in the next iteration),
(iii) modifies S to reflect its loss of p_i, thereby forming S' (to serve as S in the next iteration), and

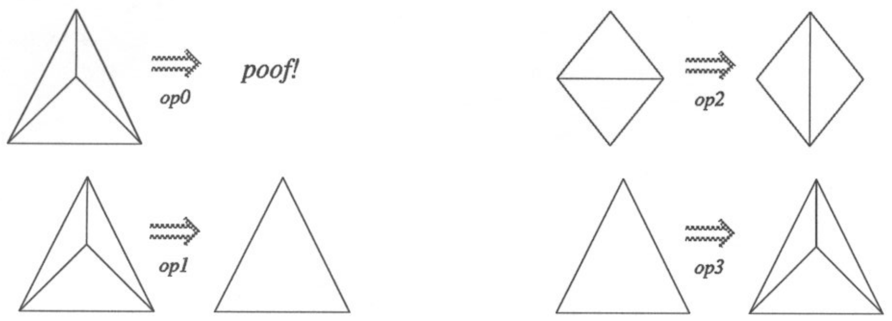

Fig. 11. This figure demonstrates how each *opk* locally modifies S to produce S'. Each drawing depicts a patch on the boundary of S or S'.

(iv) updates the link pointers of S'.

We do not elaborate on step (i); it is performed using the quad-edge operators, whose implementation in terms of the facet-edge operators was given in section 6. Assuming S is suitably represented — for concreteness we assume by the quad-edge structure — step (iii) also need not be treated. Presumably the description of p_i handed to *op* is adequate for *op* to perform steps (i) and (iii). Steps (ii) and (iv) do require elaboration. Henceforth denoting by p the polyhedron p_i constructed in step (i), we discuss in turn how we ascertain which facets of p are to be melded to C, how the link pointer is used to guide each meld operation, and how the link pointers are updated in S' to serve later iterations.

Consider the relation between p and S. That patch of p to be glued to C coincides with subcomplex $S_p = \bigcup\{star\, f | f \in S \text{ is removed by } op\}$. (Recall that *star f* is the complex consisting of the faces of cell f; in this case since f is a facet, it consists of f and the vertices and edges that bound f.) Subcomplex S_p is generally a patch of S, homeomorphic to a closed disk. (In the final iteration however, S itself is transfered to C, in which case $S_p = S$.) We denote by $\phi(c)$ that cell of p that coincides with cell $c \in S_p$. The mapping $\phi : S_p \rightarrow p$ is an isomorphism, not generally onto. Consider next the relation between p and S'. That patch of p that lies in ∂C (after *op* has attached p to C) coincides with subcomplex $S'_p = \bigcup\{star\, f | f \in S' \text{ is created by } op\}$. (At the last iteration however, $S' = S'_p = \emptyset$). We denote by $\phi'(c)$ that cell of p that coincides with cell $c \in S'_p$; the isomorphism $\phi' : S'_p \rightarrow p$ is not onto. The patches $\phi(S_p)$ and $\phi'(S'_p)$ cover polyhedron p. Their intersection $\phi(S_p) \cap \phi'(S'_p)$ is a vertex-edge cycle in p, called the *silohette* of p. These notions are depicted in Figure 12.

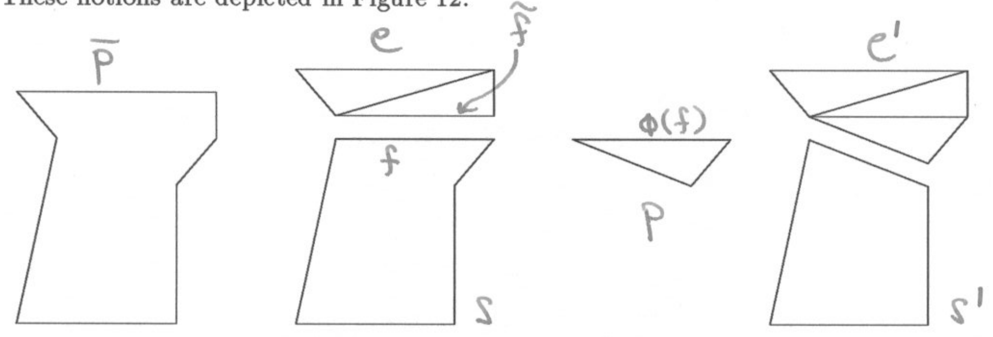

Fig. 12. This figure depicts a 2-dimensional analogue of the effect of *op*. Each edge of the figure corresponds to a facet, and each polygon to a polyhedron.

To attach p to C, for each facet $f \in S_p$, facet $\phi(f) \in p$ is melded to facet $\tilde{f} \in C$. Each facet of S_p is obtained by treating the dual 2-complex $S_p{}^*$ as a graph, and performing a search in $S_p{}^*$. Each vertex visited corresponds to a facet of S_p. The silohette of p is used to restrict the search to $S_p{}^*$, prohibiting it from passing into the rest of S^*. Specifically, the search algorithm considers two vertices adjacent iff the edge that connects them is not dual to a silohette edge of p. Pointer *link*(f) consists of two fields *edge* and *pair*, whose contents are as follows.

link(f).edge : a pointer to directed, oriented edge $e \in S$ such that $eLeft = f$.
link(f).pair : facet-edge pair b such that, where edge $e' \in \partial C$ is given by $\langle b, 0\rangle$, we have
(a) $e'Left = \tilde{f}$,
(b) $e'Lnext^i = \tilde{e}Lnext^i$ for all i, and
(c) $bPneg = p^\infty$.

When facet $f \in S_p$ is visited, facets $\phi(f)$ and $\tilde{f}$ are melded by the following operation.

```
**if** $link(f) \neq \emptyset$ {
  $e \leftarrow link(f).edge$;
  $a \leftarrow$ that facet-edge pair $a$ for which $\langle a, 0\rangle$ is $\phi(e)$;
  $b \leftarrow link(f).pair$;
  $meld(a, b)$;
  }
```

Edge $\phi(e) \in p$, required in the above block of code, is obtained by performing an identical graph search in p^*, coincident with the search in ${S_p}^*$.

Having attached p to C and modified S to produce S', the link pointers of S' must be updated. This involves setting the link pointer of each facet created by *op* (that is the facets of S'_p); the link fields of the other facets of S' are still correct. Much as before, we perform a graph search in ${S'_p}^*$ and a coinciding search in p^*, using the silohette of p to limit both searches. When we visit a vertex of ${S'_p}^*$, dual say to facet $f \in S'_p$, $link(f)$ is set by the following.

```
let $e$ be an edge for which $eLeft = f$;
$link(f).edge \leftarrow e$;
$link(f).pair \leftarrow a$ where $\phi(e)$ is represented by $\langle a, 0\rangle$;
```

The isomorphisms ϕ and ϕ' are each computed on the fly by performing identical searches in two distinct graphs. Each pair of searches must start at coinciding cells for each isomorphism to be correctly computed. To do this, we select some silohette edge e — since e belongs to both S_p and S'_p, it can be used to compute the starting point for both pairs of searches. Let $e \in S$ be oriented and directed so the $eLeft \in S_p$, and let $\phi(e) \in p$ be oriented and directed so that $\phi(e)Org = \phi(eOrg)$ and $\phi(e)Left = \phi(eLeft)$. The searches in ${S_p}^*$ and p^* then begin at vertices *eLeftDual* and $\phi(e)LeftDual$, respectively. To compute ϕ', we note that coinciding cells of S'_p and p have orientations that *disagree*: facets f and $\phi(f)$ *appear* to have the same orientation when viewed for instance from a point beyond f but beneath $\phi(f)$, say from the interior of a convex p. To determine the starting points for the searches in ${S'_p}^*$ and p^*, let edge e be oriented and directed as above. Facet $eLeft \in S$ is replaced by $eLeft \in S'$. The edge of S' coinciding with e is then $\phi(e)Flip$, so the searches of ${S'_p}^*$ and p^* begin at the vertices *eLeftDual* and $\phi(e)FlipLeftDual$, respectively. This is illustrated for Wördenweber's *op2* in Figure 13.

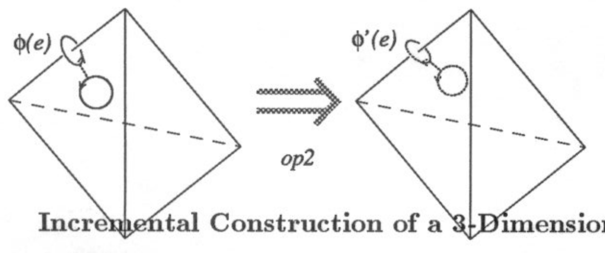

Fig. 13. This diagram depicts a tetrahedron t transfered by Wördenweber's *op2*. The tetrahedron is attached to C along the two facets behind the page, while the two facets in front of the page occur in ∂C.

8. Incremental Construction of a 3-Dimensional Delaunay Triangulation

We describe how to build the Delaunay triangulation $DT(S)$ of a set S of $n \geq 4$ points (called *sites*) of R^3, in general position. Since the facet-edge structure represents both a complex and its dual, the algorithm also serves to construct the Voronoi diagram of S. The strategy is to first construct some tetrahedron of $DT(S)$ - called a *D-tetrahedron* - to serve as an initial current complex C. C is then grown by iteratively discovering, constructing and melding a new D-tetrahedron to one or more triangular facets on the boundary of C, until

```
delaunay(S)
  {
  t ← an initial D-tetrahedron of S;
  $\mathcal{F}$ ← facet_edges_of(t);
  while ($\mathcal{F} \neq \emptyset$) {
    a ← some element of $\mathcal{F}$;
    t ← find_tetrahedron($f_a$, aPpos);
    if (t does not exist)
      $\mathcal{F} \leftarrow \mathcal{F} - a$;
    else {
      for each $\hat{a} \in$ facet_edges_of(t) {
        $a \leftarrow \mathcal{F}(\hat{a})$;
        if ($a \neq \emptyset$) {
          $\mathcal{F} \leftarrow \mathcal{F} - a$;
          a ← align($\hat{a}$, aClock);
          meld(a, $\hat{a}$);
          }
        else
          $\mathcal{F} \leftarrow \mathcal{F} + \hat{a}$;
        }
      }
    }
  }
```

Fig. 14. Procedure *delaunay*.

it is known that $C = DT(S)$. The algorithm is described in [AB], and under geometric inversion that maps S to a set of points S' on a 3-dimensional hypersphere in R^4 [Br], corresponds to the gift-wrapping method of [CK] for building the convex hull of S'. The process of finding an initial and subsequent D-tetrahedra is described in [AB], so we describe this only briefly in the next paragraph, before presenting the entire algorithm.

Assume triangle f of $DT(S)$ is on the boundary of complex C, and that the D-tetrahedron t incident to f is known. Operation *find_tetrahedron*(f, t) constructs the other D-tetrahedron t' adjacent to f (if it exists). Let $H_{f,t}$ denote that open half-space determined by *aff* f which does not contain t. The vertices that define t' are then the vertices of f together with site q, where $q \in H_{f,t}$ is that site for which the sphere determined by q and the vertices of f is of minimal radius. It is shown in [Bh] that the interior of this sphere contains no sites, hence t' is indeed a D-tetrahedron. If $S \cap H_{f,p}$ is empty, then f lies on the convex hull of S and t' does not exist.

An initial D-tetrahedron is found by first finding some triangular facet f on the convex hull of S by the method of [CK]. The D-tetrahedron adjacent to f is discovered using the strategy given above, where candidate sites q range over all sites (except for the three that determine f).

The algorithm *delaunay* of Figure 14 constructs the Delaunay triangulation $DT(S)$ of a finite set of sites S of R^3, in general position. The algorithm initializes the current complex C to contain a D-tetrahedron, then iteratively melds D-tetrahedra to C until, for every facet of C, a D-tetrahedron has been sought on both sides of the facet.

Let F denote the set of facets for which a D-tetrahedron has been sought on exactly one side of the facet. F consists of those facets belonging to the boundary of the current complex C, less those facets that have been determined to lie on the convex hull of S. Dictionary $\mathcal{F}$ contains the triangles of F; more precisely, it contains one facet-edge reference to a for each triangle f_a of F, where $aPneg = p^{\infty}$. $\mathcal{F}(a)$ performs a look-up in dictionary $\mathcal{F}$, returning that element of $\mathcal{F}$ whose determining vertices are $aOrg$, $aEnextOrg$ and $aEnext^2 Org$ if it exists, or $\emptyset$ if the dictionary contains no such element. A scheme for addressing the elements of $\mathcal{F}$ using (the indices of) the three vertices that determine its elements is easily concocted.

A tetrahedron t is represented by some facet-edge pair a such that $aPpos = t$. The set $facet\_edges\_of(t)$ contains one facet-edge pair $\hat{a}$ for each of the four triangular facets of ∂t, where $\hat{a}Ppos = t$; the set is easily derived by traversal from that facet-edge pair a that represents t. Finally, $align(a, b)$ denotes that facet-edge $bEnext^i$ for which $aOrg = bEnext^i Org$; the algorithm ensures that some such i exists for each use of $align$.

After we had submitted this paper, we learned that Dr. V. Rajan [Ra] at IBM had derived the 3-dimensional Voronoi diagram by a similar technique.

9. Conclusion

The applications presented here but scratch the surface of the data structure's potential uses. Future research includes the development and rederivation of applications that would markedly benefit from use of the structure. Two examples of these were mentioned in the introduction: a divide-and-conquer algorithm for constructing 3-dimensional Voronoi diagrams, and a scheme for modelling the motion of 3-dimensional polyhedra. Future research also includes completely characterizing the class of complexes the data structure can model, and developing sets of construction operators with respect to which various classes of complexes are closed.

References

[AB] D. Avis and B. K. Bhattacharya, "Algorithms for computing d-dimensional Voronoi diagrams and their duals", in *Advances in Computing Research*. Edited by F. P. Preparata, 1, JAI Press, 1983, pp. 159-180.

[Ba] B. G. Baumgart, "A polyhedron representation for computer vision", in *1975 National Computer Conference, AFIPS Conference Proceedings*, vol. 44, AFIPS Press, 1975, pp. 589-596.

[Bh] B. K. Bhattacharya, "Application of computational geometry to pattern recognition problems", Simon Fraser Univ. CS, Tech. Rep. 82-3 (1982).

[Br] K. Q. Brown, "Voronoi diagrams from convex hulls", Info. Proc. Lett. 9, 1979, pp. 223-228.

[BHS] I. C. Braid, R. C. Hillyard, and I. A. Stroud, "Stepwise construction of polyhedra in geometric modelling", in *Mathematical Methods in Computer Graphics and Design*, K. W. Brodlie, Ed., Academic Press, London, 1980, pp. 123-141.

[Ch] B. Chazelle, "Convex Partitions of Polyhedra", SIAM Journal of Computing, Vol. 13, No. 3, pp. 488-507.

[CD] B. Chazelle and D. P. Dobkin, "Detection is easier than computation", Proc. 12th ACM SIGACT Symposium, Los Angeles, May 1980, pp. 146-153.

[CK] D. R. Chand and S. S. Kapur, "An algorithm for convex polytopes", JACM 17(1), Jan. 1970, pp. 77-86.

[EW] C. M. Eastman and K. Weiler, "Geometric modeling using the Euler operators", Inst. of Physical Planning, Carnegie-Mellon Univ., Research Rep. 78 (Feb. 1979)

[GS] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams", ACM Trans. on Graphics, Vol. 4, No. 2, Apr. 1985, pp. 75-123.

[JB] A. Jameson and T. Baker, "Improvements to the aircraft Euler method", AIAA 25th Aerospace sciences meeting, paper AIAA-87-0452, 1987.

[La] M. J. Laszlo, "Manipulating Three-Dimensional Subdivisions", dissertation, Dept. of Computer Science, Princeton University, to appear.

[Ra] V. T. Rajan, private communication.

[Wö] B. Wördenweber, "Volume-triangulation", C. A. D. Group, University of Cambridge (1980).